

Mastering SEQUENCE

Excel's most amazing function with more than 200 examples



Meni Porat



Mastering SEQUENCE

*Excel's most amazing function with
more than 200 examples*

Meni Porat



www.bpbonline.com

Copyright © 2023 BPB Online

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor BPB Online or its dealers and distributors, will be held liable for any damages caused or alleged to have been caused directly or indirectly by this book.

BPB Online has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, BPB Online cannot guarantee the accuracy of this information.

First published: 2023

Published by BPB Online

WeWork

119 Marylebone Road

London NW1 5PU

UK | UAE | INDIA | SINGAPORE

ISBN 978-93-55518-545

www.bpbonline.com

Dedicated to

My beloved daughters

Noga

&

Gili

Who have patiently borne with me

In the last year

About the Author

Meni Porat has been working in the software industry for more than 20 years. He has played central roles in numerous projects as a systems analyst and project manager. He worked for various financial institutions: banks, credit card companies and insurance companies. Currently, he is a self-employed consultant and Excel & VBA instructor. The author is also very active on social media channels, especially Facebook and LinkedIn. On the latter, he has published during the last year more than 150 posts and articles which have been read by more than 400,000 people. During that period he also featured in four YouTube international webinars on Excel. As a tribute to his contributions to the international Excel community, the author has been awarded with the Most Valuable Professional (MVP) by Microsoft.

About the Reviewer

Cristiano Galvao is a Microsoft MVP from Brazil with over 2 decades of working experience with Microsoft Excel. He has been a guest speaker on the major Excel conferences abroad, and is the organizer of the Excel Weekend, the most important Excel conference in South America. Cristiano is the technical reviewer of many Excel books translated to Portuguese, including authors such as Bill Jelen, John Walkenbach, Paul McFedries, Ron Person, Joseph Schmuller, and more.

Acknowledgement

I would like to express my gratitude to my daughters for their support and understanding while their father was seized in a fit of writing momentum, leaving them only a small amount of patience and leisure.

I am extremely grateful to my dear friend, Dr. Isaac Gottlieb. Without his unwavering support and invaluable advice, this book would not have been written.

I wish to express my obligation to BPB Publications for their unstinted aid. I am indebted to the publisher who saw the potential in the book's subject, took the risk and made this book come true. The long enterprise has reached its successful goal.

Finally, a special thanks to many of my LinkedIn friends who read my numerous posts and responded enthusiastically. Their feedback greatly motivated me.

Preface

Excel is a ubiquitous sophisticated software. With the introduction of the new Dynamic Array Functions (DAF) in Excel 365, using Excel has become much simpler and extremely fast.

This book presents the *crème de la crème* of these new functions: SEQUENCE. This function is explained thoroughly in more than 200 examples accompanied by almost 300 pictures.

The book is designed to provide the most comprehensive guide to that function. You will be able to find examples of how to implement SEQUENCE (usually with collaboration with other functions) in almost any field imaginable: from text and numbers to finance and mathematics.

The vast number of practical examples is intended to guide the reader on how to implement this function cleverly in building solutions to challenges in Excel.

Throughout the book, the reader will encounter well-designed answers to problems in various levels of complexity. It is not necessary to read the book's chapters sequentially. Just pick randomly the chapters that interest you or those that are most relevant to your problems.

With this book, you will gain the knowledge and skills to become a proficient and efficient problem solver in Excel. We do believe that you will find this book informative, useful, and inspiring.

Chapter 1: A Short Introduction to Dynamic Array Functions in Excel 365 – Explains the new features in Excel 365. This is nothing short of a revolution in Excel: instead of a formula that returns a single-cell result, we now have an array of results in multiple cells. This introductory chapter illustrates the use of these functions in 30 examples. Some of the DAF were published only a few months ago.

Chapter 2: SEQUENCE in Text Operations – Every analyst (data/business/financial...) needs some very common text manipulation in order to clean his/her data and prepare it for analysis. This chapter supplies more than 60 examples of how to do it.

Chapter 3: Using SEQUENCE with Numbers – Easy-to-apply methods for creating a sequence of numbers (ascending or descending, positive or negative), duplicating a number (or a sequence of numbers), extracting numbers from a mixed string, etc. are showcased in this chapter.

Chapter 4: SEQUENCE in Arrays – As the name implies, Dynamic Array Functions are meant to deal with arrays of data, both in input and output. This chapter will show you several ways to build dynamic arrays, flip them vertically and horizontally, transpose arrays, fetch multiple results for a search value and more.

Chapter 5: SEQUENCE in Date and Time Operations - Dates and times calculations are very common in Excel. The SEQUENCE function will show you neat tricks to generate sequence of dates, create various dynamic monthly and yearly calendars, generate automatic schedules, produce lists of dates (for example: last day of each month of the year), calculate net workdays in a given period (with/without holidays) and many more.

Chapter 6: Financial Operations with SEQUENCE – This chapter will be of great interest for readers whose expertise is in finance: accountants, economists, financial analysts, CFOs etc. The chapter supplies a new approach to old solutions for financial functions: PMT, DB, NPV, PDURATION, RATE, IRR etc.

Chapter 7: SEQUENCE - The Ancilla of Math – Whether you are a mathematics teacher or not, here, you will find some fresh ideas of how to harness new approaches (sometimes with dynamic Excel charts to better illustrate the concept) to mathematical challenges in algebra, trigonometry and the number theory.

Chapter 8: SEQUENCE and Other Animals – Last but not least, this chapter is dedicated to some more complex examples. It demonstrates the powerful symbiosis and cooperation of SEQUENCE and other advanced functions. Several examples with the new versatile LAMBDA function are also given.

Coloured Images

Please follow the link to download the
Coloured Images of the book:

<https://rebrand.ly/21ud2u7>

We have code bundles from our rich catalogue of books and videos available at
<https://github.com/bpbpublications>. Check them out!

Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at :

errata@bpbonline.com

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.

Did you know that BPB offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.bpbonline.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at :

business@bpbonline.com for more details.

At www.bpbonline.com, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on BPB books and eBooks.

Piracy

If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at **business@bpbonline.com** with a link to the material.

If you are interested in becoming an author

If there is a topic that you have expertise in, and you are interested in either writing or contributing to a book, please visit **www.bpbonline.com**. We have worked with thousands of developers and tech professionals, just like you, to help them share their insights with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions. We at BPB can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about BPB, please visit **www.bpbonline.com**.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



Table of Contents

1. A Short Introduction to Dynamic Array Functions in Excel 365.....	1
Introduction.....	1
Structure.....	1
Objectives.....	3
Introducing Dynamic Array functions	3
Examples of the new DAF.....	3
<i>UNIQUE</i> function.....	4
<i>FILTER</i> function.....	4
<i>SORT</i> function.....	5
<i>SORTBY</i> function	6
<i>RANDARRAY</i> function	6
<i>SEQUENCE</i> function	7
<i>TEXTSPLIT</i> function	7
<i>TOCOL</i> function	8
<i>TOROW</i> function.....	9
<i>VSTACK</i> function	9
<i>VSTACK</i> versus <i>TOCOL</i>	10
<i>HSTACK</i> function	10
<i>EXPAND</i> function	11
<i>ARRAYTOTEXT</i> function.....	11
<i>TEXTBEFORE</i> function	12
<i>TEXTAFTER</i> function	12
<i>CHOOSECOLS</i> function.....	13
<i>CHOOSEROWS</i> function	14
<i>WRAPROWS</i> function	14
<i>WRAPCOLS</i> function.....	15
<i>XLOOKUP</i> function	15
<i>XMATCH</i> function	17

<i>TAKE function</i>	17
<i>DROP function</i>	18
<i>VALUETOTEXT function</i>	18
Conclusion	19
Points to Remember	19
2. SEQUENCE in Text Operations.....	21
Introduction.....	21
Structure.....	21
Objectives.....	24
Examples of SEQUENCE with text.....	24
<i>Finding the names of the 10 highest-paid employees</i>	24
<i>How many words are there in the cell (version 1)</i>	25
<i>How many words are there in the cell (version 2)</i>	25
<i>How many times does a string appear in the cell</i>	
Method 1 of 5.....	26
Method 2 of 5.....	26
Method 3 of 5.....	27
Method 4 of 5.....	27
Method 5 of 5.....	28
<i>Extract all characters - Horizontally</i>	28
<i>Extract all characters – Vertically</i>	29
<i>All uppercase English in one column</i>	29
<i>All uppercase English in one cell</i>	30
<i>Duplicate a sequence of characters</i>	30
<i>Duplicate a cell by a duplication factor</i>	31
Method 1.....	31
Method 2.....	32
<i>Creating English uppercase letters without knowing</i> <i>how many letters there are</i>	32
<i>Transpose without TRANSPOSE</i>	33
<i>Extract only first three letter of weekday names</i>	33
<i>Extract only digits from a string</i>	34

Method 1 of 3.....	34
Method 2 of 3.....	34
Method 3 of 3.....	35
Extract only unique Alphabetic characters from a string	35
Split numbers and text	36
Remove unwanted characters from string (2 named ranges as parameters)...	36
Remove unwanted characters from string (Formula)	37
How many times does a string appear in a range.....	37
Is it a Palindrome?.....	38
Add vendor to list (the table).....	38
Add vendor to list (the formula).....	39
Remove all digits from the string.....	39
Move first name from end of cell to the beginning	40
Reverse String.....	40
Method 1.....	40
Method 2.....	41
Sort Text in alphabetical order	41
How many words are there in the cell without the separator (SEQUENCE). 42	
How many words are there in the cell without the separator (TEXTSPLIT) . 42	
Off with their heads	43
Extract only digits and add a separator.....	43
How many lower-case letters are there in the cell	44
Method 1.....	44
Method 2.....	44
All Greek letters in one formula.....	45
Find last word in cell	
Method 1 of 3.....	45
Method 2 of 3.....	46
Method 3 of 3.....	46
Number of characters in cell (without the separator).....	47
Number of non-empty cells in a column	47
Strip leading and trailing digits	48

Method 1 of 2.....	48
Method 2 of 2.....	48
Increasing Text from end to start.....	49
Increasing Text from start to end	49
Hebrew Gematria (Formula)	50
Hebrew Gematria (Gtable – Translation table).....	50
Extract only Country Names	51
How many occurrences of a String starting from a certain position	52
Remove Diacritics from Hebrew words	52
Is it a Palindrome (Arabic)	53
Convert Hebrew letters into English letters	53
Fetch description of Nth item of a non-sorted Key	54
Extract letters only from a chosen language (Formula)	55
Extract letters only from a chosen language (Validation list)	55
Gematria in English	56
Method 1.....	56
Method 2.....	56
How many Words are there in a Range?.....	57
Extract only non-digits from String.....	58
Find Unicode value for any character in the string, no matter which language	58
Conclusion	59
Points to remember	59
3. Using SEQUENCE with Numbers	61
Introduction.....	61
Structure	61
Objectives.....	63
Examples of SEQUENCE with numbers.....	63
Five methods to generate 12 positive integers	63
Method 1.....	63
Method 2.....	64

Method 3.....	64
Method 4.....	65
Method 5.....	65
Five methods to generate 12 negative integers.....	66
Method 1.....	66
Method 2.....	66
Method 3.....	67
Method 4.....	68
Method 5.....	68
Descending SEQUENCE – Two methods.....	68
Method 1.....	69
Method 2.....	69
Duplicate cell horizontally.....	70
Duplicate cell vertically.....	70
Duplicate numbers.....	71
Creating a vertical SEQUENCE of numbers – Two methods.....	71
Find missing numbers in a list.....	73
Reverse a Number.....	73
Reverse a horizontal ascending array.....	74
Reverse a horizontal descending array.....	74
SEQUENCE of odd and even numbers.....	75
Sum all digits in a cell which has only digits.....	76
Sum all digits in a cell which has digits and text.....	76
Sum every Nth row.....	77
Sum the largest N numbers.....	77
Sum the smallest N numbers.....	78
Two tricks with SEQUENCE (ROW()).....	78
Create a SEQUENCE of n Rows starting from Row(n).....	78
SUM a virtual array created by SEQUENCE (ROW()).....	79
SUM SEQUENCE (virtual array).....	79
Alternate 1s and 0s.....	80

Dynamic SEQUENCE	81
Two methods to extract a number from the string's end	81
Method 1.....	81
Method 2.....	82
Two methods to extract a number from the string's start	82
Method 1.....	82
Method 2.....	83
Find N largest numbers (Ascending)	84
Find N largest numbers (Descending)	84
How many columns in a sheet.....	85
How many digits	85
Reverse numbers horizontally by a parameter.....	86
Reverse order of a SEQUENCE of numbers	87
Subject with the highest score.....	87
SEQUENCE based on number of unique values	88
Dynamic frequency based on dynamic bins.....	89
SEQUENCE column.....	90
SEQUENCE and COLUMNS	90
Building a chessboard in three steps.....	91
Chessboard - Step 1	91
Chessboard - Step 2	92
Chessboard - Step 3	92
Creating N-digit number with the same digit repeated N times	93
Conclusion	94
Points to remember	94
4. SEQUENCE in Arrays	95
Introduction.....	95
Structure	95
Objectives.....	96
Examples of SEQUENCE with arrays	96

<i>Creating an array of identical numbers - Two methods</i>	97
<i>Creating an array of ascending numbers – three methods</i>	97
<i>From one cell to a vertical array</i>	98
<i>How many active months</i>	99
<i>Build a dynamic array - horizontal or vertical</i>	100
<i>Flip columns horizontally</i>	100
Method 1.....	100
Method 2.....	101
<i>Flip vertical array (with and without SEQUENCE)</i>	102
<i>Flip part of vertical array using a parameter</i>	102
<i>Create a two-dimensional array using four parameters</i>	103
<i>Flexible LARGE</i>	104
<i>MMULT with static ranges and with dynamic Arrays</i>	104
<i>Four useful tricks with VLOOKUP</i>	105
VLOOKUP - Fetch all columns of an item searched	105
VLOOKUP - Fetch all data per lookup key in reverse order	105
VLOOKUP - Fetch last two columns for a search key	106
VLOOKUP - Fetch the first and third data items per lookup key	107
<i>From vertical to horizontal – Two Methods</i>	107
Method 1.....	107
Method 2.....	108
<i>Four two-dimensional arrays generated by two parameters</i>	109
<i>Select columns by parameters</i>	110
<i>Transpose a vertical array without knowing its size beforehand</i>	110
<i>Fetching multiple results for a search value</i>	111
<i>Vertical to horizontal without TRANSPOSE</i>	112
Conclusion	112
Points to remember	112
5. SEQUENCE in Date and Time Operations	113
Introduction.....	113
Structure	113

Objectives.....	115
Examples of SEQUENCE with date and time.....	115
12 months with each month's first day	115
The year's months with the last day of each month - three methods	116
Display the last date of each month of the year – method 1	116
Display the last date of each month of the year – method 2	116
Display the last date of each month of the year – method 3	117
N consecutive dates starting from today (two methods)	118
Method 1.....	118
Method 2.....	119
Display month names without a specific date	119
Two methods to display the weekday names	120
Adding minutes to time.....	121
Adding seconds to time.....	121
Sequence of days in a given month.....	122
A SEQUENCE of dates (between start and end dates).....	122
Extract only time –four traditional methods.....	123
Extract only time – a new method.....	124
Presence in class by month	125
How many Mondays are there in a given month	125
How many Saturdays between two dates?	126
Method 1.....	126
Method 2.....	127
Display only dates of Wednesdays in 2020	128
How many Wednesdays are there in 2020?.....	128
Sequence of the month's last date for each month (two methods).....	129
Method 1.....	129
Method 2.....	130
A horizontal SEQUENCE of descending dates - First of each month	130
A substitute for NETWORKDAYS.INTL (for a certain month)	131
The MonCal Named Range.....	132

<i>A substitute for the NETWORKDAYS.INTL (any period)</i>	133
<i>A substitute for NETWORKDAYS.INTL - with/without weekends</i>	133
<i>The Definition of MonCal</i>	134
<i>How many working days are there in each month of a given period?</i>	134
<i>How many eligibility days?</i>	135
<i>The doctor’s schedule (two versions)</i>	136
<i>The doctor’s schedule (version 1)</i>	136
<i>The doctor’s schedule (version 2)</i>	137
<i>Monthly calendar – classic versus non-classic</i>	137
<i>Monthly calendar - classic</i>	137
<i>Monthly calendar – non-classic</i>	139
<i>Monthly calendar in 20 languages</i>	139
<i>Monthly calendar in 20 languages – list of languages and formats</i>	140
<i>Monthly calendar in 20 languages – list of month numbers</i>	141
<i>Two methods for creating a list of the month’s days</i>	142
<i>Monthly calendar – a bad attitude</i>	142
<i>Monthly calendar – a good attitude</i>	142
<i>Yearly calendar – good versus bad</i>	143
<i>Dynamic yearly calendar – in one formula</i>	144
<i>Dynamic yearly calendar – Conditional Formatting</i>	144
<i>Dynamic yearly calendar - by month</i>	145
<i>Dynamic yearly calendar - by week</i>	145
<i>Yearly Horizontal calendar with highlighted weekday (two examples)</i>	146
<i>Yearly horizontal calendar</i>	147
<i>Example 1 (weekday chosen: Sunday)</i>	147
<i>Example 2 (weekday chosen: Saturday)</i>	147
<i>Yearly horizontal calendar - Conditional Formatting - calendar</i>	148
<i>Yearly horizontal calendar - Conditional Formatting - weekday names</i>	148
<i>Yearly vertical calendar with highlighted weekday (2 examples)</i>	149
<i>Yearly vertical calendar – example 1</i>	150
<i>Yearly vertical calendar – example 2</i>	150

<i>Yearly vertical calendar – Conditional Formatting - calendar</i>	151
<i>Yearly vertical calendar – Conditional Formatting – weekday names</i>	151
<i>Yearly calendar: one formula with Conditional Formatting</i>	152
<i>Conditional Formatting – each weekday is formatted differently</i>	153
Conclusion	154
Points to remember	154
6. Financial Operations with SEQUENCE	155
Introduction.....	155
Structure.....	155
Objectives.....	156
Examples of SEQUENCE with financial functions.....	156
<i>Loan return by payments per period</i>	156
<i>PMT - Periodic payment of a loan – traditional method</i>	157
<i>Periodic payment of a loan – a more flexible method</i>	160
<i>The Depreciation function in Excel – DB</i>	162
<i>Equally divide a sum of money over a period of time</i>	163
<i>One formula - How varying loan amounts impact the loan's installments</i> ..	163
<i>NPV – No need for a data table</i>	167
<i>PDURATION - Multiple results</i>	168
<i>The RATE function – multiple results</i>	170
<i>RRI - calculate the average annual interest rate of an investment</i>	171
<i>SEQUENCE and SUM</i>	173
Conclusion	174
Points to remember	175
7. SEQUENCE - The Ancilla of Math	177
Introduction.....	177
Structure.....	177
Objectives.....	178
Examples of SEQUENCE in math operations	178
<i>A number to the power of</i>	179

<i>Two methods to create a sequence of square roots</i>	180
<i>Two methods to generate a sequence of fractions</i>	181
<i>Creating a sequence of alternate 1's and 0's</i>	182
<i>Dynamic quadratic equation</i>	182
<i>SUM - a virtual Array</i>	183
<i>How many candles</i>	184
<i>Raising the number 2 to the power of 10 using bit operation</i>	184
<i>Simplest OR</i>	185
<i>Dynamic Sine with two Spin buttons</i>	185
<i>Exponential Growth example</i>	187
<i>Dynamic multiplication table</i>	188
<i>BIN2DEC – MMULT and SEQUENCE</i>	188
<i>BIN2DEC - SUM (or SUMPRODUCT) with SEQUENCE</i>	189
<i>Filling the missing values in a geometric series</i>	190
<i>Trigonometry with SEQUENCE</i>	190
<i>An array of duplicate numbers generated by bit operations</i>	191
<i>Using MMULT and SEQUENCE to track wins, losses, and ties in each quarter</i>	192
<i>Digital root</i>	192
<i>First n odd numbers squared (A simple solution)</i>	193
<i>First N odd numbers squared (A complex solution)</i>	194
<i>Find first divisor of a number (divisor found)</i>	194
<i>Find first divisor of a number (divisor not found)</i>	195
<i>Conclusion</i>	196
<i>Points to remember</i>	197
8. SEQUENCE and Other Animals	199
<i>Introduction</i>	199
<i>Structure</i>	199
<i>Objectives</i>	200
<i>Examples of SEQUENCE with other animals</i>	200
<i>Better than nested IF</i>	200

<i>Traditional solution – nested IF</i>	200
<i>XLOOKUP and SEQUENCE instead of nested IF (example 1)</i>	201
<i>XLOOKUP and SEQUENCE instead of nested IF (example 2)</i>	202
<i>XLOOKUP and SEQUENCE instead of nested IF (example 3)</i>	203
<i>Fetch the first and last digits from a string</i>	203
<i>Data validation – only Hebrew letters</i>	204
<i>Data validation – only uppercase English letters</i>	206
<i>Splitting cell by chunk size and separator (two examples)</i>	207
<i>Remove all digits from string</i>	208
<i>Remove names and split numbers to separate cells</i>	209
<i>Removing “A”, “B” and “C” from string – two methods</i>	209
<i>Method 1</i>	210
<i>Method 2</i>	210
<i>INDEX-SQRT instead of FILTER</i>	211
<i>Remove all uppercase or lowercase letters from string</i>	212
<i>Verifying the validity of a check digit with the Luhn algorithm</i>	213
Conclusion	215
Points to remember	215
Index	217-223

CHAPTER 1

A Short Introduction to Dynamic Array Functions in Excel 365

Introduction

The latest version of Excel (Excel 365) is nothing short of a revolution. The new formula engine of Excel allows you to reference an array in one formula and get multiple results. This was not the case until a few years ago. However, the most revolutionary aspect of this new version is that Microsoft has added some new functions, which operate on arrays and enhance their flexibility and ease of use.

This chapter will briefly introduce some of these powerful functions and demonstrate their utility and efficiency.

This introduction demonstrates only a small fraction of these functions' power. Also, only in few cases will we exemplify the combination of two or more of these functions. Such co-operation of two or more DAF can have endless applications for problems in Excel that could not have been solved earlier with formulae.

Structure

This chapter will discuss the functions in the bulleted list below. Each function will be briefly explained and elucidated in a picture/pictures which will show the function in action.

In this chapter, we will discuss the following topics:

- Introducing Dynamic Array Functions
- Examples of the new DAF
 - UNIQUE - returns a list of unique values in a range.
 - FILTER - selects data according to condition/s.
 - SORT - sorts a range, ascending/descending.
 - SORTBY - sorts a range by another range/array.
 - RANDARRAY - generates a bounded array of random numbers.
 - SEQUENCE - generates a sequence of numbers/characters.
 - TEXTSPLIT - splits text across columns/rows by a specified delimiter.
 - TOCOL - converts a horizontal/bi-dimensional array to a vertical one.
 - TOROW - converts a vertical/bi-dimensional array to a horizontal one.
 - VSTACK - stacks one or more arrays vertically, upon the other.
 - HSTACK - stacks one or more arrays horizontally, one after the other.
 - EXPAND - expands/pads an array by specified dimensions.
 - ARRAYTOTEXT - converts an array/range to a single-cell text string.
 - TEXTBEFORE - returns the text before a delimiter/substring.
 - TEXTAFTER - returns the text after a delimiter/substring.
 - CHOOSECOLS - returns data by specified column number/s.
 - CHOOSEROWS - returns data by specified row number/s.
 - WRAPROWS - wraps a row/column after a specified number.
 - WRAPCOLS - wraps a row/column after a specified number.
 - XLOOKUP - searches a range/array and returns first found item.
 - XMATCH - searches an item in a range/array and returns its position.
 - TAKE - returns part of an array according to rows/columns specified.
 - DROP - drops rows and/or columns from an array.
 - VALUETOTEXT - converts non-text to text, text left intact or wrapped in quotes.

Objectives

After reading this chapter, you will be able to understand the difference between single-cell and multiple-cell functions in Excel. You will also realize the quantum leap of the new **Dynamic Array Functions (DAF)**. This chapter will make it easier for you to comprehend the versatility and flexibility of the new generation functions, and implement these functions in your daily work.

Introducing Dynamic Array functions

This chapter exemplifies all the **Dynamic Array Functions (DAF)** introduced by Microsoft until the date of writing this chapter (December 2022). Some of these functions might be available only in the Beta version, which gives you the advantage of getting acquainted with them in advance.

These functions are very versatile and powerful. They can accomplish many new features that were not available in previous versions of Excel.

However, this chapter does not include the new **LET** and **LAMBDA** functions. Nor does it contain the **LAMBDA helper** functions that do not operate alone and serve as ancillary functions to **LAMBDA**. For example: **MAP**, **MAKEARRAY**, **DROP**, **SCAN**, **REDUCE**, **BYCOL**, and **BYROW**.

Although these non-DAF functions are not presented in the current chapter, some of them will be demonstrated in the next chapters, where we will expose the **SEQUENCE** Function in action.

The DAF list, however, is up to date and includes all the *independent* functions published till the date of writing this chapter.

Examples of the new DAF

We will now discuss the examples of the new DAF functions in detail:

UNIQUE function

The **UNIQUE** function extracts unique values from a list/array as illustrated in the following figure:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
2	aaa		aaa							Arguments to Function:								
3	bbb		bbb							1. Array - array to return unique values from								
4	aaa									2. By Col - unique rows (0) or unique columns (omitted)								
5	aaa									3. Exactly once - occur exactly once or distinct (omitted)								
6	bbb			=UNIQUE(A2:A4)														
7																		
8																		
9																		
10																		
11				The UNIQUE Function extracts unique values from a list/array														
12																		
13																		
14																		
15																		
16																		

Figure 1.1: The UNIQUE function: A simple Example

FILTER function

The **FILTER** function retrieves from an array only records that match a condition/conditions as illustrated in the following figure:

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Name	Registration Date			Month	1							
2	Yossi	01/01/2020								Arguments to Function:			
3	Eli	01/02/2020								1. Array - range to filter			
4	David	01/03/2020								2. Include - condition/s for filtering			
5	Anat	01/04/2020								3. If empty - if no items were selected (omitted)			
6	Yafa	01/05/2020											
7	Herzl	01/06/2020			Yossi	01/01/2020							
8	Avivit	01/07/2020											
9													
10													
11				=FILTER(\$A\$2:\$B\$8,MONTH(\$B\$2:\$B\$8)=\$F\$1)									
12													
13													
14													
15				Filter only customers whose registration date matches the parameter's month									
16													
17													
18													
19													
20													
21													

Figure 1.2: The FILTER function - filter only those who registered in January

SORT function

The **SORT** function sorts an array according to parameters. If the parameters are omitted (as in our example), the sort is being executed with the default parameters. In our example, the sort is in ascending order (the default):

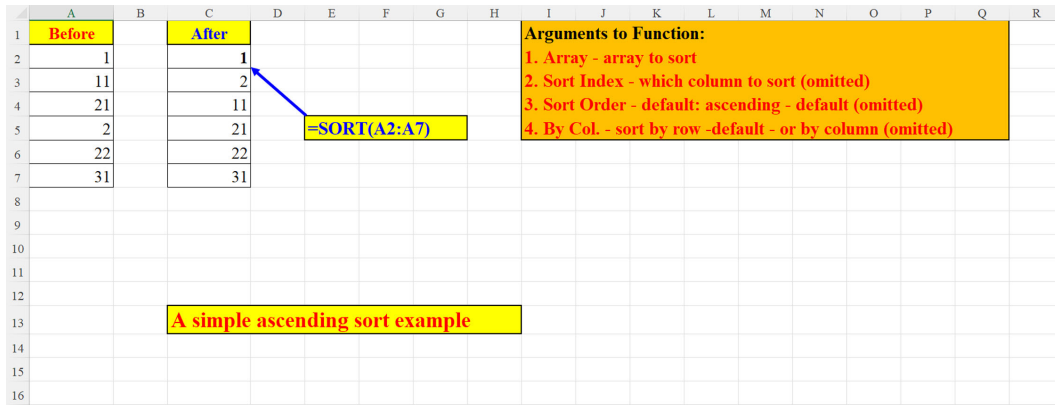


Figure 1.3: The SORT function (example 1): ascending SORT

In the following example, we are sorting in descending order (the third parameter is -1):

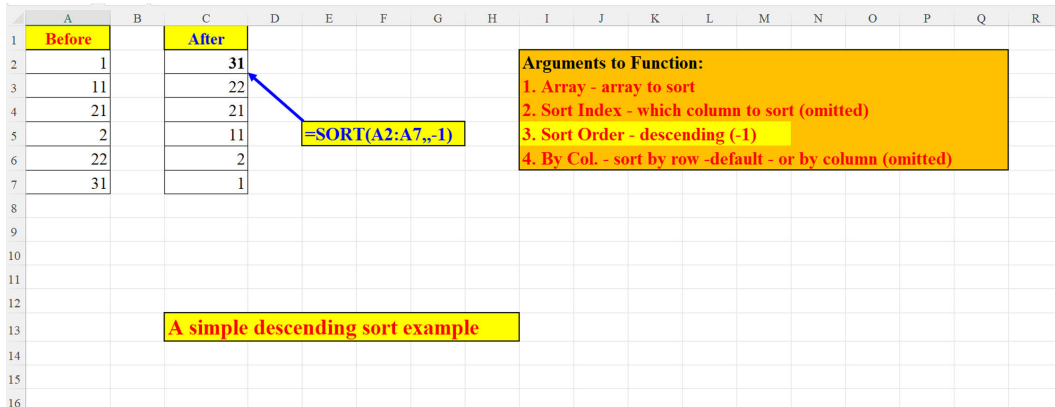


Figure 1.4: The SORT function (example 2): descending SORT

SORTBY function

The **SORTBY** function sorts by an array. In our example, the months of the dates array (B2:B8):

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1		Before			After					Arguments to Function:						
2		25/09/2022			18/01/2022					1. Array - array to sort						
3		16/08/2022			02/02/2022					2. By Array - which Array to sort by						
4		04/10/2022			13/05/2022											
5		18/01/2022			16/08/2022											
6		13/05/2022			25/09/2022											
7		12/12/2022			04/10/2022											
8		02/02/2022			12/12/2022											
9																
10										=SORTBY(B2:B8,MONTH(B2:B8))						
11																
12																
13		Sort an array of dates by month														
14																
15																
16																

Figure 1.5: The SORTBY function: SORT dates by month

RANDARRAY function

The **RANDARRAY** function generates a random array. In our example, it creates ten random integer numbers between 1 and 100:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	95									Arguments to Function:								
2	62									1. Rows - number of rows (default: 1)								
3	84									2. Columns - number of columns (default: 1)								
4	14									3. Min - smallest number in the resulting array (default: 0)								
5	75									4. Max - largest number in the resulting array (default: 1)								
6	40	=RANDARRAY(10,1,1,100,1)								5. Integer - whole/non-whole numbers (default: non-whole)								
7	50																	
8	82																	
9	34																	
10	60																	
11																		
12		This formula generates ten random whole numbers between 1 and 100																
13																		
14																		
15																		
16																		
17																		
18																		
19																		
20																		

Figure 1.6: The RANDARRAY function: generate random numbers

SEQUENCE function

We will learn the **SEQUENCE** function inside out, so here is a simple example of creating a sequence of 12 descending numbers:

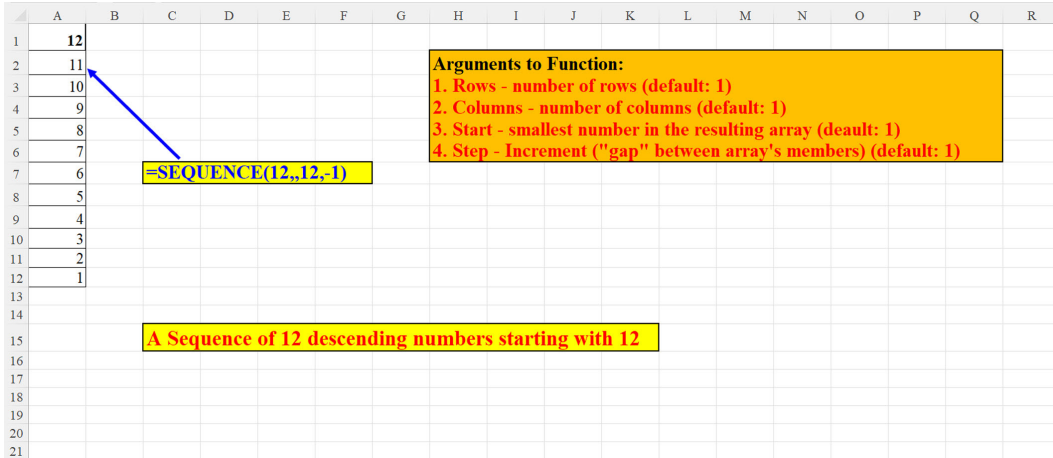


Figure 1.7: The SEQUENCE function: create a sequence of numbers

TEXTSPLIT function

The **TEXTSPLIT** function splits a cell's value by a delimiter.

In the following example, we use an array constant (in curly braces) to define more than one delimiter.

Please note, that although this function is considered a DAF it can operate properly only on single cells:

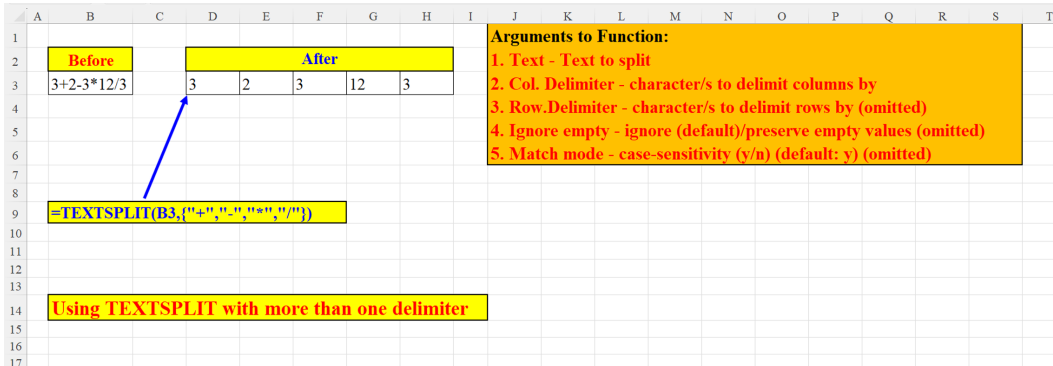


Figure 1.8: The TEXTSPLIT function: Split a cell's text into several cells

TOCOL function

Example 1:

A simple example of the **TOCOL** function in which we transpose the original array from horizontal to vertical:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1								Arguments to Function:												
2		H	e	l	l	o		1. Array - horizontal/bi-dimensional array to return as a vertical array												
3								2. Ignore - indicator whether to ignore certain cases (error, blanks). Default: keep all (omitted)												
4								3. Scan by column - indicates whether to scan by row or by column. Default: by column (omitted)												
5																				
6			H																	
7			e																	
8			l																	
9			l																	
10			o																	
11																				
12																				
13																				
14																				
15																				
16																				
17																				
18																				
19																				
20																				
21																				
22																				
23																				

Figure 1.9: TOCOL: A very simple Example

Example 2:

In this example, we use a “trick”: the inner IF statement returns an invalid value (“NA”), and since TOCOL’s second argument = 2 tells Excel to ignore errors, the resulting array (Col.E) displays matching rows only:

	A	B	C	D	E	F	G	H	I	J	K	L
1	List 1	List 2			Matching Rows only							
2	689	689			689							
3	14776	333			4448							
4	1678901	9999			57779							
5	4448	4448			399999							
6	57779	57779										
7	123454	11111										
8	399999	399999										
9	24598	13										
10	135782	14										
11	134											
12												
13												
14												
15												
16												
17												
18												
19												
20												

Figure 1.10: TOCOL: A more “sophisticated” Example

TOROW function

A simple example of the **TOROW** function in which we transpose the original array from vertical to horizontal is illustrated in the following figure:

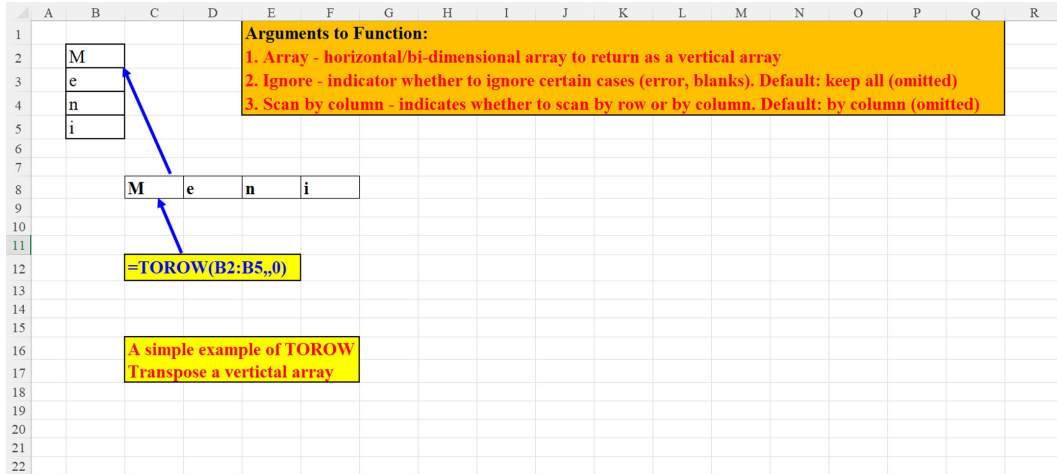


Figure 1.11: TOROW: A very simple example

VSTACK function

In the following example, we *stack* two arrays one on top of the other to create one vertical array:

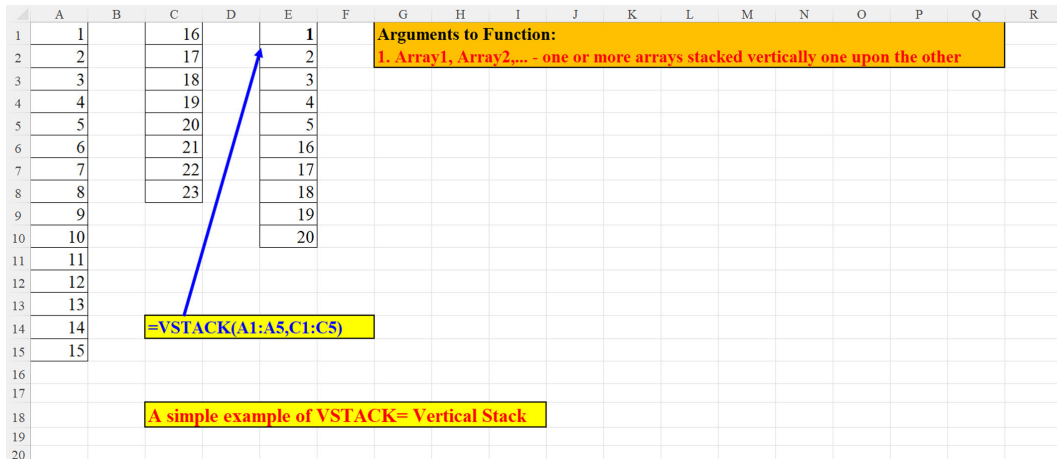


Figure 1.12: VSTACK: A very simple example

VSTACK versus TOCOL

The following image demonstrates the differences between **TOCOL** (on the left-hand side) and **VSTACK** (on the right-hand side).

TOCOL creates the stack horizontally, from left to right. However, **VSTACK** stacks the first column on top of the second one:

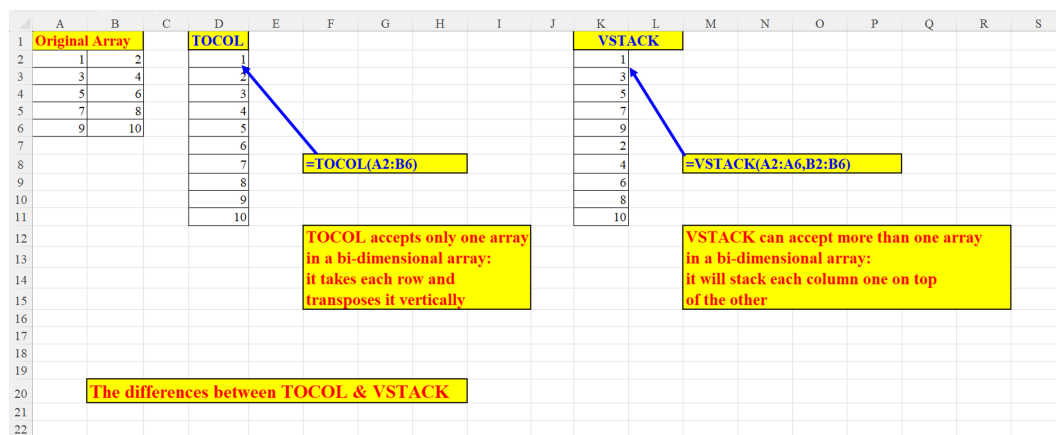


Figure 1.13: The differences between TOCOL and VSTACK

HSTACK function

The **HSTACK** function stacks horizontally, as can be seen in the following figure. We slice two rows and combine them to one long array, arranged horizontally:

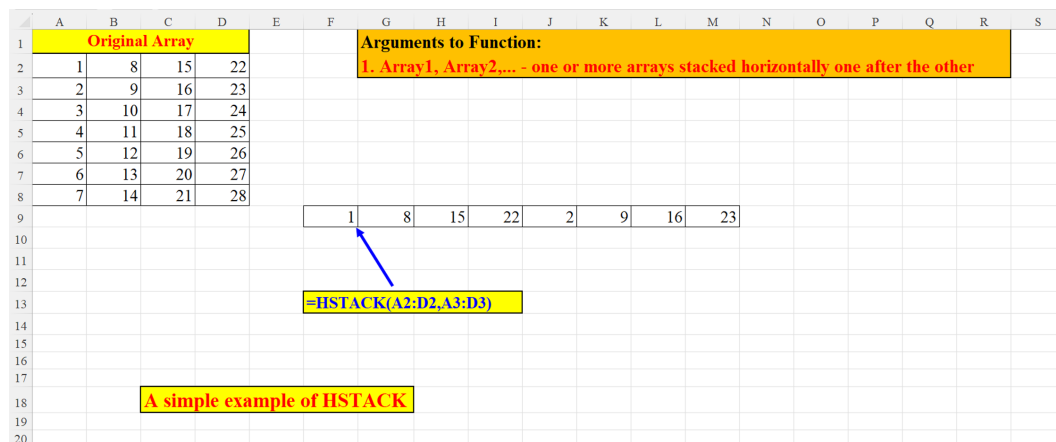


Figure 1.14: HSTACK: a very simple example

EXPAND function

This example utilizes the new Data Types feature of Excel 365 (Data*Data Types*Geography):

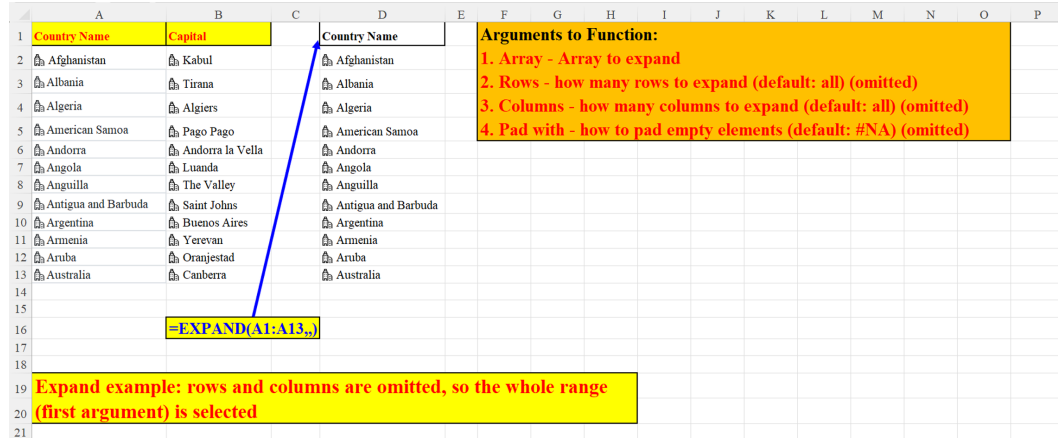


Figure 1.15: EXPAND example

ARRAYTOTEXT function

Some examples of the **ARRAYTOTEXT** function, from top to bottom: converting an array of numbers to text in a single cell (two examples), converting an array of characters to a single cell (two examples) and showing the similarities between this function and the **TEXTJOIN** Function.

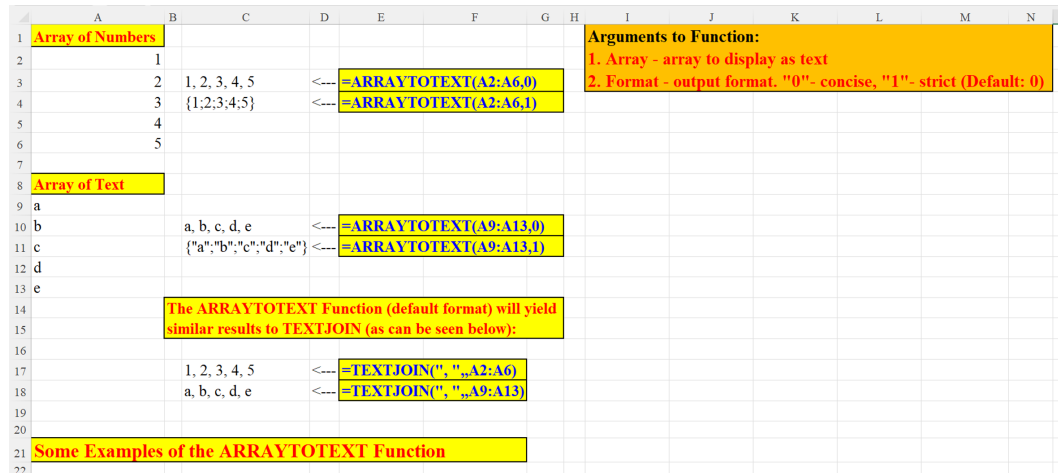


Figure 1.16: ARRAYTOTEXT example

TEXTBEFORE function

The **TEXTBEFORE** function can extract text from an array / range preceding a delimiter:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	City & State	City only			Arguments to Function:											
2	New York, NY	New York			1. Text - text to search for the delimiter											
3	Los Angeles, CA	Los Angeles			2. Delimiter - the character/s used as the delimiter											
4	Chicago, IL	Chicago			3. Instance num - instance of delimiter in text (Default: 1)											
5	Houston, TX	Houston			4. Match mode - is delimiter case-sensitive (0-No,1-yes) (Default: 1)											
6	Phoenix, AZ	Phoenix			5. Match end - match delimiter against end-of-text (0-No,1-yes) (Default: 0)											
7	Philadelphia, PA	Philadelphia														
8	San Antonio, TX	San Antonio														
9	San Diego, CA	San Diego			=TEXTBEFORE(A2:A17, ",")											
10	Dallas, TX	Dallas														
11	San Jose, CA	San Jose														
12	Austin, TX	Austin														
13	Jacksonville, FL	Jacksonville														
14	Fort Worth, TX	Fort Worth														
15	Columbus, OH	Columbus														
16	Indianapolis, IN	Indianapolis														
17	Charlotte, NC	Charlotte														
18																
19																
20																
21																
22																

An example of **TEXTBEFORE**
Extract text before the delimiter (comma)

Figure 1.17: **TEXTBEFORE** example

TEXTAFTER function

Here, two examples are being demonstrated.

Example 1:

The first is a simple one, where the instance number = -1, which means: finding the last occurrence of the delimiter and fetching the text immediately following it. This is illustrated in the following figure:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Before	After		Arguments to Function:											
2	J.S. Bach Leipzig Germany	Germany		1. Text - text to search for the delimiter											
3	W.A. Mozart Vienna Austria	Austria		2. Delimiter - the character/s used as the delimiter											
4				3. Instance num - instance of delimiter in text (Default: 1) (-1=last)											
5				4. Match mode - is delimiter case-sensitive (0-No,1-yes) (Default: 1)											
6				5. Match end - match delimiter against end-of-text (0-No,1-yes) (Default: 0)											
7															
8				=TEXTAFTER(A2:A3, " ", -1)											
9															
10															
11															
12															
13															
14															
15															
16															
17															
18															
19															

TEXTAFTER example:
The formula fetches the substring after the last delimiter (blank)

Figure 1.18: **TEXTAFTER** – a simple example

Example 2:

The second example illustrates a very interesting feature of this function:

Even though the delimiter is defined as a single blank character (" "), the function returns the text after more than one blank.

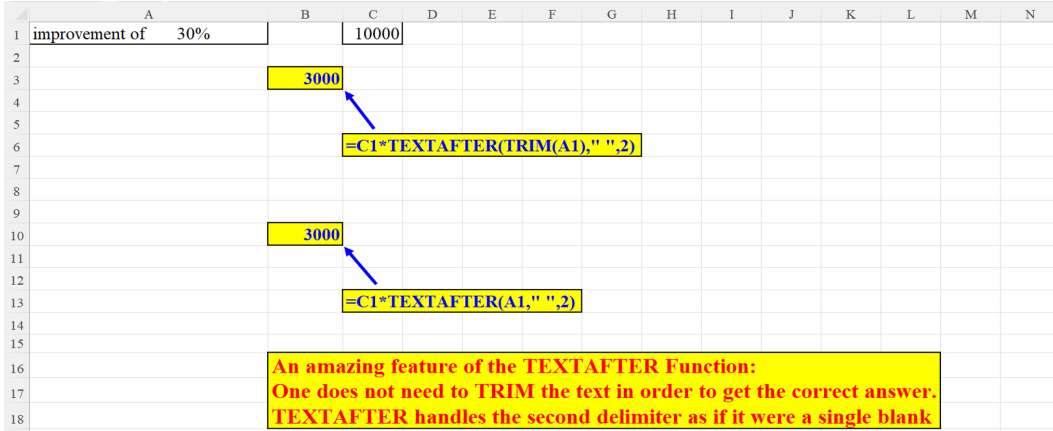


Figure 1.19: TEXTAFTER – An amazing feature

CHOOSECOLS function

The **CHOOSECOLS** function allows us to select only certain columns of a multi-column array.

Please note that:

- The second argument can be written either as a list of numbers, that is, 1,2 or as an array constant: {1,2}
- Specifying -1 as the second argument will fetch the last column only:

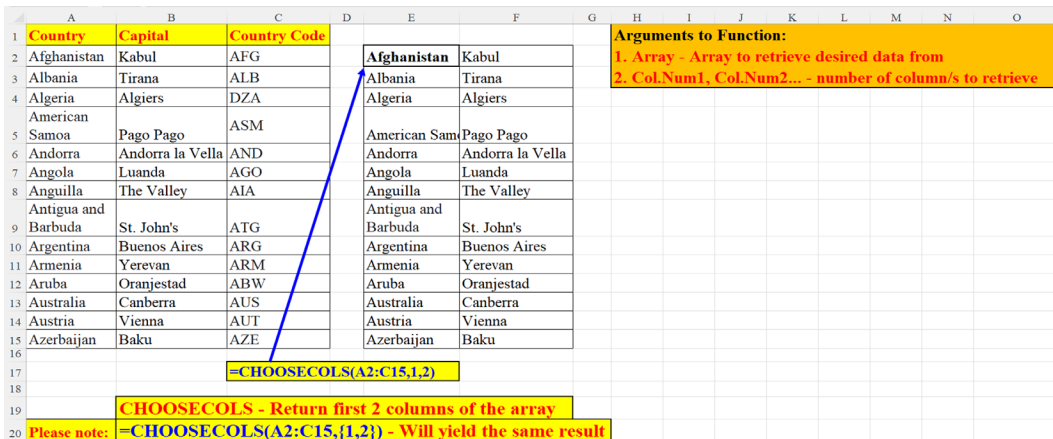


Figure 1.20: CHOOSECOLS example

CHOOSEROWS function

The **CHOOSECOLS** function allows us to select only certain rows of a multi-row array.

Please note that:

- The second argument can be written either as a list of numbers, that is, 1,6 or as an array constant: {1,6}
- Specifying -1 as the second argument will fetch the last row only:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Country	Capital	Country Code						Arguments to Function: 1. Array - Array to retrieve desired data from 2. Row.Num1, Row.Num2... - number of row/s to retrieve								
2	Afghanistan	Kabul	AFG	Afghanistan	Kabul	AFG											
3	Albania	Tirana	ALB	Angola	Luanda	AGO											
4	Algeria	Algiers	DZA														
5	American Samoa	Pago Pago	ASM														
6	Andorra	Andorra la Vella	AND														
7	Angola	Luanda	AGO														
8	Anguilla	The Valley	AIA														
9	Antigua and Barbuda	St. John's	ATG														
10	Argentina	Buenos Aires	ARG														
11	Armenia	Yerevan	ARM														
12	Aruba	Oranjestad	ABW														
13	Australia	Canberra	AUS														
14	Austria	Vienna	AUT														
15	Azerbaijan	Baku	AZE														
16																	
17				=CHOOSEROWS(A2:C15,1,6)													
18																	
19																	
20				CHOOSEROWS - Return 1st and 6th rows of the array													
21																	
22																	
23																	

Figure 1.21: CHOOSEROWS example

WRAPROWS function

The **WRAPROWS** function *wraps* a vertical array by rows. In the following example, we *wrap* two consecutive rows of the original (one-dimensional) array into a two-column array:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Premiers			Premiers				Arguments to Function: 1. Vector - vector to be wrapped 2. Wrap count - number of cells to wrap 3. Pad with - character/s to pad with (default: none)								
2	USA			USA	Joe Biden											
3	Joe Biden			Germany	Olaf Scholz											
4	Germany			UK	Rishi Sunak											
5	Olaf Scholz			Russia	Vladimir Putin											
6	UK			China	Xi Jinping											
7	Rishi Sunak															
8	Russia															
9	Vladimir Putin															
10	China			=WRAPROWS(A2:A11,2)												
11	Xi Jinping															
12																
13																
14																
15																
16				WRAPROWS Example												
17				Each two consecutive rows will be "wrapped"												
18				into a single row in 2 consecutive columns												
19																

Figure 1.22: WRAPROWS example

WRAPCOLS function

In the **WRAPCOLS** function example, each two consecutive rows of the original (one-dimensional) array are being *wrapped* into a two-row array as illustrated in the following figure:

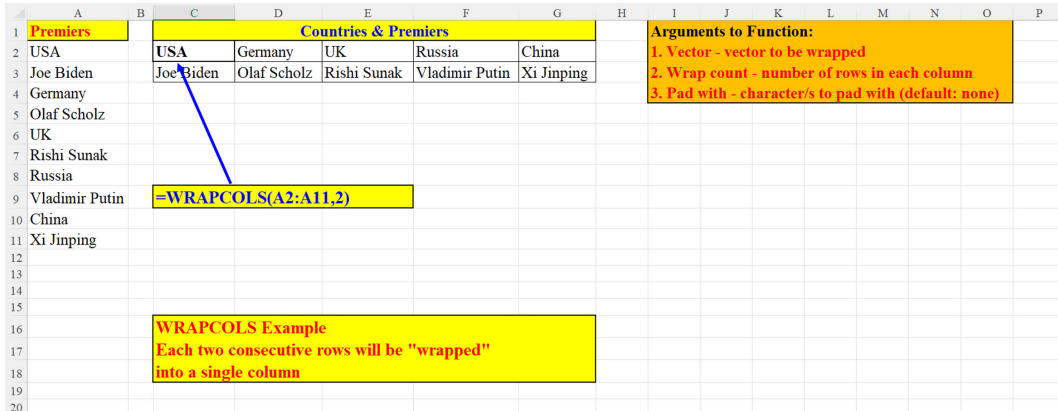


Figure 1.23: WRAPCOLS example

XLOOKUP function

The **XLOOKUP** function is the more advanced and flexible version of the good old **VLOOKUP** function.

Example 1:

In our example, Argument no. 5 is -1 means: Exact match. If the Lookup Value is not found, the next smaller item is returned.

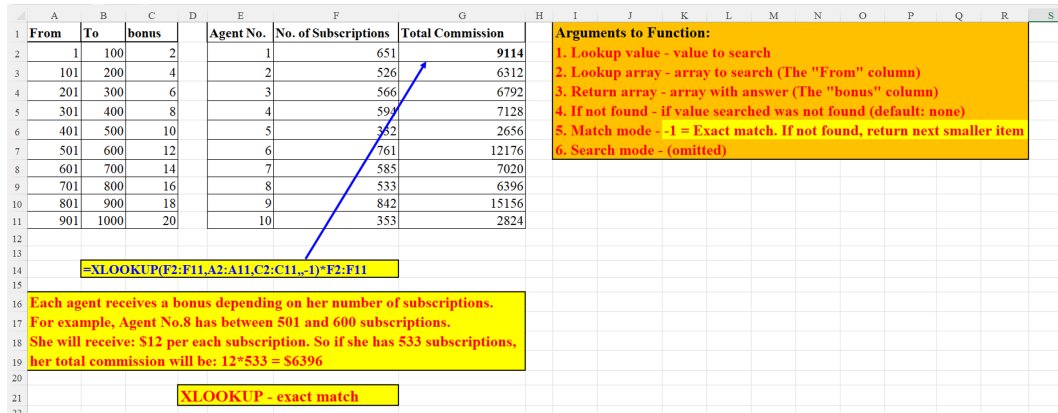


Figure 1.24: XLOOKUP example 1 – exact match returns the next smaller value

Example 2:

Another example with **XLOOKUP**. Here, Argument no.5 is 2 which means: wildcard match as shown in the following figure:

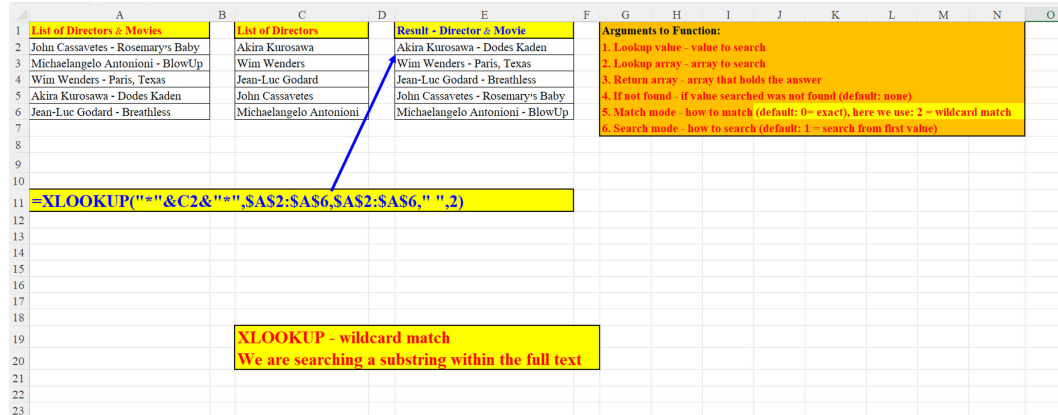


Figure 1.25: XLOOKUP example 2 – wildcard match

Example 3:

Third example of XLOOKUP. Here Argument no.5 is 1 which means: exact match. If not found, as in our case, the next larger item is returned as shown in the following figure:

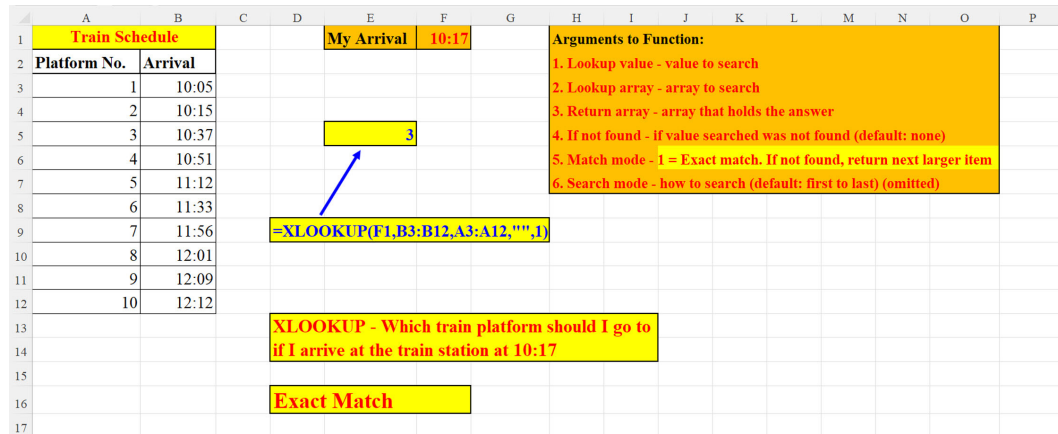


Figure 1.26: XLOOKUP example 3 – exact match returns the next larger item

XMATCH function

XMATCH is a more advanced version of the veteran **MATCH** function. When searching for an exact match, (as in our example) the looked-up column does not need to be sorted (as in **MATCH**). See the following screenshot:

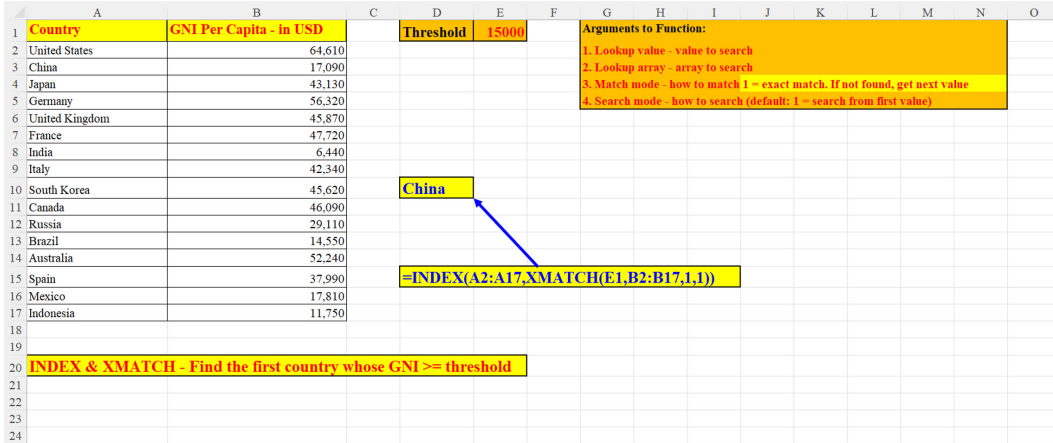


Figure 1.27: XMATCH combined with INDEX: find 1st country whose GNI >= 15,000

TAKE function

The **TAKE** function *carves* a sub-array from the original array. Here, the result is a 3*3 array:

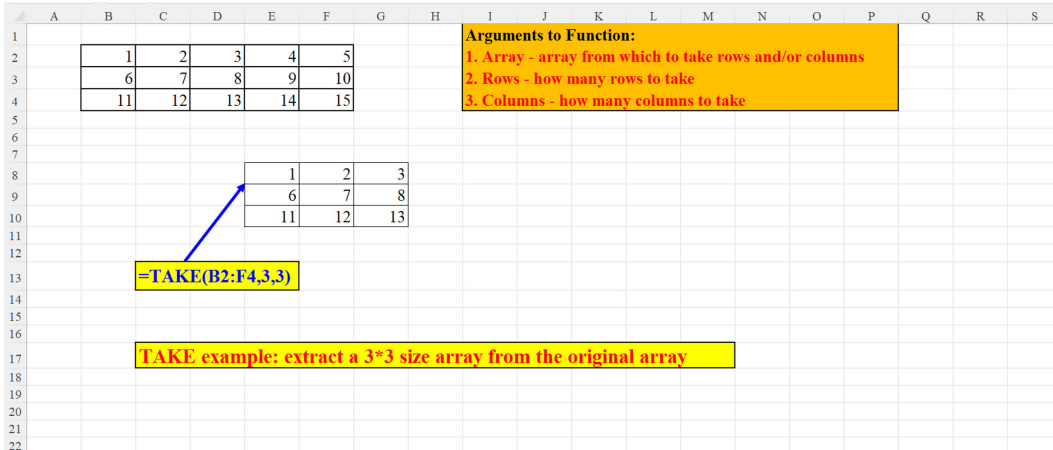


Figure 1.28: TAKE example: create a two-dimensional “sub-array”

DROP function

The **DROP** function omits rows and/or columns from an array to create a smaller array.

The second argument is negative, which tells Excel to drop rows from the array's end:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Country	Capital	Country Code		Country	Capital	Country Code		Arguments to Function:						
2	Afghanistan	Kabul	AFG		Afghanistan	Kabul	AFG		1. Array - array to retrieve desired data from						
3	Albania	Tirana	ALB		Albania	Tirana	ALB		2. Rows - no. of rows to omit (negative=from bottom)						
4	Algeria	Algiers	DZA		Algeria	Algiers	DZA		3. Columns - no. of columns to drop (default: none)						
5	American Samoa	Pago Pago	ASM		American Samoa	Pago Pago	ASM								
6	Andorra	Andorra la Vella	AND		Andorra	Andorra la Vella	AND								
7	Angola	Luanda	AGO		Angola	Luanda	AGO								
8	Anguilla	The Valley	AIA		Anguilla	The Valley	AIA								
9	Antigua and Barbuda	St. John's	ATG		Antigua and Barbuda	St. John's	ATG								
10	Argentina	Buenos Aires	ARG		Argentina	Buenos Aires	ARG								
11	Armenia	Yerevan	ARM												
12	Aruba	Oranjestad	ABW												
13	Australia	Canberra	AUS												
14	Austria	Vienna	AUT												
15	Azerbaijan	Baku	AZE												
16		=DROP(A1:C15,-5,)													
17		DROP - Remove rows and/or columns from array													
18		In this example we "drop" the last 5 rows of the original array													
19															
20															
21															
22															
23															

Figure 1.29: DROP example

VALUETOTEXT function

The **VALUETOTEXT** function converts non-text to text. Since the second argument is omitted, the text is unaltered (that is, not put in quotes):

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Name	Registration Date						Function's Arguments:							
2	Yossi	01/01/2020			Yossi	43831		1. Value - range to be converted to text							
3	Eli	01/02/2020			Eli	43862		2. Format - "0" - text unaltered, "1" - text put in quotes							
4	David	01/03/2020			David	43891									
5	Anat	01/04/2020			Anat	43922									
6	Yafa	01/05/2020			Yafa	43952									
7	Herzl	01/06/2020			Herzl	43983									
8	Avivit	01/07/2020			Avivit	44013									
9															
10															
11		=VALUETOTEXT(A2:B8)													
12															
13															
14															
15															
16															
17		VALUETOTEXT - converts non-text to text, text is either unaltered or wrapped in quotes													
18															
19															
20															

Figure 1.30: VALUETOTEXT example

Conclusion

In this chapter, we became acquainted with a new type of functions in Excel. No more functions that operate on a single cell or yield a result in a single cell. These functions are the Dynamic Array Functions which operate on an array or produce an array, either from a single cell or from an array. Some of these new functions can operate simultaneously on rows as well as on columns.

The next chapter, *SEQUENCE* and *TEXT* operations is going to delve into the intricate features of *SEQUENCE* in abundant cases and examples, which show its amazing potency and suppleness when slicing and dicing text.

Points to Remember

- The above DAF can be organized into the following categories: Transpose: **TOROW, TOCOL**
- Compact and rearrange: **HSTACK, VSTACK, CHOOSEROWS, CHOOSECOLS, WRAPROWS, WRAPCOLS, TAKE, EXPAND, DROP, UNIQUE, SORT, SORTBY**
- Search and Select: **XLOOKUP, XMATCH, FILTER**
- Slice and cleave: **TEXTSPLIT, TEXTBEFORE, TEXTAFTER**
- Manipulate data: **VALUETOTEXT, ARRAYTOTEXT**
- Randomize data: **RANDARRAY**
- Create series: **SEQUENCE**
- Please be advised that these categories are given only for the sake of simplicity. As you practice them, you will surely find new ways to implement and apply them, often in concert with other DAF to enhance the power and robustness of your formulae and achieve neat results which were not possible in the past.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



CHAPTER 2

SEQUENCE in Text Operations

Introduction

This chapter will thoroughly discuss how useful the **SEQUENCE** function is in text operations. After *Chapter 1, Introduction to Dynamic Array Functions in Excel 365*, the introductory chapter, we are going to dive in this chapter and the next ones to understand the various areas of Excel in which **SEQUENCE** can be helpful, useful, and efficient.

This chapter, as the title hints, will exemplify the use of **SEQUENCE** with other Excel functions to execute solutions to variegated tasks. Some examples are going to use the new functions discussed in the previous chapter (and also the **LET** function, which was only mentioned).

Structure

The chapter covers the following topics:

- Examples of **SEQUENCE** with text
 - Finding the names of the 10 highest-paid employees
 - How many words are there in the cell (version 1)
 - How many words are there in the cell (version 2)

- How many times does a string appear in the cell (method 1 of 5)
- How many times does a string appear in the cell (method 2 of 5)
- How many times does a string appear in the cell (method 3 of 5)
- How many times does a string appear in the cell (method 4 of 5)
- How many times does a string appear in the cell (method 5 of 5)
- Extract all characters - Horizontally
- Extract all characters – Vertically
- All uppercase English in one column
- All uppercase English in one cell
- Duplicate a sequence of characters
- Duplicate a cell by a duplication factor (method1)
- Duplicate a cell by a duplication factor (method2)
- Creating English uppercase letters without knowing how many letters there are
- Transpose without TRANSPOSE
- Extract only first three letter of weekday names
- Extract only digits from a string (method 1 of 3)
- Extract only digits from a string (method 2 of 3)
- Extract only digits from a string (method 3 of 3)
- Extract only unique Alphabetic characters from a string
- Split numbers and Text
- Remove unwanted Characters from string (2 named ranges as parameters)
- Remove unwanted characters from string (Formula)
- How many times does a string appear in a range
- Is it a Palindrome?
- Add vendor to list (the table)
- Add vendor to list (the formula)
- Remove all digits from the string
- Move first name from end of cell to the beginning
- Reverse String (Method 1)

- Reverse String (Method 2)
- From Vertical to Horizontal
- How many words are there in the cell without the separator (SEQUENCE)
- How many words are there in the cell without the separator (TEXTSPLIT)
- “Off with their heads”
- Extract only digits and add a separator
- How many lower-case letters are there in the cell (Method 1)
- How many lower-case letters are there in the cell (Method 2)
- All Greek letters in one formula
- Find last word in cell (Method 1 of 3)
- Find last word in cell (Method 2 of 3)
- Find last word in cell (Method 3 of 3)
- Number of characters in cell (without the separator)
- Number of non-empty cells in a column
- Strip leading and trailing digits (Method 1 of 2)
- Strip leading and trailing digits (Method 2 of 2)
- Increasing Text from End to Start
- Increasing Text from Start to End
- Hebrew Gematria (Formula)
- Hebrew Gematria (Gtable – Translation table)
- Extract only Country Names
- How many occurrences of a String starting from a certain Position
- Remove Diacritics from Hebrew words
- Is it a Palindrome (Arabic)
- Convert Hebrew Letters into English Letters
- Fetch description of Nth item of a non-sorted Key
- Extract Letters only from a chosen Language (Formula)
- Extract Letters only from a chosen Language (Validation List)
- Gematria in English (Method 1)

- Gematria in English (Method 2)
- How many active months?
- VLOOKUP&SEQUENCE – fetch more than one column
- Find Unicode value for any character in the string, no matter which language

Objectives

The objective of this chapter is to familiarize the reader with the variety of features, challenges, and flexibility by which the **SEQUENCE** function can be implemented when dealing with textual challenges in Excel.

Examples of SEQUENCE with text

In this section, we will look at various examples of SEQUENCE with text:

Finding the names of the 10 highest-paid employees

This formula displays the 10 highest-paid employees in the list. The parameter in cell H1 enables us to choose any number we desire, be it smaller or larger than 10 as shown in the following figure:

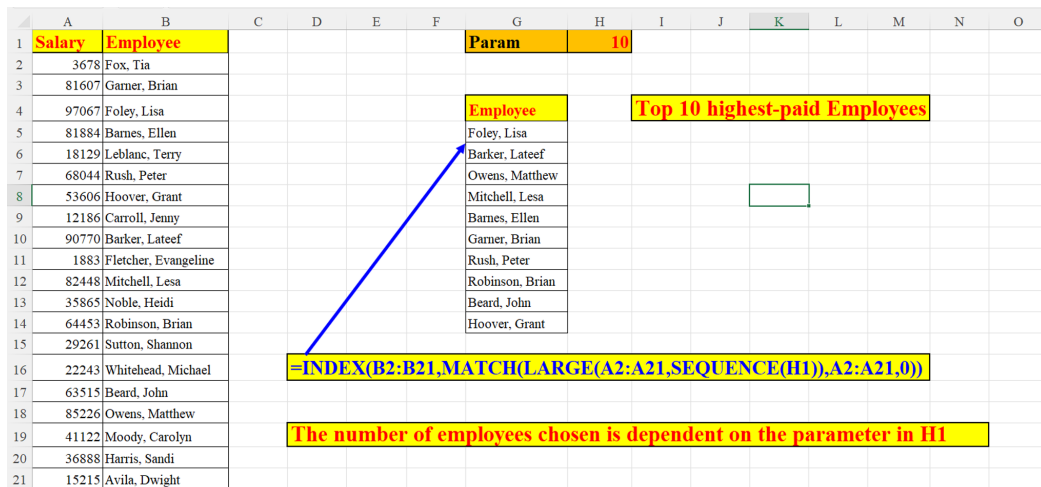


Figure 2.1: Names of the top 10 highest-paid employees

How many words are there in the cell (version 1)

This is the first solution version for this challenge:

The words are separated by a delimiter (in our example, a space (" ")). In such a case, the formula trims excessive spaces between words, giving us the correct result as shown in the following screenshot:

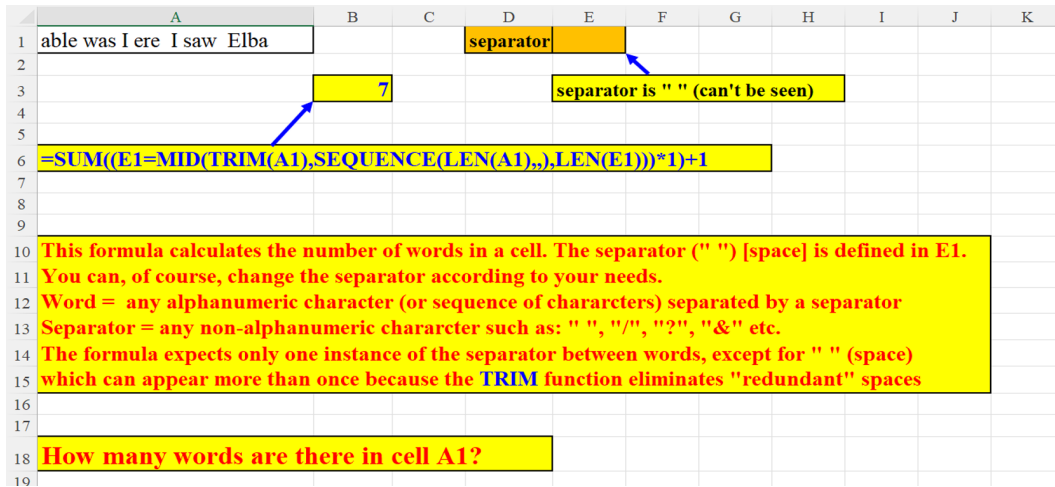


Figure 2.2: How many words are there in the cell (version 1)

How many words are there in the cell (version 2)

A shorter alternative to the previous example, using the more advanced function: **TEXTSPLIT** as shown in the following screenshot (released lately and is explained in *Chapter 1, Introduction to Dynamic Array Functions in Excel 365*):

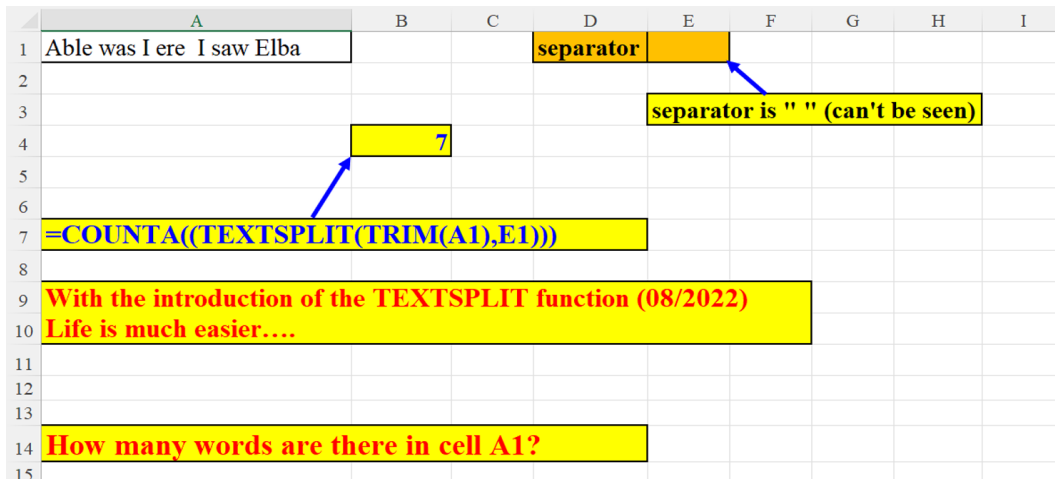


Figure 2.3: How many words are there in the cell (version 2)

How many times does a string appear in the cell Method 1 of 5

Method 1 (of 5): We want to know how many times the string defined in D2 (parameter) appears in cell A2. If we want a case-sensitive search, we should change **SEARCH** to **FIND** as shown in the following screenshot:

	A	B	C	D	E	F	G	H	I	J	K	L
1	Fruits			String to search								
2	oranges, apples, oranges, oranges, apples, oranges			orange								
3												
4												
5			4		method 1	for case-sensitive use FIND instead of SEARCH						
6												
7												
8	=COUNT(UNIQUE(SEARCH(D2,A2,SEQUENCE(LEN(A2))))))											
9												
10												
11												
12												
13												
14	How many times does the string in D2 appear in A2											
15												
16												
17												

Figure 2.4: How many times does a string appear in the cell (method 1)

Method 2 of 5

Method 2 (of 5): using **SUM**:

	A	B	C	D	E	F	G
1	Fruits			String to search			
2	oranges, apples, oranges, oranges, apples, oranges			orange			
3							
4							
5							
6			4		method 2		
7							
8							
9	=SUM(IF(ISERROR(FIND(D2,MID(A2,SEQUENCE(LEN(A2)),LEN(D2))))),'',1))						
10							
11							
12							
13							
14							
15	How many times does the string in D2 appear in A2						
16							
17							

Figure 2.5: How many times does a string appear in the cell (method 2)

Method 3 of 5

Method 3 (of 5): using **EXACT**:

	A	B	C	D	E	F
1	Fruits			String to search		
2	oranges, apples, oranges, oranges, apples, oranges			orange		
3						
4						
5						
6			4		method 3	
7						
8						
9	=SUM(--EXACT(D2,MID(A2,SEQUENCE(LEN(A2)),LEN(D2))))					
10						
11						
12	How many times does the string in D2 appear in A2					
13						
14						
15						
16						

Figure 2.6: How many times does a string appear in the cell (method 3)

Method 4 of 5

Method 4 (of 5): The N function converts **FALSE** / **TRUE** to 0/1:

	A	B	C	D	E	F
1	Fruits			String to search		
2	oranges, apples, oranges, oranges, apples, oranges			orange		
3						
4						
5			4		method 4	
6						
7						
8	=SUM(N(D2=MID(A2,SEQUENCE(LEN(A2)),LEN(D2))))					
9						
10						
11						
12						
13						
14	How many times does the string in D2 appear in A2					
15						
16						
17						
18						

Figure 2.7: How many times does a string appear in the cell (method 4)

Method 5 of 5

Method 5 (of 5): Using **UNIQUE**:

	A	B	C	D	E	F
1	Fruits			String to search		
2	oranges, apples, oranges, oranges, apples, oranges			orange		
3						
4						
5						
6						
7			4		method 5	
8						
9						
10	=COUNT(UNIQUE(FIND(D2,CONCAT(A2),SEQUENCE(LEN(A2))))))					
11						
12						
13						
14	How many times does the string in D2 appear in A2					
15						
16						
17						

Figure 2.8: How many times does a string appear in the cell (method 5)

Extract all characters - Horizontally

Split each character of the original string into a separate cell (horizontally):

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1														
2	Meni Porat		M	e	n	i		P	o	r	a	t		
3														
4														
5														
6														
7														
8														
9														
10		=MID(A2, SEQUENCE(LEN(A2)),1)												
11														
12														
13														
14														
15														
16		Split each character horizontally to a unique cell												
17														
18														
19														
20														

Figure 2.9: Extract all characters – Horizontally

Extract all characters – Vertically

Split each character of the original string into a separate cell (vertically) as shown in the following screenshot:

	A	B	C	D	E	F	G	H	I	J
1	Punctuation									
2										
3		P								
4		u								
5		n								
6		c								
7		t								
8		u								
9		a		=MID(\$A\$1,SEQUENCE(LEN(\$A\$1)),1)						
10		t								
11		i								
12		o								
13		n								
14										
15										
16										
17										
18				Extract all characters in a text string (Vertically)						
19										
20										

Figure 2.10: Extract all characters – Vertically

All uppercase English in one column

Generating the whole English alphabet (Uppercase Letters) with one simple formula in a range is shown in the following figure:

	A	B	C	D
1	A			
2	B			
3	C			
4	D			
5	E			
6	F		=CHAR(SEQUENCE(26,,65))	
7	G			
8	H		26 Uppercase English Letters Starting from CHAR(65)="A"	
9	I			
10	J			
11	K			
12	L			
13	M			
14	N			
15	O			
16	P			
17	Q			
18	R			
19	S			
20	T			
21	U			
22	V			
23	W			
24	X			
25	Y			
26	Z			
27				

Figure 2.11: All uppercase English Letters in one Column

All uppercase English in one cell

Generating the whole English alphabet (Uppercase letters) with one simple formula in one cell can be seen in the following screenshot:

	A	B	C	D
1				
2		ABCDEFGHIJKLMNOPQRSTUVWXYZ		
3				
4				
5				
6				
7		=CONCAT(CHAR(SEQUENCE(26,,65)))		
8				
9				
10				
11				
12				
13		All 26 Uppercase Letters in One Cell		
14				
15				
16				
17				
18				
19				
20				
21				

Figure 2.12: All uppercase English letters in one cell

Duplicate a sequence of characters

The following technique is very useful when you want to duplicate a sequence of characters. You need to supply three parameters:

- which characters to start with (cell: G2)
- the total number of characters in the array (in cell: I2)
- by what factor to duplicate each letter (how many times should each character appear) (in cell: M2):

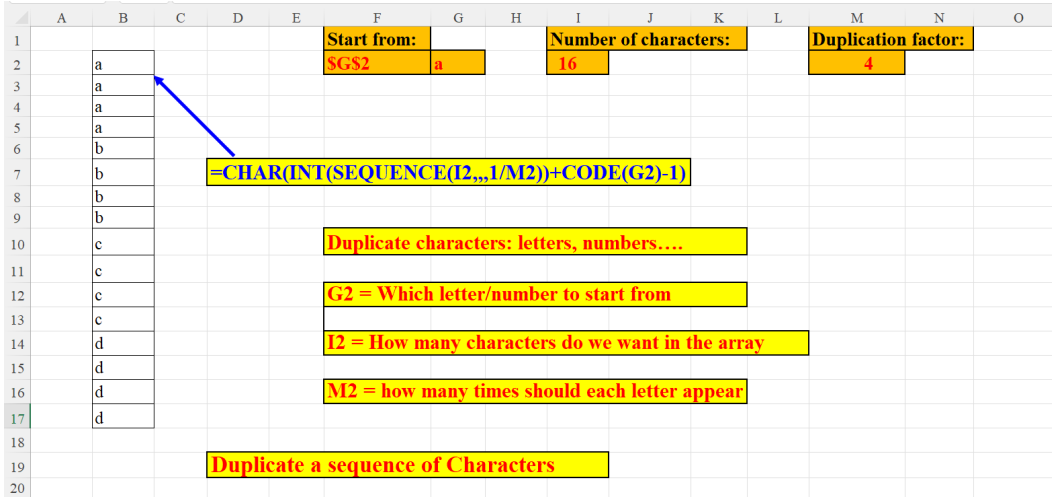


Figure 2.13: Duplicate a sequence of characters

Duplicate a cell by a duplication factor

Method 1

The duplication factor (in cell: I2) specifies how many times the string in cell A2 should be duplicated.

Method 1 of 2 is illustrated in the following figure:

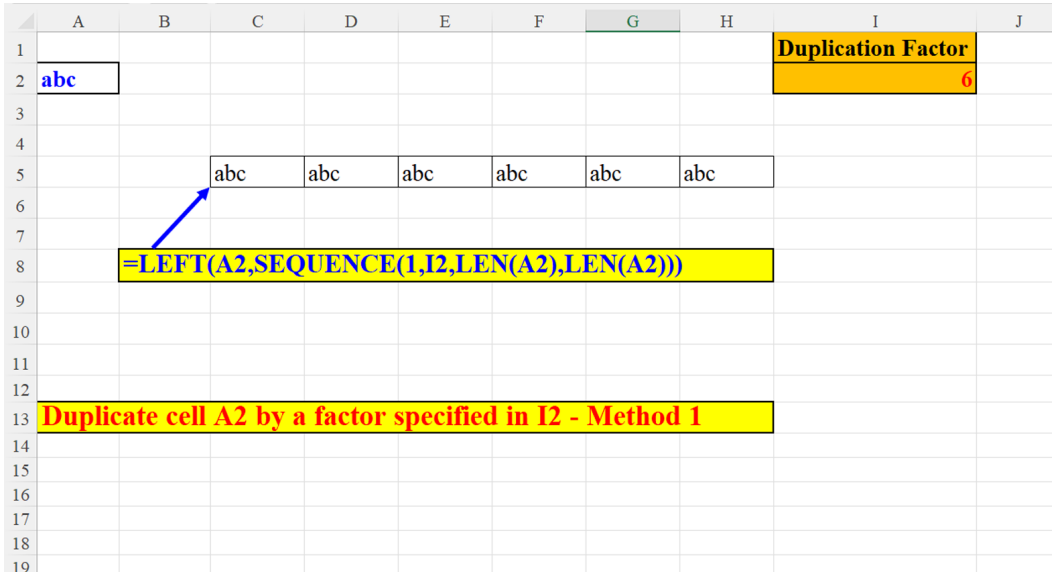


Figure 2.14: Duplicate a cell by a duplication factor (method1)

Method 2

The duplication factor (in cell: I2) specifies how many times the string in cell A2 should be duplicated.

Method 2 of 2 is illustrated in the following figure:

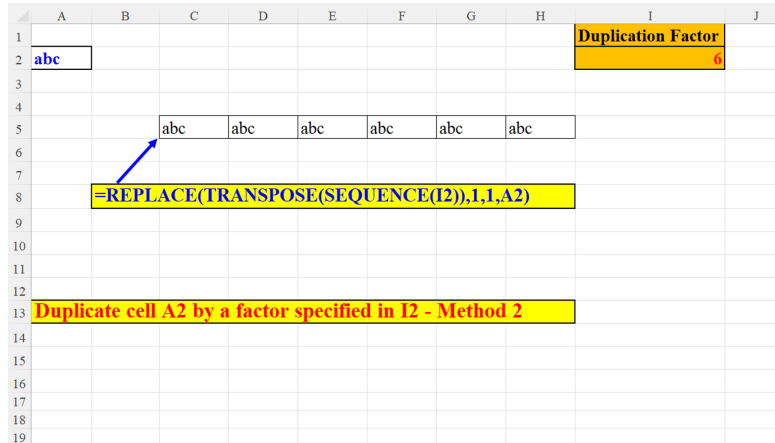


Figure 2.15: Duplicate a cell by a duplication factor (method2)

Creating English uppercase letters without knowing how many letters there are

The trick here is to find the number of letters by subtracting the ASCII value of the first letter ("A") from that of the last letter ("Z") as shown in the following figure:

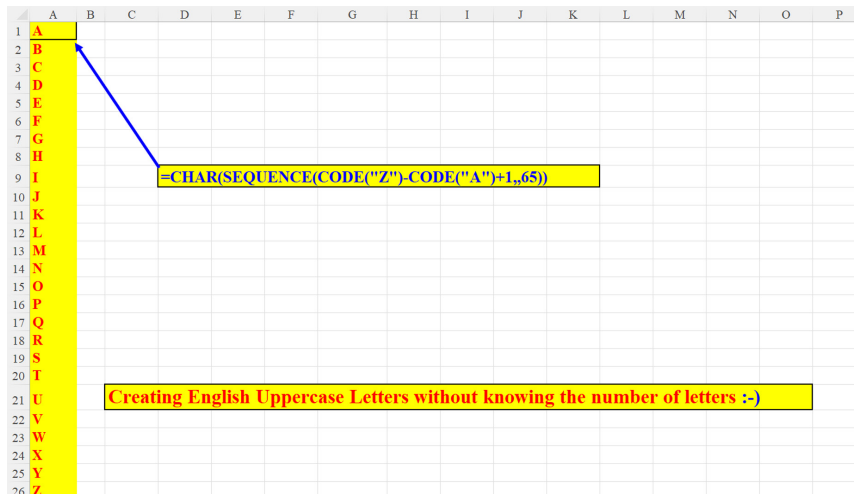


Figure 2.16: Creating the English uppercase letters without knowing the number of the Alphabet letters

Transpose without TRANSPOSE

Converting a vertical list into a horizontal list without using the **TRANSPOSE** function can be done as seen in the following screenshot:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1			January	February	March	April	May	June	July	August	September	October	November	December	
2	January														
3	February														
4	March														
5	April														
6	May														
7	June														
8	July														
9	August			=INDEX(A2:A13,SEQUENCE(12),1)											
10	September														
11	October														
12	November														
13	December														
14															
15															
16															
17															
18															
19															

Figure 2.17: Transpose without TRANSPOSE

Extract only first three letter of weekday names

Two methods to extract only the first three letters of the weekday names – with and without **SEQUENCE**:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1																		
2																		
3	Sunday			Sun									Sun					
4	Monday			Mon									Mon					
5	Tuesday			Tue									Tue					
6	Wednesday			Wed									Wed					
7	Thursday			Thu									Thu					
8	Friday			Fri									Fri					
9	Saturday			Sat									Sat					
10																		
11																		
12																		
13																		
14																		
15																		
16																		
17																		
18																		
19																		
20																		
21																		

Figure 2.18: Extract first three letters of the weekday names (with & without SEQUENCE)

Extract only digits from a string

Method 1 of 3

This is the first of three methods which demonstrate how to extract only digits from a cell.

An additional method will be demonstrated in *Chapter 8, SEQUENCE and other animals*.

	A	B	C	D	E	F
1						
2	Joh\$%^n1222C/ +?ecilia1224M\:\aria12344					
3						
4						
5		1222122412344				
6						
7						
8						
9	=CONCAT(IFERROR(--MID(A2,SEQUENCE(LEN(A2)),1),""))					
10						
11						
12						
13	Extract only Digits - method 1					
14						
15						
16						

Figure 2.19: Extracting only digits from a string (method 1 of 3)

Method 2 of 3

Extracting only digits from a cell (Method 2) is shown in the following screenshot:

	A	B	C	D	E
2	Joh\$%^n1222C/ +?ecilia1224M\:\aria12344				
3					
4					
5		1222122412344			
6					
7					
8					
9	=TEXTJOIN(,1,IFERROR(MID(A2,SEQUENCE(LEN(A2)),1)*1,""))				
10					
11					
12					
13	Extract only Digits - method 2				
14					
15					
16					

Figure 2.20: Extracting only digits from a string (method 2 of 3)

Method 3 of 3

Extracting only digits from a cell (Method 3) is shown in the following screenshot:

	A	B	C	D	E	F	G
1							
2	Joh\$%^n1222C/ +?ecilia1224M:aria12344						
3							
4							
5				1222122412344			
6							
7							
8							
9	=LET(a,MID(A2,SEQUENCE(LEN(A2),1),1),CONCAT(IF(ISNUMBER(--a),a,"")))						
10							
11							
12							
13	Extract only Digits - method 3						
14							
15							

Figure 2.21: Extracting only digits from a string (method 3 of 3)

Extract only unique Alphabetic characters from a string

Extracting only one occurrence of each letter in cell B4 is shown in the following screenshot:

	A	B	C	D	E	F	G	H	I	J	K	L
1												
2												
3												
4		a,d,a,b,a,b,b,c,c,d,d,d,d,e,e,		a	d	b	c	e				
5												
6												
7												
8		=LET(a, MID(B4,SEQUENCE(LEN(B4)),1),TRANSPOSE(UNIQUE(FILTER(a,a>="a"))))										
9												
10												
11												
12												
13	Extract only unique Alphabetic Characters from a string											
14												
15												
16												
17												
18												

Figure 2.22: Extract only unique alphabetic characters from a string

Split numbers and text

The two formulae combine two new functions (**TEXTBEFORE** and **TEXTAFTER**, explained in *Chapter 1, Introduction to Dynamic Array Functions in Excel 365*) with **SEQUENCE** in order to separate international country codes from the country names as is shown in the following screenshot:

Code & Country	Code	Country
44United Kingdom	44	United Kingdom
1United States	1	United States
39Italy	39	Italy
962Jordan	962	Jordan
234Nigeria	234	Nigeria

=TEXTBEFORE(A2,CHAR(SEQUENCE(26,,65))) =TEXTAFTER(A2,SEQUENCE(999))

Split Numbers from Text
 Text always begins with a Capital Letter

The trick here is that the TEXTAFTER
 fetches the text immediately after
 the highest 3-digit number (999)

Figure 2.23: Split numbers from text

Remove unwanted characters from string (2 named ranges as parameters)

In order to remove unwanted characters, the parameters shown here are the Switches (indicating which subset should be kept, for example: digits, lowercase letters, uppercase letters, and so on) and the ASCII values associated with these switches. These parameters make the solution flexible, as we can always choose which characters to remove (that is, non-printable characters) and which to keep. The following figure displays the parameters (switches and values) defined in the Name manager. These parameters will be used in the formula shown in the next section (see *Figure 2.25*).

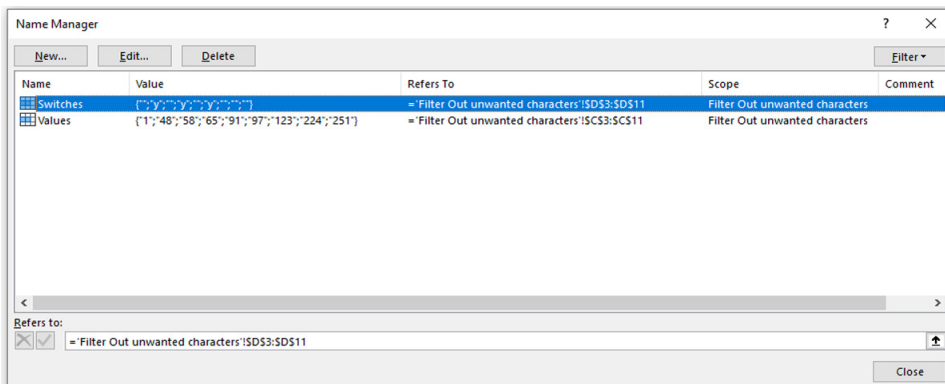


Figure 2.24: Remove unwanted characters (the Parameters pane)

Remove unwanted characters from string (Formula)

In this example, we want to keep in each cell of the Before range (A27:A33) only: digits, upper case English characters and lower-case English characters.

The results in the After range (B27:B33) show the *clean* data as can be seen in the following screenshot:

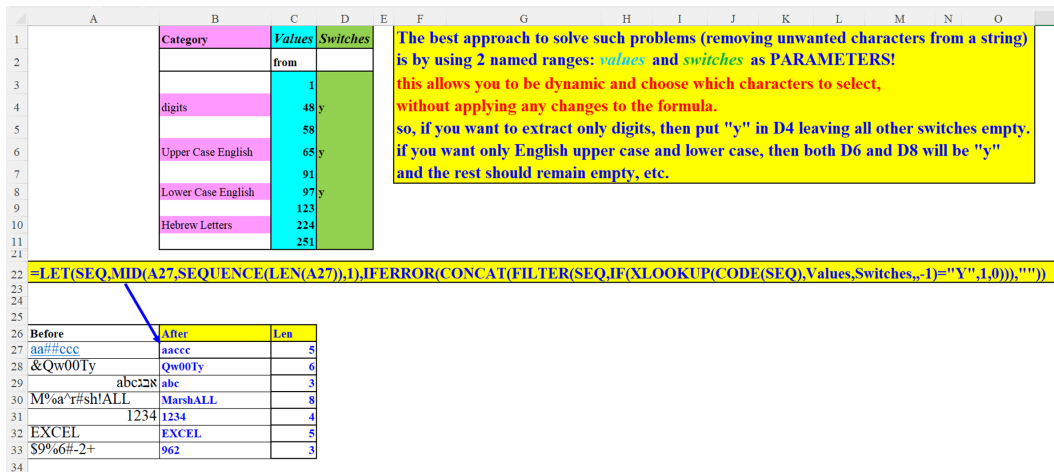


Figure 2.25: Remove unwanted characters (the Formula)

How many times does a string appear in a range

We want to find out the number of times the string defined as a parameter (cell: I1) appears in the searched range (Cell A2:A5) as shown in the following figure:

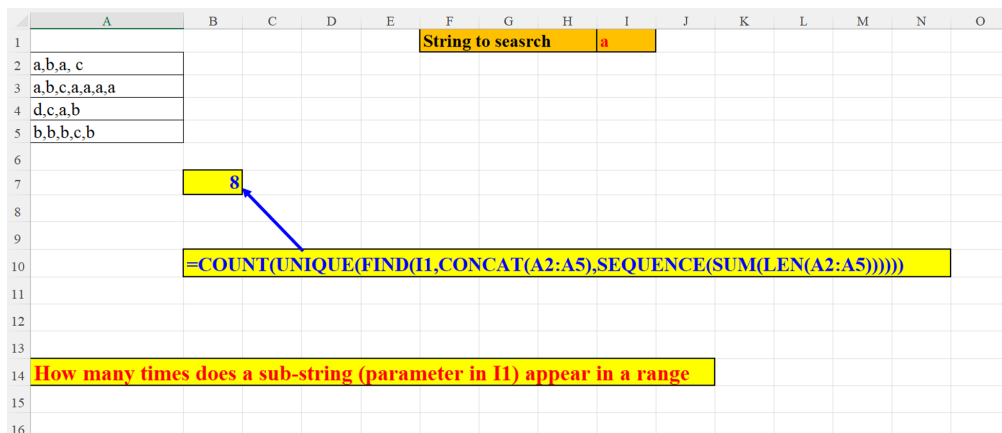


Figure 2.26: How many times does a string appear in a range

Is it a Palindrome?

A palindrome is a word, phrase or sentence that can be read exactly the same from beginning to end as well as from end to beginning. This example demonstrates the famous saying attributed to Napoleon Bonaparte, after returning from exile in the island of Elba: *Able was I ere I saw Elba* as shown in the following screenshot:

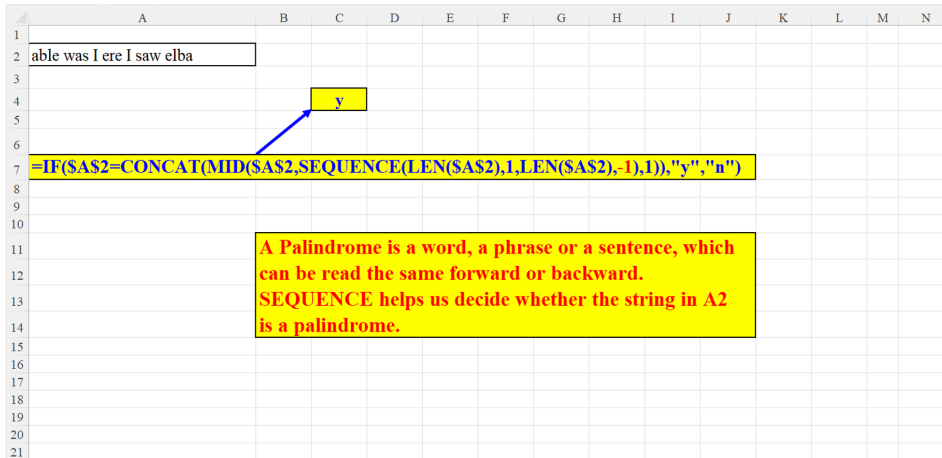


Figure 2.27: Is it a Palindrome?

Add vendor to list (the table)

If you add a new name into the **Desc.** Column (column A), and that name appears on the vendors table (defined in this figure as TVEN, column D), then by moving to the **Vendor** column (column B), that name is automatically added to the list of valid vendors in column B as shown in the following screenshot:

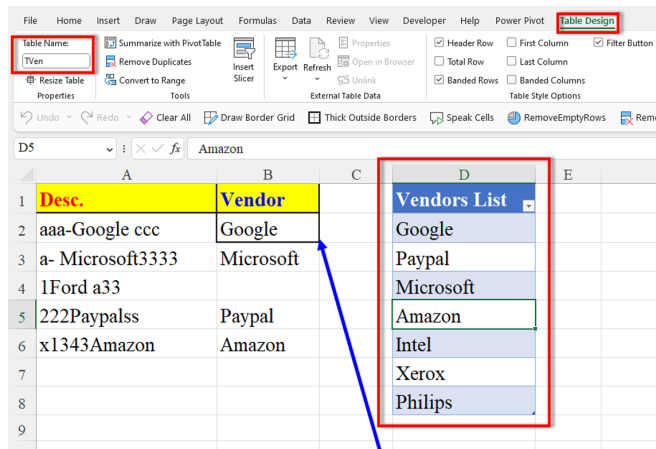


Figure 2.28: Add vendor to list (dynamic table)

Add vendor to list (the formula)

If you add a new name into the **Desc.** Column (column A), and that name appears on the vendors table (defined in the previous figure as TVEN column D), then by moving to the **Vendor** column (column B), that name is automatically added to the list of valid vendors in column B as shown in the following screenshot:

1	Desc.	Vendor	Vendors List
2	aaa-Google ccc	Google	Google
3	a- Microsoft3333	Microsoft	Paypal
4	1Ford a33		Microsoft
5	222Paypals	Paypal	Amazon
6	x1343Amazon	Amazon	Intel
7			Xerox
8			Philips
9			
10			
11			
12			
13	=IFERROR(LOOKUP(SUM(IF(ISNUMBER(FIND(TVen,A2)),SEQUENCE(COUNTA(TVen))),SEQUENCE(COUNTA(TVen),TVen),""))		
14			
15			
16	If a vendor name found in column A appears in the Vendors List (Table Tven on column D),		
17	then add the vendor's name in column B		
18			
19			
20			

Figure 2.29: Add vendor to list (the formula)

Remove all digits from the string

How to remove only the digits from a cell leaving all other characters intact can be seen illustrated in the following figure:

1	Before	After
2	a11b2c3	abc
3		
4		
5		
6	=LET(x,MID(A2,SEQUENCE(LEN(A2)),1),CONCAT(IF(ISNUMBER(--x),"",x)))	
7		
8		
9		
10		
11		
12		
13	Remove Digits from String	
14		
15		
16		
17		
18		
19		
20		

Figure 2.30: Remove all the digits from the string (cell A2)

Move first name from end of cell to the beginning

What happens if we receive a file of names where the first name appears at the end, and we want to move it to the beginning?

Here is a solution:

	A	B	C	D	E	F	G	H	I
1	Before			After					
2	Cohen Yossi			Yossi Cohen					
3	Roosevelt Dylan Franklin			Franklin Dylan Roosevelt					
4									
5									
6									
7									
8	=LET(x,TEXTSPLIT(A2," "),TEXTJOIN(" ",SORTBY(x,SEQUENCE(COLUMNS(x)),-1)))								
9									
10									
11									
12									
13									
14									
15									
16	Move First name from end of cell to the beginning								
17									
18									
19									

Figure 2.31: Move first name from the end to the beginning

Reverse String Method 1

Reverse a string (method 1 of 2) – Using **TEXTJOIN** with **SEQUENCE**:

	A	B	C	D
1	Before		After	
2	Jenny I've got your number		rebmun ruoy tog ev'l ynneJ	
3				
4				
5				
6				
7				
8	=TEXTJOIN(""," ",MID(A2,SEQUENCE(LEN(A2),1,LEN(A2),-1),1))			
9				
10				
11				
12				
13				
14				
15	Reverse String - Method 1			
16				
17				
18				
19				
20				

Figure 2.32: Reverse string (Method 1)

Method 2

Reverse a string (method 2 of 2) – Using **CONCAT** with **SEQUENCE** as shown in the following figure:

	A	B	C	D
1	Before		After	
2	Jenny I've got your number		rebmun ruoy tog ev'I ynneJ	
3				
4				
5				
6				
7				
8	=CONCAT(MID(A2,SEQUENCE(LEN(A2)),LEN(A2),-1),1))			
9				
10				
11				
12				
13				
14				
15	Reverse String - Method 2			
16				
17				
18				
19				
20				
21				

Figure 2.33: Reverse string (Method 2)

Sort Text in alphabetical order

The string in cell A2, which contains only lowercase letters, is sorted in alphabetical order. The “a” defined by the LET function replaces the reference to cell A2:

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Before		After										
2	dabecf		abcdef										
3													
4													
5													
6	=CONCAT(LET(a,A2,SORT(MID(a,SEQUENCE(LEN(a)),1))))												
7													
8													
9													
10													
11													
12	Sort text alphabetically												
13													
14													

Figure 2.34: Sort text in alphabetical order

How many words are there in the cell without the separator (SEQUENCE)

In this example, we count the number of *words* in the cell. A word, for that matter, is any combination of characters which is different from the separator. The separator is defined in cell E1 as shown in the following screenshot:

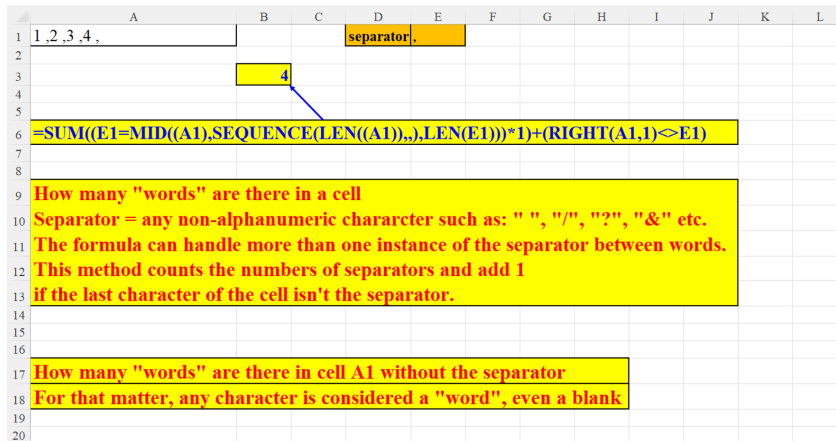


Figure 2.35: How many words are there in the cell A2 without the separator? (SEQUENCE function)

How many words are there in the cell without the separator (TEXTSPLIT)

A shorter alternative to the previous example, using the more advanced function: **TEXTSPLIT** (which is explained in *Chapter 1, Introduction to Dynamic Array Functions in Excel 365*):

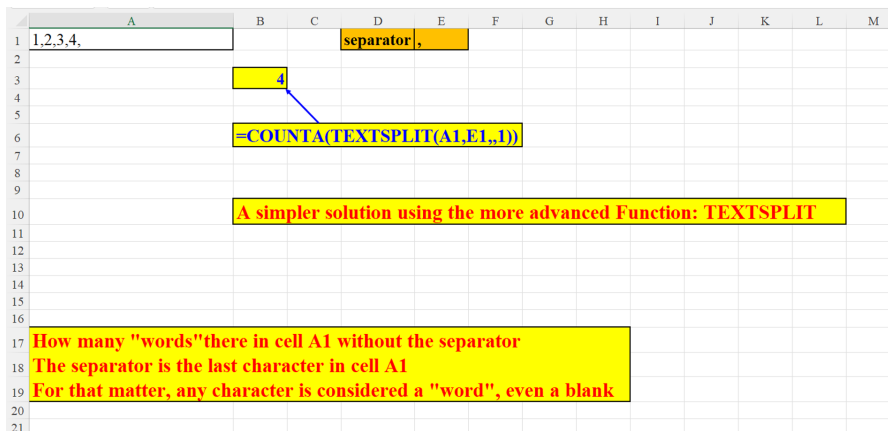


Figure 2.36: How many words are there in cell A2 without the separator? (TEXTSPLIT function)

Off with their heads

In this example, we *chop off* one character from the original string (defined in Cell F1) as shown in the following figure:

	A	B	C	D	E	F	G	H	I	J	K
1	four score and seven years ago					four score and seven years ago				<----	The original sentence
2	our score and seven years ago										
3	ur score and seven years ago										
4	r score and seven years ago										
5	score and seven years ago										
6	core and seven years ago										
7	ore and seven years ago										
8	re and seven years ago										
9	e and seven years ago										
10	nd seven years ago										
11	d seven years ago										
12	seven years ago										
13	ven years ago										
14	en years ago										
15	n years ago										
16	years ago										
17	years ago										
18											
19											
20											
21											
22											

=RIGHT(F1,SEQUENCE(LEN(F1),,LEN(F1),-1))

**Each consecutive cell contains one less character
We start with the full string, and we "decapitate"
one character at-a-time from the beginning**

**"Off with their heads..."
Said the Queen of Hearts in Lewis Carroll's "Alice's Adventures in Wonderland"**

Figure 2.37: "off with their heads" – chopping one character at a time from the "head" of the string

Extract only digits and add a separator

Removing all non-numeric characters from cell A3. Each group of consecutive digits is then separated by the separator defined in cell F2:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1														
2	Before		After			Separator	*							
3	a123b456c789		123*456*789											
4														
5														
6														
7														
8														
9														
10														
11														
12														
13														
14														
15														
16														
17														
18														
19														

=SUBSTITUTE(TRIM(CONCAT(IFERROR(--MID(\$A\$3,SEQUENCE(LEN(\$A\$3),1)," ")), " ")), " ",\$F\$2)

**Extract only digits.
If separated by any character, the defined separator (F1)
will separate each group of consecutive digits**

Figure 2.38: Extract only digits and add a separator (defined in cell F2)

How many lower-case letters are there in the cell

Method 1

Counting the number of lower-case letters in each cell (A2:A4) - Method 1:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Text		Number of lower-case letters													
2	aAb4567		2													
3	aAb456c		3													
4	a1d3c3e4		4													
5																
6																
7																
8																
9	=SUM(-(CODE(MID(A2,SEQUENCE(LEN(A2),,1))>=CODE("a"))*(CODE(MID(A2,SEQUENCE(LEN(A2),,1))<=CODE("z"))))															
10																
11																
12																
13																
14																
15																
16																
17																
18																
19																
20																
21																
22																

How many lower-case letters are there in the cell - Method 1

Figure 2.39: How many lower-case letters are there in the cell (Method 1)

Method 2

Counting the number of lower-case letters in each cell (A2:A4) – Method 2.

This formula uses the more advanced function LET (mentioned in *Chapter 1, Introduction to Dynamic Array Functions in Excel 365*):

	A	B	C	D	E	F	G	H	I	J	K
1	Text		Number of lower-case letters								
2	aAb4567		2								
3	aAb456c		3								
4	a1d3c3e4		4								
5											
6											
7											
8											
9	=LET(a,CODE(MID(A2,SEQUENCE(LEN(A2),,1)),SUM((a>=CODE("a"))*(a<=CODE("z"))))										
10											
11											
12											
13											
14											
15											
16											
17											
18											
19											

How many lower-case letters are there in the cell - Method 2

Figure 2.40: How many lower-case letters are there in the cell (Method2)

All Greek letters in one formula

Generating the list of all the Greek letters. The Greek characters are not a part of the standard ASCII character set; therefore, we used the **UNICHAR** function (that uses the more comprehensive **UNICODE** character set) instead of the **CHAR** function as illustrated in the following screenshot:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Greek Letters																
2	α																
3	β																
4	γ																
5	δ																
6	ε																
7	ζ																
8	η																
9	θ																
10	ι																
11	κ																
12	λ																
13	μ																
14	ν																
15	ξ																
16	ο																
17	π																
18	ρ																
19	ς																
20	σ																
21	τ																

Figure 2.41: All Greek letters in one formula

Find last word in cell

Method 1 of 3

Finding the last word in cell A1 (Using **LOOKUP** and **SEQUENCE**) – Method 1:

	A	B	C	D	E	F	G	H	I	J	K	L
1	My name is Meni Porat											
2												
3												
4												
5												
6												
7												
8												
9												
10												
11												
12												
13												
14												
15												
16												
17												
18												
19												
20												

Figure 2.42: Find last word in cell using **SEQUENCE** (Method 1 of 3)

Method 2 of 3

Finding the last word in cell A1 (Using **MAX**, **FIND** and **SEQUENCE**) – Method 2:

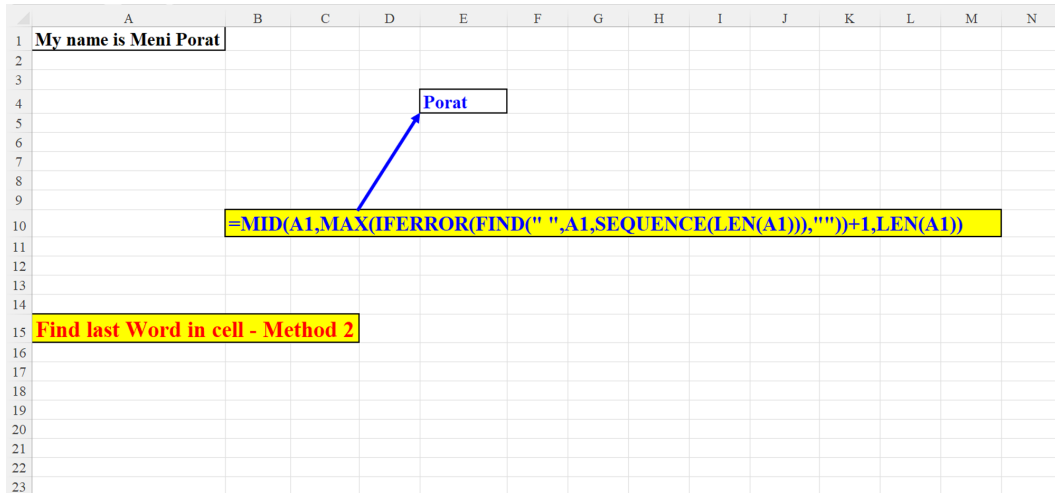


Figure 2.43: Find last word in cell using SEQUENCE (Method 2 of 3)

Method 3 of 3

A shorter alternative to the previous examples (Figures 2.42 and Figure 2.43), using the more advanced function: **TEXTAFTER** (which was explained in Chapter 1, Introduction to Dynamic Array Functions in Excel 365):

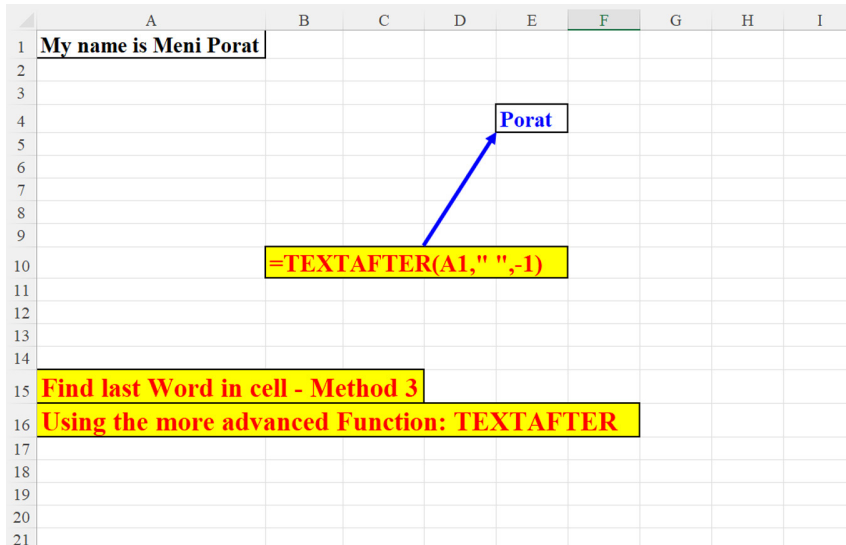


Figure 2.44: Find last word in cell – using TEXTAFTER (Method 3 of 3)

Number of characters in cell (without the separator)

In our example, the separator defined in the parameter is " " (blank). The formula "trims" any additional blanks between the words (as, for example, between "saw" and "Elba" where there are two blank characters).

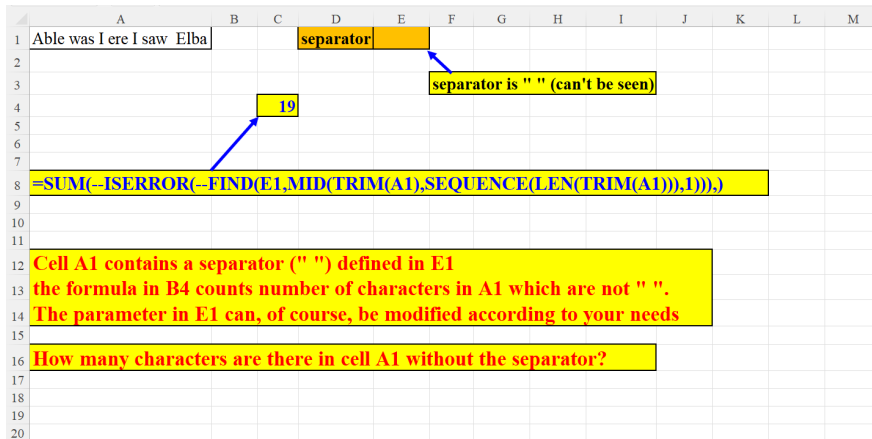


Figure 2.45: Number of characters in cell without the separator

Number of non-empty cells in a column

Finding the number of non-empty cells in the entire column is shown in the following figure:

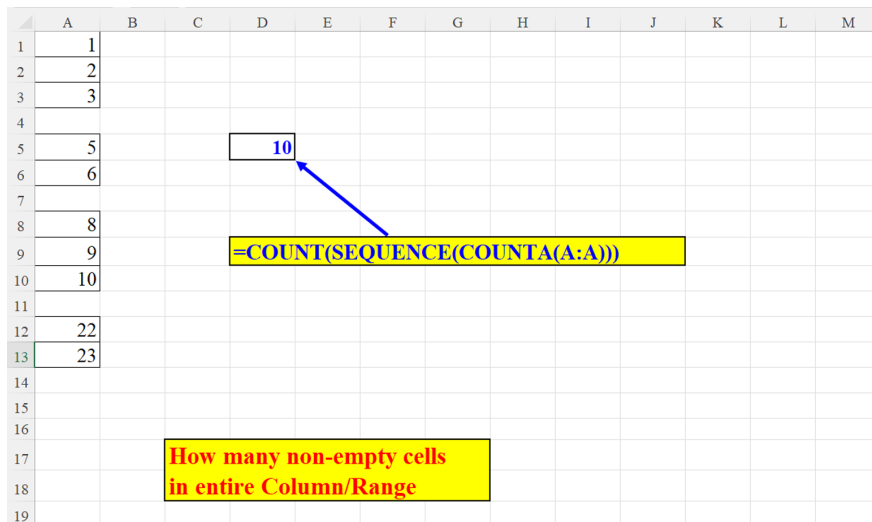


Figure 2.46: Number of non-empty cells in column A

Strip leading and trailing digits

Method 1 of 2

Removing leading and trailing digits using **LET**, **SEQUENCE** and **CONCAT** (method 1):

	A	B	C	D	E	F	G	H	I	J	K
1	Before			After							
2	123banana123			banana							
3	12cucuricu122			cucuricu							
4	ccucu			ccucu							
5	12334cucu			cucu							
6	cucu123			cucu							
7											
8											
9											
10	=LET(fnd,MID(\$A2,SEQUENCE(LEN(\$A2)),1),CONCAT((IF(ISNUMBER(--fnd),"",fnd))))										
11											
12											
13											
14											
15											
16											
17											
18											
19											
20											
21											

Strip leading and trailing digits - Method 1

Figure 2.47: Strip leading and trailing digits (Method 1)

Method 2 of 2

Removing leading and trailing digits using **LET**, **SEQUENCE** and **TEXTJOIN** (method 2):

	A	B	C	D	E	F	G	H	I	J	K	L
1	Before			After								
2	123banana123			banana								
3	12cucuricu122			cucuricu								
4	ccucu			ccucu								
5	12334cucu			cucu								
6	cucu123			cucu								
7												
8												
9												
10	=LET(fnd,MID(\$A2,SEQUENCE(LEN(\$A2)),1),TEXTJOIN("","",(IF(ISNUMBER(--fnd),"",fnd))))											
11												
12												
13												
14												
15												
16												
17												
18												
19												
20												
21												
22												

Strip leading and trailing digits - Method 2

Figure 2.48: Strip leading and trailing digits (Method 2)

Increasing Text from end to start

We start from the last character of the sentence (in cell D1) and each consecutive cell contains one more character, till we reach the full-blown sentence:

	A	B	C	D	E	F	G	H	I	J	K	L
1	o			four score and seven years ago								
2	go											
3	ago											
4	ago											
5	s ago											
6	rs ago											
7	ars ago											
8	ears ago											
9	years ago											
10	years ago											
11	n years ago											
12	en years ago											
13	ven years ago											
14	even years ago											
15	seven years ago											
16	seven years ago											
17	d seven years ago											
18	nd seven years ago											
19	and seven years ago											

Figure 2.49: “Increasing” text from End to Start

Increasing Text from start to end

We start from the first character of the sentence (in cell D1) and each consecutive cell contains one more character, till we reach the full-blown sentence as shown in the following screenshot:

	A	B	C	D	E	F	G	H
1	f			four score and seven years ago				
2	fo							
3	fou							
4	four							
5	four							
6	four s							
7	four sc							
8	four sco							
9	four scor							
10	four score							
11	four score							
12	four score a							
13	four score an							
14	four score and							
15	four score and							
16	four score and s							
17	four score and se							
18	four score and sev							
19	four score and seve							

Figure 2.50: “Increasing” text from Start to End

Hebrew Gematria (Formula)

Gematria is the practice of assigning numerical values to Alphabet letters and summing up these values (for a word, phrase, or sentence). Gtable (A6:E33 as shown in figure 2.52) is an Excel table where for each Hebrew letter a numeric value is given as shown in the following screenshot:

Text	Value
מבא	4

Formula: `=SUM(IFNA(LOOKUP(MID(A2,SEQUENCE(LEN(A2)),1),Gtable[Letter],Gtable[Numeric value]),0),)`

Letter	Numeric value
א	1
ב	2
ג	3
ד	4
ה	5
ו	6
ז	7
ח	8
ט	9
י	10
כ	20
ך	20
ל	30
מ	40
ם	40
נ	50
ך	50
ס	60
ע	70
פ	80

Hebrew Gematria
 Each letter in the Hebrew Alphabet has a unique value
 The Value in B2 is the sum of all the characters' value in A2.
 The value for each character is being LOOKUP-ed in the Gtable

Figure 2.51: Hebrew Gematria (Formula)

Hebrew Gematria (Gtable – Translation table)

This is the translation table used in the solution displayed in the previous section: **Hebrew Gematria (Formula)** as shown in the following screenshot:

Letter	Numeric value
א	1
ב	2
ג	3
ד	4
ה	5
ו	6
ז	7
ח	8
ט	9
י	10
כ	20
ך	20
ל	30
מ	40
ם	40
נ	50
ן	50
ס	60
ע	70
פ	80
ף	80
צ	90
ץ	90
ק	100
ך	200
ש	300
ת	400

Figure 2.52: Hebrew Gematria (Gtable – Translation table)

Extract only Country Names

This is how to extract only the country names from a long string:

The inner function (**TEXTSPLIT**) splits the text by any digit (0-9) which removes all the digits. Then, the outer function (**FILTER**) filters out every comma (","), because its length is 1:

1	China2,000,000 2India1,450,000 3United States1,390,000 44North Korea1,200,000 543Russia850,000				
2					
3					
4		China	India	United States	North Korea
5					Russia
6					
7					
8		=LET(x,TEXTSPLIT(A1,CHAR(SEQUENCE(10,,48))),FILTER(x,LEN(x)>1))			
9					
10					
11					
12					
13					
14		Extracting only country names from a long string:			
15		TEXTSPLIT "splits" the string by digits (0-9) which removes them from the string			
16		then, we filter out only cells whose length > 1, which removes all the commas			
17					
18		Extract only country names			
19					

Figure 2.53: Extract only country names

How many occurrences of a String starting from a certain position

We are looking for the number of times the parameter string (defined in cell: D2) appears in cell A2, but we start at the position specified in the “starting from” parameter (cell: F2) as shown in the following screenshot:

	A	B	C	D	E	F	G	H
1	Fruits			String to search		Starting from		
2	oranges, apples, oranges, oranges, apples, oranges			oran		10		
3								
4								
5								
6								
7								
8			3					
9								
10								
11	=SUM(N(D2=MID(A2,SEQUENCE(LEN(A2),,F2),LEN(D2))))							
12								
13								
14	How many times does the string defined in D2 ("oran")							
15	Appear in cell A2, starting from the position defined in F2 (10)?							
16								
17								
18								
19								
20								

Figure 2.54: How many times does the search string appear, starting from a certain position

Remove Diacritics from Hebrew words

The Hebrew languages has Diacritics, and special symbols that help pronounce the syllables correctly (since there are no vowels in Hebrew).

Sometimes, we want to remove these Diacritics as they usually do not appear in books, articles, internet sites, and so on. This is illustrated in the following figure:

	A	B	C	D	E	F	G	H	I	J	K	L
1	Before	After										
2	בְּרֵאשִׁית	בראשית										
3	בְּרֵא	ברא										
4	אֱלֹהִים	אלהים										
5	אֵת	את										
6	הַשָּׁמַיִם	השמים										
7	וְאֵת	ואת										
8	הָאָרֶץ	הארץ										
9												
10												
11												
12	=CONCAT(IF(UNICODE(MID(A2,SEQUENCE(LEN(A2),,1))>1487,MID(A2,SEQUENCE(LEN(A2),,1),"")))											
13												
14												
15												
16												
17												
18												
19	Removing Diacritics from Hebrew Text											
20												

Figure 2.55: Remove Diacritics from Hebrew Words

Is it a Palindrome (Arabic)

A palindrome is a word, phrase or sentence that can be read exactly the same from beginning to end as well as from end to beginning.

Here we check three Arabic words to find out whether they are palindromes or not.

Another example of a Palindrome checking can be found above, in the section: *Is it a Palindrome?* (Figure 2.27)

	A	B	C	D	E	F	G	H	I	J	K	L	M
1													
2		انا	y										
3		باب	y										
4		بابا	n										
5													
6													
7													
8													
9													
10													
11													
12	=IF(\$A\$2=CONCAT(MID(\$A\$2,SEQUENCE(LEN(\$A\$2),1,LEN(\$A\$2),-1),1),"y","n"))												
13													
14													
15													
16													
17	Is it a Palindrome - Arabic Text												
18	A palindrome is a word, a phrase or a sentence												
19	that reads : from beginning to end												
20	same as: from end to beginning												
21													

Figure 2.56: Is it a Palindrome (Arabic)

Convert Hebrew letters into English letters

This formula helps solve an issue that is very common for bilingual keyboards users. If, for example, you type both in Hebrew and English, sometimes you unintentionally forget to switch to English while typing in Hebrew. The result is *gibberish* (as can be

seen in cells A1 and C1). So, instead of erasing the contents of the entire cell, use the formula to convert “gibberish” into English as shown in the following screenshot:

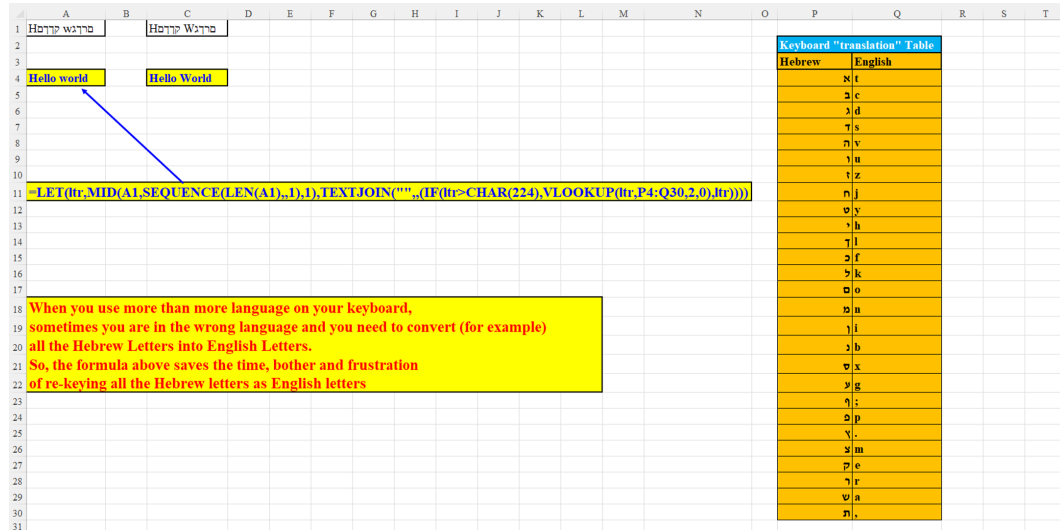


Figure 2.57: Convert Hebrew Letters into English Letters

Fetch description of Nth item of a non-sorted Key

We want to find the Nth (in our case: 5th, defined as a parameter in cell: E2) for Cat. No. 2 (defined as a parameter in cell: D2) as shown in the following screenshot:

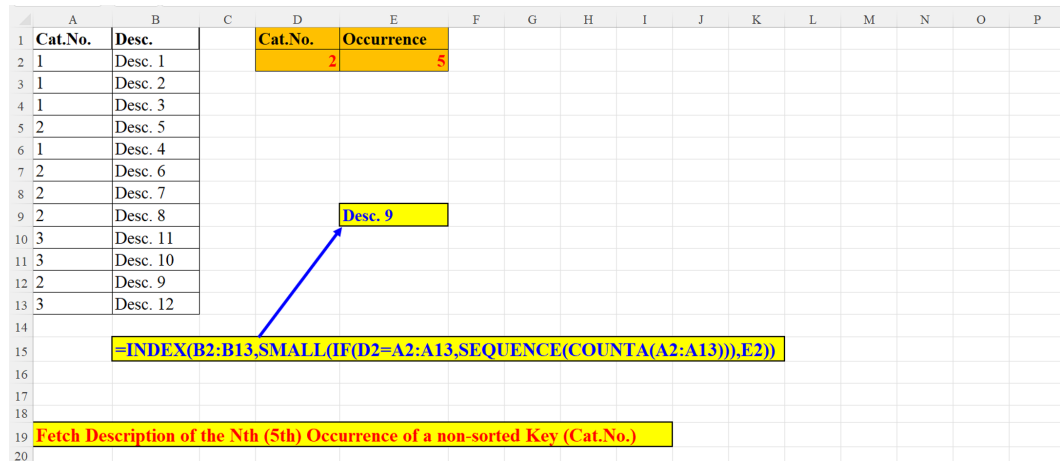


Figure 2.58: Fetch Description of the Nth (5th) occurrence of a non-sorted Key (Column A: Cat.No.)

Extract letters only from a chosen language (Formula)

The formula in the following figure extract only letters of the language chosen in cell J2 (which is a validation list, as can be seen in the next section):

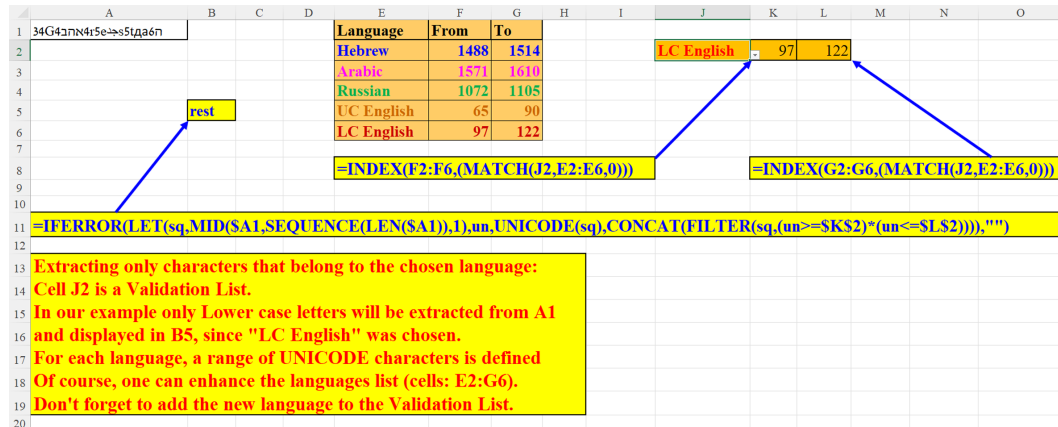


Figure 2.59: Extract only letters from the Language chosen in J2 (Validation List)

Extract letters only from a chosen language (Validation list)

Here is the Validation list in cell J2 that selects the language whose letters should be extracted from cell A1 (see previous section, *Extract letters only from a chosen language (Formula)*):

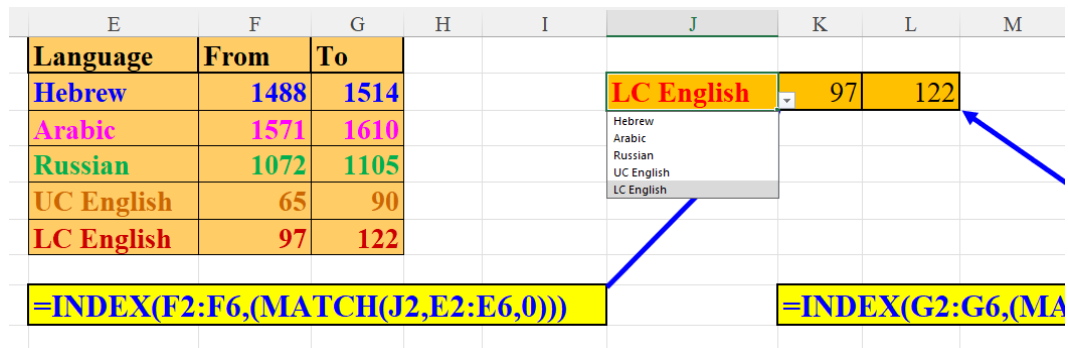


Figure 2.60: The Validation List (Cell: J2) contains all the Languages on the list (E1:G6)

Gematria in English

Method 1

Gematria (as explained above, in the section, *Hebrew Gematria (Formula)*, see Figure 2.51) is the practice of assigning numerical values to Alphabet letters and summing up these values (for a word, phrase or sentence). The dataset (K1:L27) is an Excel table where for each English letter a numeric value is given.

Method 1: using SUM and LOOKUP:

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Text		Value			Simple English Gematria Chart:					Letter	Value	
2	King Henry		111			A=1	J=10	S=19			a	1	
3						B=2	K=11	T=20			b	2	
4						C=3	L=12	U=21			c	3	
5						D=4	M=13	V=22			d	4	
6						E=5	N=14	W=23			e	5	
7	=SUM(IFNA(LOOKUP(MID(A2,SEQUENCE(LEN(A2)),1),K2:K27,L2:L27),0),)												
8						F=6	O=15	X=24			f	6	
9						G=7	P=16	Y=25			g	7	
10						H=8	Q=17	Z=26			h	8	
11						I=9	R=18				i	9	
12											j	10	
13											k	11	
14	What is Gematria? Gematria is the practice of assigning numerical values to letters and summing up these values (for a word, phrase or sentence)												
15	The method used here is LOOKUP-ing each English letter's value in a table.												
16											l	12	
17											m	13	
18											n	14	
19											o	15	
20											p	16	
21	You can compare results interactively with this internet site:												
22	https://calculator-online.net/gematria-calculator/												
23											q	17	
24											r	18	
25											s	19	
26											t	20	
27											u	21	
											v	22	
											w	23	
											x	24	
											y	25	
											z	26	

Figure 2.61: Gematria in English – Method 1

Method 2

Gematria (as explained above, in the section *Hebrew Gematria (Formula)*) is the practice of assigning numerical values to alphabet letters and summing up these values (for a word, phrase or sentence).

Method 2: We do not need a table, as in the previous solution. We subtract 96 from each ASCII value (since a = 97, b = 98, c = 99 etc.). In case there are both upper-case and lower-case letters, we convert them all to lower-case since their values are the same. The formula ignores any characters that do not belong to the English alphabet as shown in the following figure:

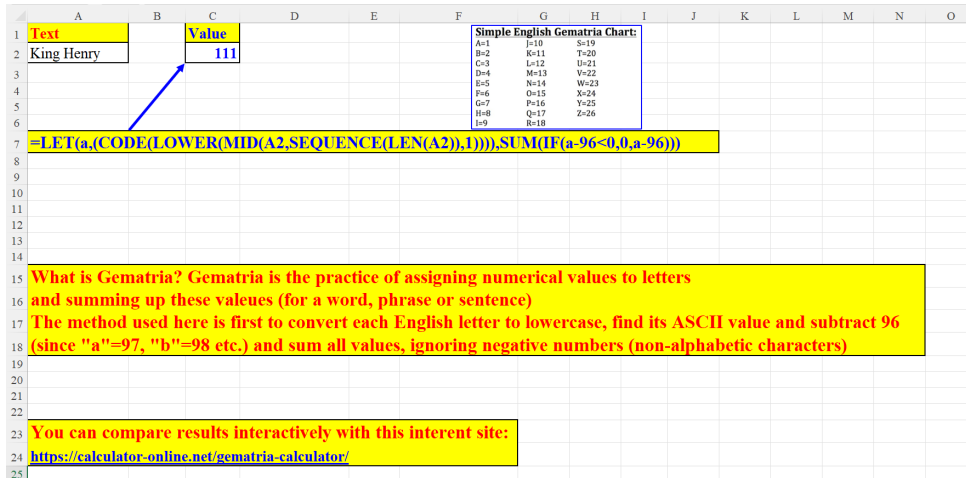


Figure 2.62: Gematria in English – Method 2

How many Words are there in a Range?

This example is similar to the one demonstrated in Section: How many words are there in the cell (version 2), Figure 2.3. Both examples use the **TEXTSPLIT** function (explained in Chapter 1, Introduction to Dynamic Array Functions in Excel 365). Cells A1:A3 hold several words separated by space/s. The **TRIM** function removes spaces from the beginning and end of each cell. In addition, it leaves only one space between words. Then the **TEXTJOIN** function combines all the words in the range where a delimiter (space) is inserted between every text item. The **TEXTSPLIT** splits the string by that same delimiter into distinct cells. Finally, the **COUNTA** function counts the number of cells created by **TEXTSPLIT**:

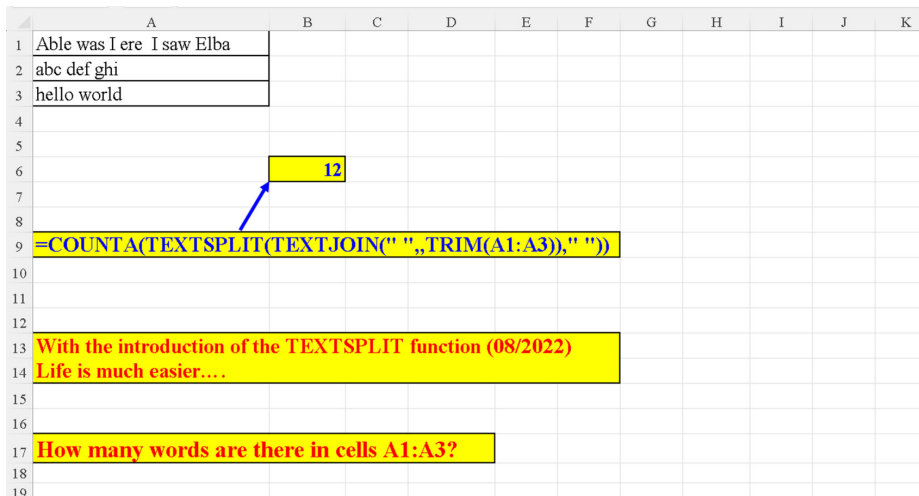


Figure 2.63: How many words are there in a range

Extract only non-digits from String

The string in cell A2 contains both digits and non-digits. We want to remove all the digits from the string. We split the string into its components and build it back (without the digits) by concatenating each and every character of the original string that is not a digit. The **LET** function is used to avoid repetition of that part of the formula which splits the string **MID(\$A2,SEQUENCE(LEN(\$A2)),1)** as shown in the following screenshot:

	A	B	C	D	E	F	G
1	Before		After				
2	J1o2h5h 45F. 3Ke4n4n4e9dy 1962		Johh F. Kennedy				
3	0A1b2r3a4h5a6m7 L8i9n1c12o13l14n 1862		Abraham Lincoln				
4							
5							
6							
7							
8	=LET(sq,MID(\$A2,SEQUENCE(LEN(\$A2)),1),CONCAT(IF(ISNUMBER(--sq),"",sq)))						
9							
10							
11							
12	Extract only not-numbers from string						
13							
14							
15							

Figure 2.64: Extract only non-digits from string

Find Unicode value for any character in the string, no matter which language

The **UNICODE** combined with **SEQUENCE** brings each character's **UNICODE** value. It does not matter if we have text from more than one language:

	A	B	C	D	E	F	G	H	I	J	K
1	доброе утро صباح الخير										
2										1076	
3										1086	
4										1073	
5										1088	
6										1086	
7										1077	
8										32	
9										1091	
10										1090	
11										1088	
12										1086	
13	Extracting Unicode value for any character in any language										32
14										1589	
15										1576	
16										1575	
17										1581	
18										32	
19										1575	
20										1604	
21										1582	
22										1610	
23										1585	

Figure 2.65: Find Unicode value for any character

Conclusion

This chapter illustrated more than 60 cases of SEQUENCE usage with textual operations.

The next chapter will discuss the utilization of SEQUENCE when dealing with numbers.

Points to remember

- SEQUENCE can handle text from any direction (from beginning to end or: from end to beginning).
- SEQUENCE can easily split any text, in any language.
- SEQUENCE can easily extract parts of text: only letters, only digits, only special characters, and so on.
- SEQUENCE can effortlessly “translate” text.
- SEQUENCE can duplicate strings.
- SEQUENCE can generate series of strings, for example: The Alphabet of languages.
- SEQUENCE is capable of transposing text: vertical to horizontal and vice versa.
- SEQUENCE can assist us with removing unnecessary characters from text.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



CHAPTER 3

Using SEQUENCE with Numbers

Introduction

This chapter will demonstrate how to employ the **SEQUENCE** function when dealing with numbers. In a spreadsheet, numbers are the *bread and butter* of Excel. The **SEQUENCE** function has many tricks “up her sleeve” to handle sundry cases where manipulation with numbers is needed. Sometimes, as we will see in several instances, the data is inexistent, and the **SEQUENCE** function has to generate data out of nothing.

Structure

In this chapter, we will discuss the following topics:

- Examples of **SEQUENCE** with numbers
 - Five methods to generate 12 positive integers
 - Five methods to generate 12 negative integers
 - Descending **SEQUENCE** – Two methods
 - Duplicate cell horizontally
 - Duplicate cell vertically
 - Duplicate numbers

- Creating a vertical **SEQUENCE** of numbers – 2 methods
- Find missing numbers in a list
- Reverse a number
- Reverse a horizontal ascending array
- Reverse a horizontal descending array
- **SEQUENCE** of odd and even numbers
- Sum all digits in a cell which has only digits
- Sum all digits in a cell which has digits and text
- Sum every Nth row
- Sum the largest N numbers
- Sum the smallest N numbers
- Two tricks with **SEQUENCE** (ROW())
- SUM (**SEQUENCE** (virtual array))
- Alternate 1s and 0s
- Dynamic **SEQUENCE**
- Two methods to extract a number from string's end
- Two methods to extract a number from string's start
- Find N largest numbers (Asc.)
- Find N largest numbers (Desc.)
- How many columns in a sheet
- How many digits
- Reverse numbers horizontally by a parameter
- Reverse order of **SEQUENCE** of numbers
- Subject with the highest score
- **SEQUENCE** based on number of unique values
- Dynamic frequency based on dynamic bins
- **SEQUENCE** column
- Building a chessboard in three steps
- Creating N-digit number with the same digit repeated N times

Objectives

After reading this chapter, you will be able to perform many Excel tasks, like handling numbers, generating an array, reversing an array's order, summing part of an array, and so on. Applying the methods exemplified and explained in this chapter will save you a lot of time and effort. The **SEQUENCE** function is extremely useful with handling numbers in myriad situations in Excel. Understanding and practicing this chapter's examples will immensely improve your level of aptitude in Excel.

Examples of SEQUENCE with numbers

We will now discuss in detail the examples of applying the **SEQUENCE** function in numerical operations. Please be aware of the fact that we are not going to deliberate upon mathematical issues, but only on numbers as integers. The integration of **SEQUENCE** with mathematics will be explained in *Chapter 7, SEQUENCE - The Ancilla of Math*.

Five methods to generate 12 positive integers

The following five examples exhibit five methods of creating a vertical array of 12 numbers, using only the first argument of the function: **ROWS**. The number of desired rows (12) is contrived by five different methods: explicit number, Excel generated number (The **ROWS()** function), the number of the year's months from a fictitious date and a range of column names which yields the desired number.

Method 1

This is the simplest way to generate the first twelve natural numbers with **SEQUENCE**:

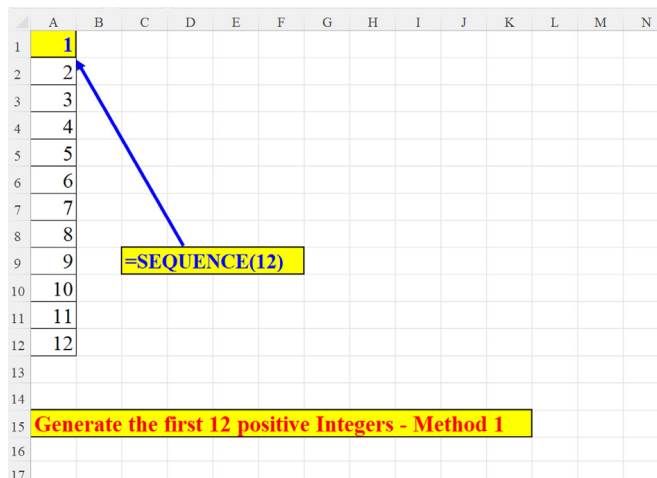


Figure 3.1: Generate 12 positive integers - Method 1

Method 2

Another variation with **SEQUENCE** is using the **ROW** function. This function returns the (row) number of a reference:

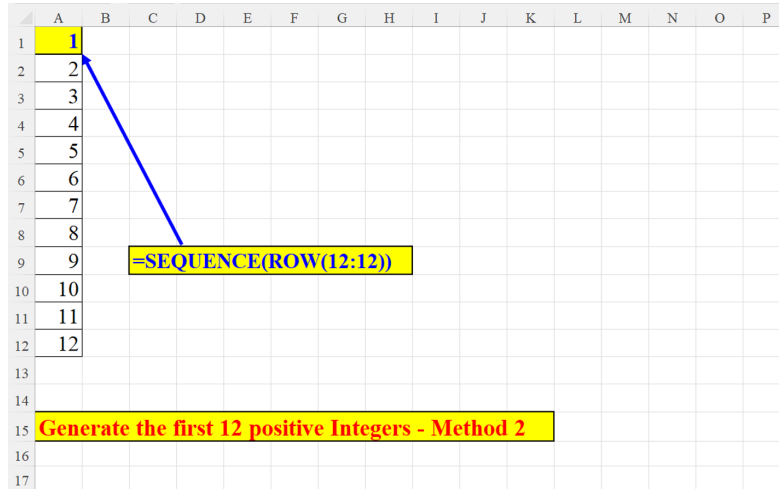


Figure 3.2: Generate 12 positive integers - Method 2

Method 3

The **SEQUENCE** function combined with **ROWS**. The **ROWS** function returns the count of rows of the reference. The reference can be a range of cells (that is, A1:A12) or a range of numbers (as in our example):

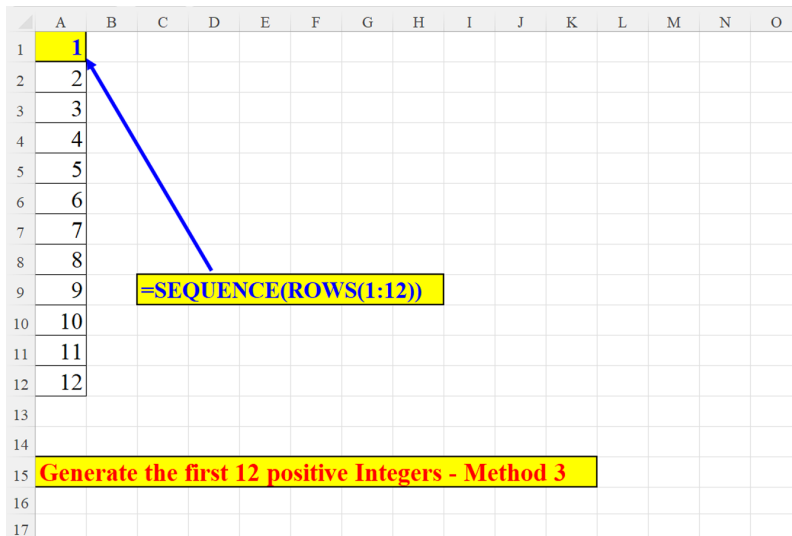


Figure 3.3: Generate 12 positive integers - Method 3

Method 4

Here, we use the number of months in a year to produce the first twelve integers. The expression: **DATE(1,1,)** creates the date: 31/12/1900 and the **MONTH** function that wraps it, yields the month number of this date: 12 as shown in the following figure:

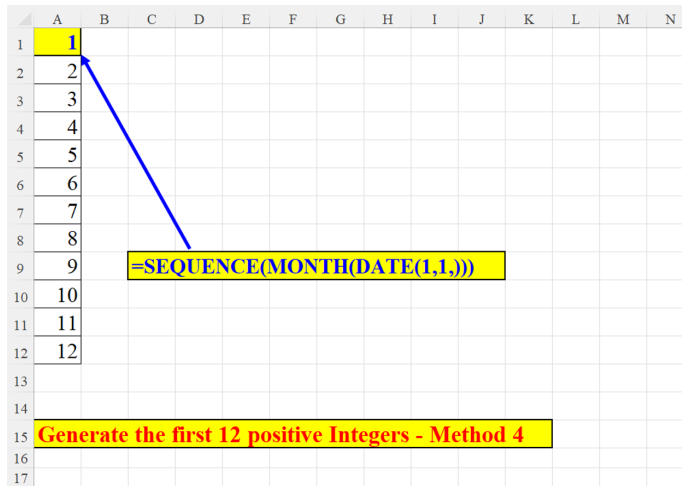


Figure 3.4: Generate 12 positive integers - Method 4

Method 5

The **COLUMNS** function returns the number of columns in a reference. In our example, the range of columns A:L has twelve columns:

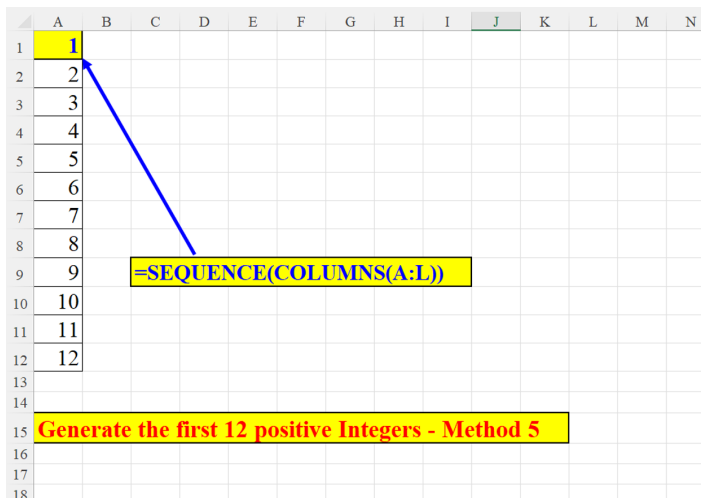


Figure 3.5: Generate 12 positive integers - Method 5

Five methods to generate 12 negative integers

In this section, we will look at five techniques to create a **SEQUENCE** of 12 negative numbers. Some of them use the same techniques used above to initiate positive numbers, but the result is multiplied by -1 . Other techniques use more than one argument of the **SEQUENCE** function, where the Starting number and Increment are both negative. The last example illustrates a lesser-known *trick* in Excel: instead of beginning the formula with the obvious equal sign (=) which precedes any formula, we establish the negative numbers with the minus sign (-) without specifying the =.

Method 1

Generating 10 consecutive negative integers. The third argument (-1) tells the function which number to start with. The fourth argument (-1) tells that the increment is negative as shown in the following figure:

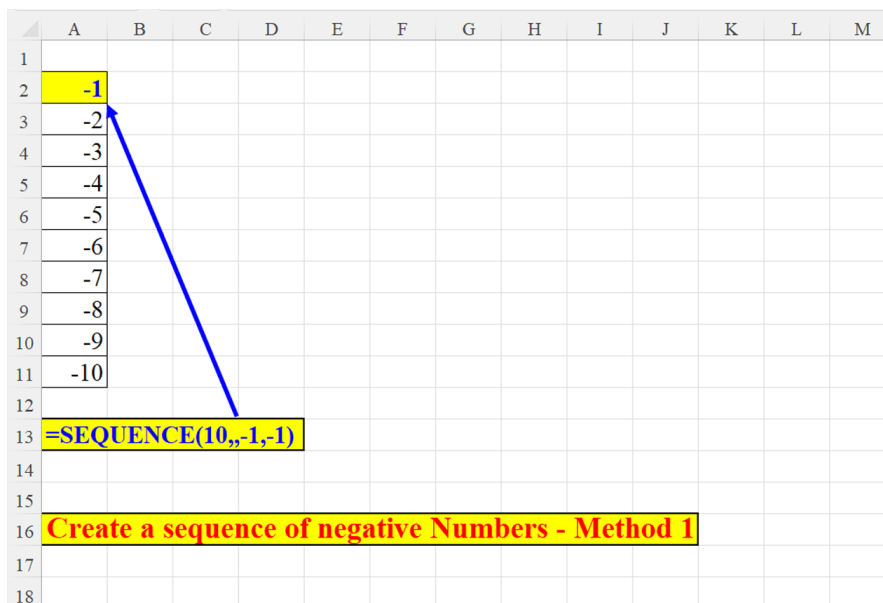


Figure 3.6: Negative SEQUENCE – Method 1

Method 2

Multiplication of a positive array by (-1) turns the entire array into a negative one:

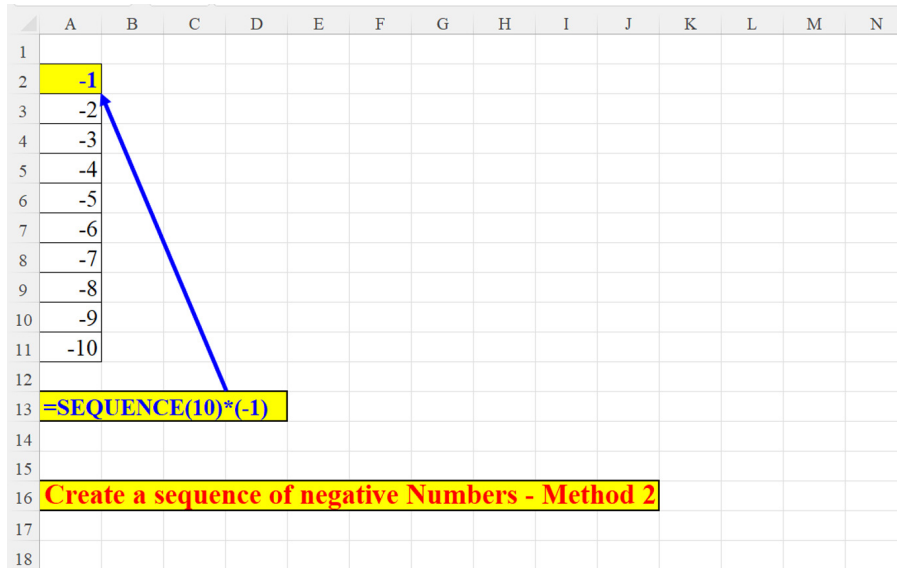


Figure 3.7: Negative SEQUENCE – Method 2

Method 3

In the following figure we create a horizontal array, and then transpose it vertically:

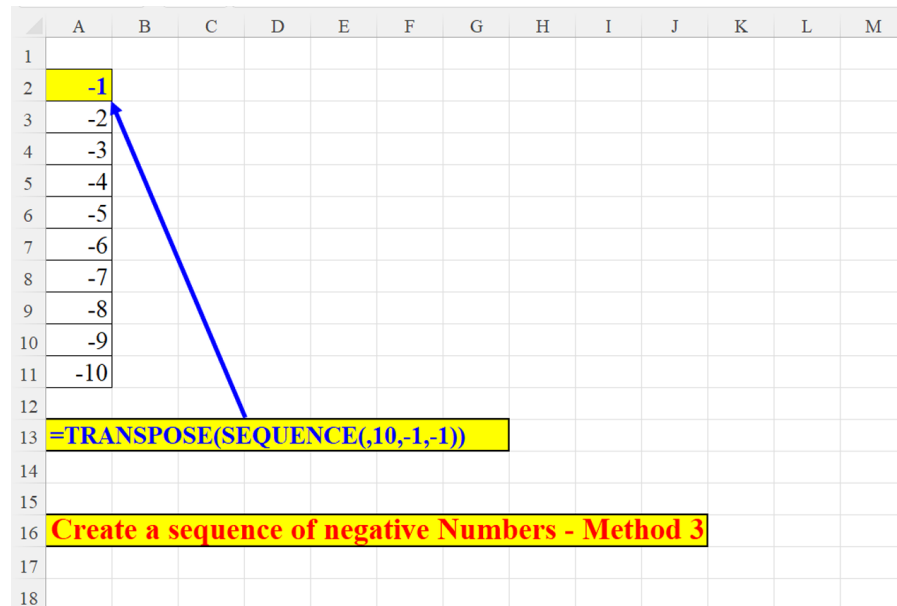


Figure 3.8: Negative SEQUENCE – Method 3

Method 4

This picture is similar to *Figure 3.3*. The only difference is that we multiply the result by (-1) for a negative array:

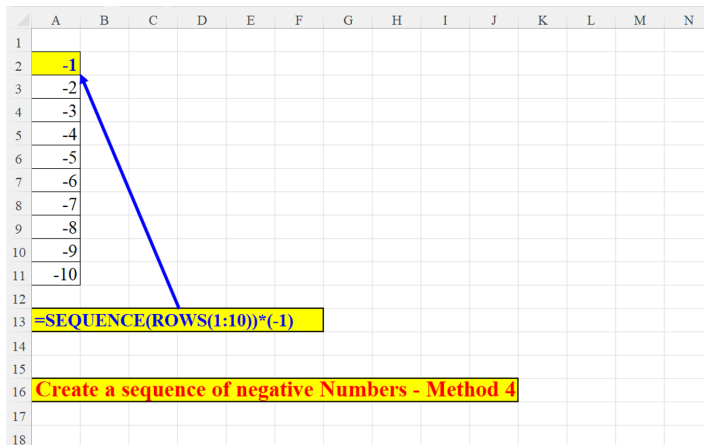


Figure 3.9: Negative SEQUENCE – Method 4

Method 5

The minus sign (-) following the equals sign (=) which starts every formula in Excel, not only causes the array to be negative: it also makes the (=) superfluous. We could have written the formula without it as shown in the following screenshot:

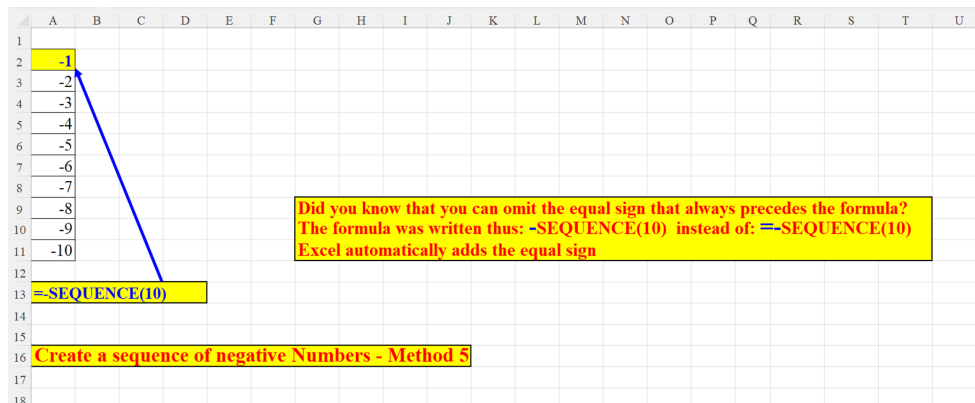


Figure 3.10: Negative SEQUENCE – Method 5

Descending SEQUENCE – Two methods

Using a parameter, we demonstrate two procedures to initiate an array of descending numbers. The parameter specifies the size of the array as well as the number with

which to begin the array. The second procedure combines the **SORT** function with the **SEQUENCE**: we need to sort the array in descending order such that its elements are arranged from largest to smallest.

Method 1

The parameter (in cell **G1**) tells the **SEQUENCE** function both the size of the array and its starting number. Since the last argument is (-1) this is a descending sequence:

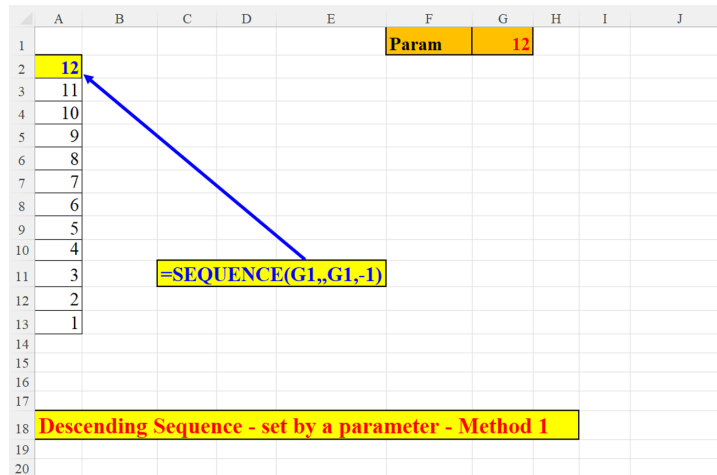


Figure 3.11: Descending SEQUENCE – Method 1

Method 2

Wrapping the **SORT** function around the **SEQUENCE** function: **SEQUENCE** returns an ascending 12-member array which is then converted to a descending array by the **SORT** function:

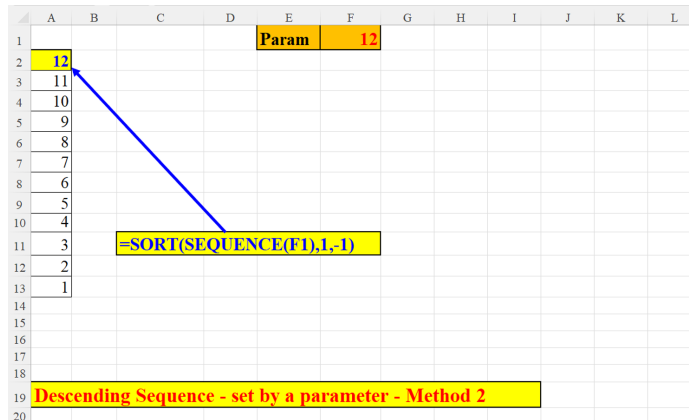


Figure 3.12: Descending SEQUENCE – Method 2

Duplicate cell horizontally

The value in cell A2 is duplicated 10 times (the duplication factor is defined as a parameter in cell M2). The value in A2 is pointed by the **INDIRECT** of the value defined in cell L2. The **INDIRECT** function *translates* the text (in cell L2) into a valid Excel reference. The **TEXT** part of the formula generates 10 empty horizontal cells which are concatenated to the value in A2 to create a textual array. Multiplying this array by 1 converts the text array into a numeric one as shown in the following figure:

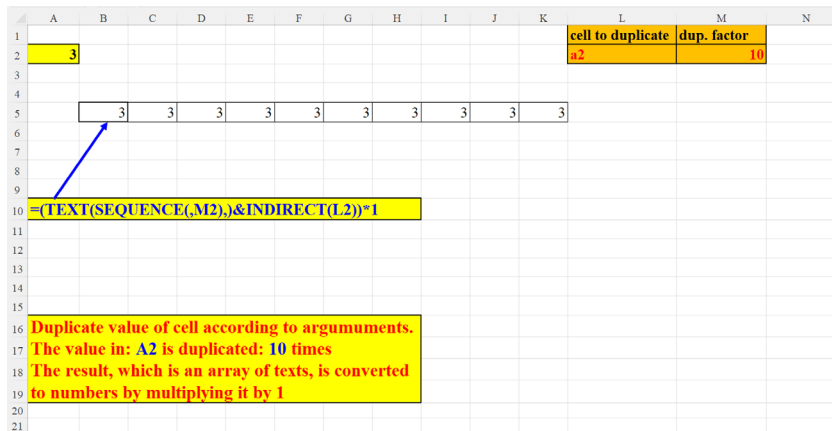


Figure 3.13: Duplicate Cell Horizontally

Duplicate cell vertically

The techniques used here are similar to the ones used in the previous section, only the **SEQUENCE** here returns a vertical array and not a horizontal one. The double-minus (--) at the beginning of the formula transforms the textual array into a numeric array as shown in the following figure:

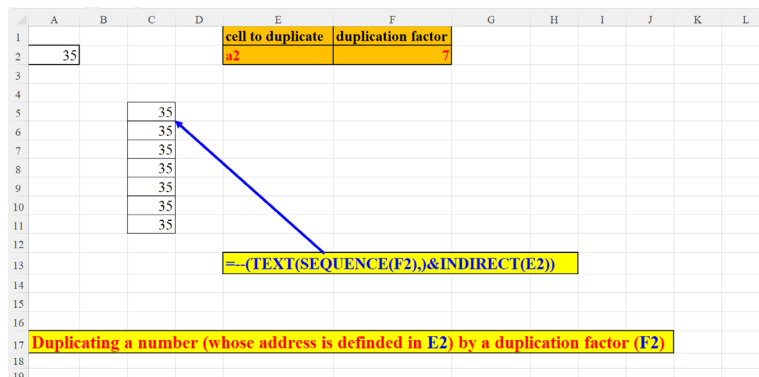


Figure 3.14: Duplicate Cell vertically

Duplicate numbers

The Duplication factor in cell O2 determines the number of occurrences for each number. Since the last argument of the **SEQUENCE** (“increment”) is a fraction, the **INT** function turns the first 6 items into the number: 1, the next 6 items into: 2, and so on as shown in the following figure:

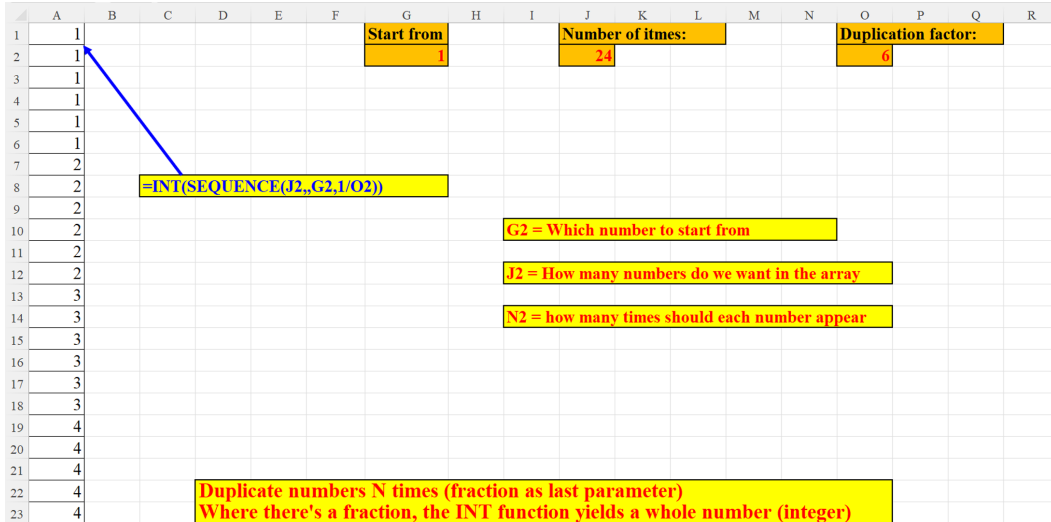


Figure 3.15: Duplicate numbers

Creating a vertical SEQUENCE of numbers – Two methods

The following two pictures demonstrate how to generate a vertical **SEQUENCE** of numbers using an old method **ROW()** and our acquaintance **SEQUENCE**. The advantage of **SEQUENCE** over the old method is not so obvious in the first picture, though it is quite clear in the second one, with the parameter. The old method becomes much less intuitive when using a parameter (always better than *hard-coded* values within the formula). It is noteworthy to mention that the **ROW** function will not create the integer **SEQUENCE** 1-6 if moved to a different starting row, since **ROW()** always points to the row number in which it is defined:

Method 1: Creating a sequence of numbers **without** a parameter

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1			1											
2			2											
3			3											
4			4	=ROW(1:6)										
5			5											
6			6	Old Method										
7														
8														
9														
10			1											
11			2											
12			3											
13			4	=SEQUENCE(6)										
14			5											
15			6	New Method										
16														
17														
18														
19				2 methods to create a vertical sequence of numbers										
20				Old vs. New - without a parameter										
21														
22														
23														
24														

Figure 3.16: Two methods to create a vertical SEQUENCE of numbers (*without* a parameter)

Method 2: Creating a sequence of numbers **with** a parameter

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1			1								n	6		
2			2											
3			3											
4			4	=ROW(INDIRECT("1:"&L1&""))										
5			5											
6			6	Old Method										
7														
8														
9														
10			1											
11			2											
12			3											
13			4	=SEQUENCE(L1)										
14			5											
15			6	New Method										
16														
17														
18														
19				2 methods to create a vertical sequence of numbers										
20				Old vs. New - with a parameter										
21														
22														
23														
24														

Figure 3.17: Two methods to create a vertical SEQUENCE of numbers (*with* a parameter)

Find missing numbers in a list

Suppose you have an unordered, incomplete list of numbers, and you want to know which numbers are missing from the list. The new **Dynamic Array Function (DAF)**, **TOCOL**, (explained in *Chapter 1, A short introduction to Dynamic Array Functions in Excel 365*), together with **SEQUENCE**, generates the list of missing numbers. The formula does not need to know in advance what is the largest number in the original list as shown in the following screenshot:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	
1	List		Missing Numbers																
2		13		3															
3		2		4															
4		1		6															
5		5		7															
6		15		10															
7		8		11															
8		9		16															
9				17															
10		14		18															
11		12																	
12		19																	
13				=LET(a,SEQUENCE(MAX(A:A)),TOCOL(IF(NOT(COUNTIF(A2:A12,a)),a,NA),2,1))															
14				suppose you have an unordered, incomplete list of numbers, and you want to know which numbers are missing															
15				The formula in the picture uses the TOCOL function in unison with the virtual array, created by the SEQUENCE function.															
16				The formula ignores empty cells, too. :-)															
17																			
18																			
19																			

Figure 3.18: Find the missing numbers in a list

Reverse a Number

The following example displays a simple procedure to reverse a number. The last argument of the **SEQUENCE** function is negative: (-1) which tells Excel to create a reverse (decending) **SEQUENCE**, and then the **MID** function splits the *reversed SEQUENCE* into N consecutive horizontal cells (where N is the length of the original number).

Even though all the numbers are left-aligned (as if they were text), they are numbers, nevertheless:

	A	B	C	D	E	F	G	H	I	J	K	L
1												
2		987645321										
3												
4			1	2	3	5	4	6	7	8	9	
5												
6												
7												
8												
9												
10			=MID(A2,SEQUENCE(LEN(A2),LEN(A2),-1),1)									
11												
12												
13												
14												
15			Reverse a number									
16												
17												
18												
19												

Figure 3.19: Reverse a number

Reverse a horizontal ascending array

In cells A2:J2 we have an array of 10 consecutive numbers. We do not know in advance whether it is an ascending (first item=1) or descending (first item>1) array.

Since we skipped the first argument (there are no rows in the array), the array's size is decided by the second argument (COUNT(2:2)) which returns the number of non-empty cells in row number 2. The array's starting number is established by the third argument (identical to the second) and the last argument determines whether the resulting array is going to be ascending or descending as shown in the following screenshot. Another very interesting feature of this formula is the fact that if we move the array in row 2, the formula "adapts itself" automatically to the array's new starting location:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1		Before													
2	1	2	3	4	5	6	7	8	9	10					
3															
4															
5		=SEQUENCE(,10)													
6															
7															
8															
9		After													
10	10	9	8	7	6	5	4	3	2	1					
11															
12															
13		=SEQUENCE(COUNT(2:2),INDEX(2:2,COUNT(2:2)),IF(A2=1,-1,1))													
14															
15															
16															
17		Reverse a horizontal ascending array by its last member.													
18		We don't need to know in advance the array's size													
19															

Figure 3.20: Reverse horizontal ascending array

Reverse a horizontal descending array

This example utilizes the same technique as in the previous section. The formula reverses a descending array into an ascending one as shown in the following screenshot:

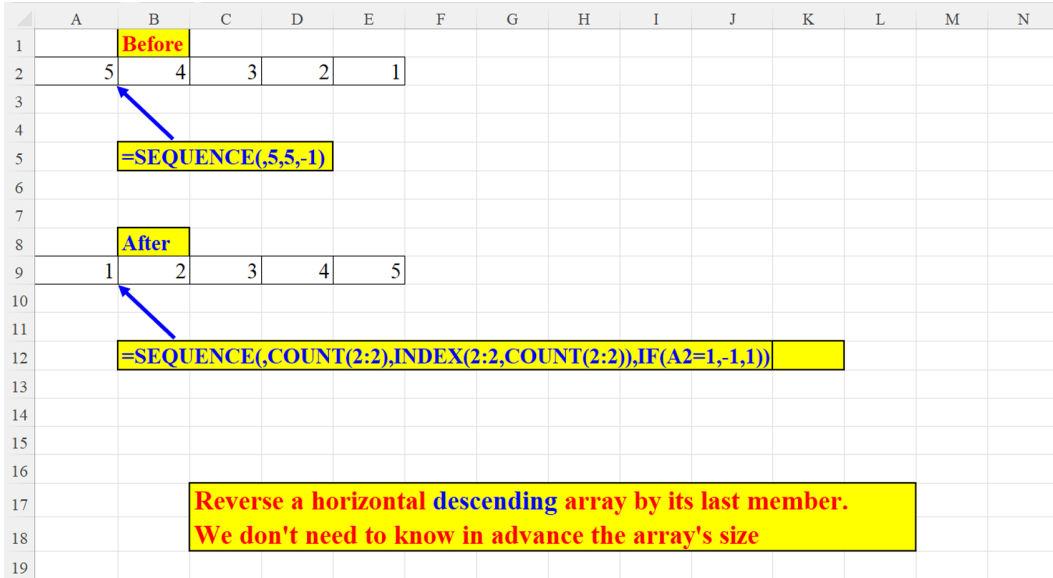


Figure 3.21: Reverse horizontal descending array

SEQUENCE of odd and even numbers

The only difference between the two formulae is the third argument: 1 for odd numbers, and 2 for even numbers as shown in the following screenshot:

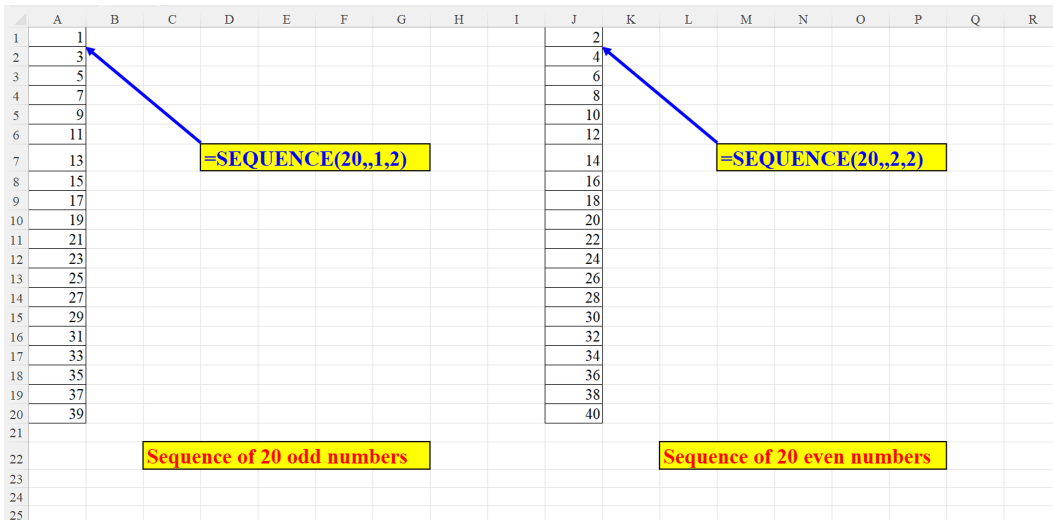


Figure 3.22: SEQUENCE of odd and even numbers

Sum all digits in a cell which has only digits

The inner part of the formula *splits* the number into separate digits, which are then SUMmed by the outer function as shown in the following screenshot:

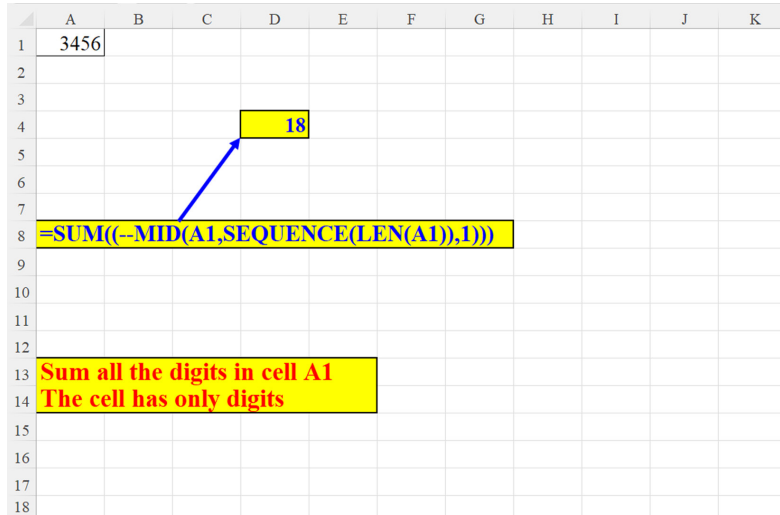


Figure 3.23: SUM all digits in a cell containing only digits

Sum all digits in a cell which has digits and text

The inner part of the formula *splits* the number into separate digits and ignores non-digit characters, which are then SUMmed by the outer function as shown in the following screenshot:

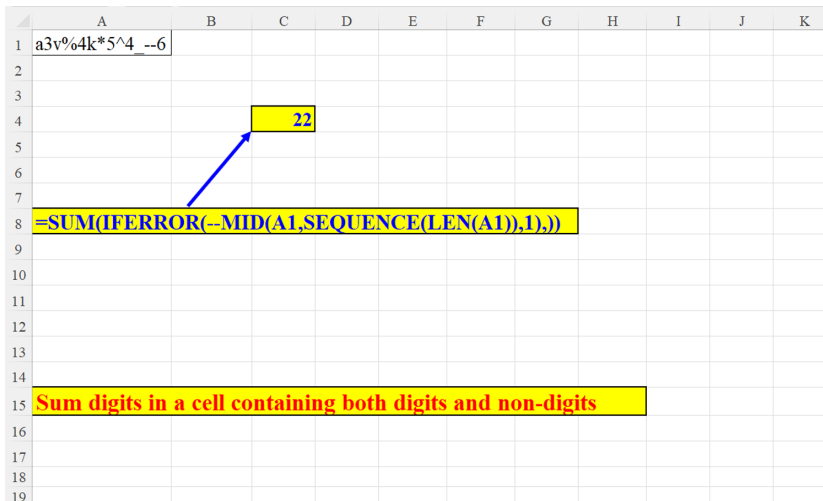


Figure 3.24: SUM all digits in a cell containing digits and text

Sum every Nth row

The formula first locates all the numbers in the dataset whose location is divisible by 3 (3rd, 6th, 9th) [numbers divisible by 3 have no remainder]. These numbers are then SUMmed up by the **SUMPRODUCT** function as shown in the following screenshot:

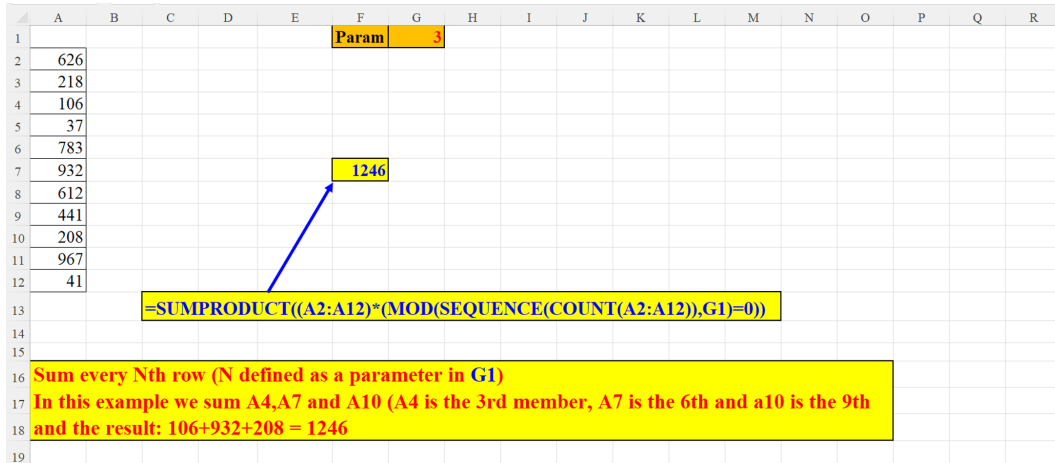


Figure 3.25: SUM every Nth row

Sum the largest N numbers

The **LARGE** function returns the N(3) largest numbers, which are then SUMmed by the **SUMPRODUCT** function as shown in the following screenshot:

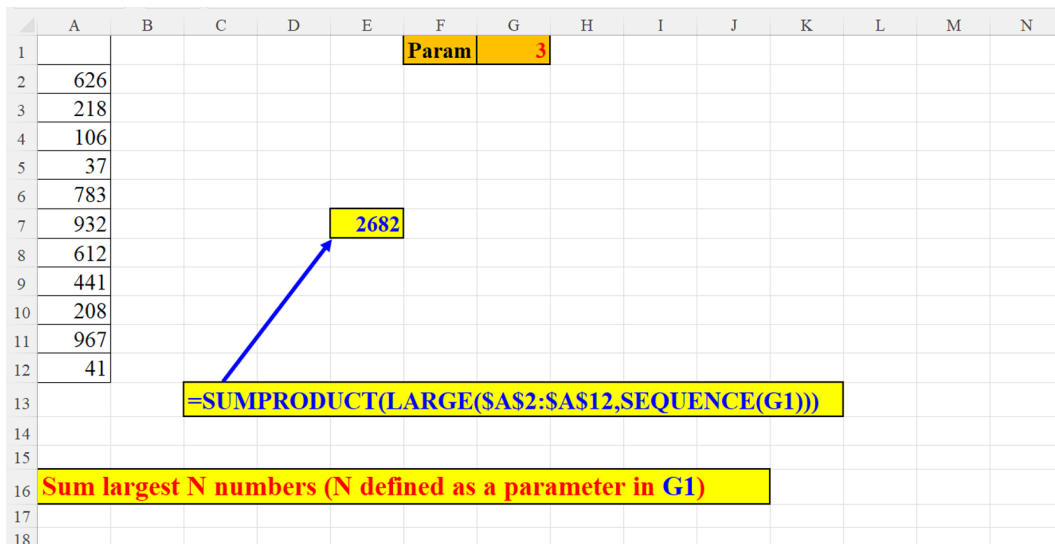


Figure 3.26: SUM largest N numbers

Sum the smallest N numbers

This example is similar to the previous one, but here the **SMALL** function returns the three smallest numbers as shown in the following screenshot:

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	List					Param	3						
2	626												
3	218												
4	106							184					
5	37												
6	783												
7	932												
8	612												
9	441												
10	208					=SUMPRODUCT(SMALL(SAS2:SAS12,SEQUENCE(G1)))							
11	967												
12	41												
13													
14						Sum smallest N numbers (N defined as a parameter in G1)							
15													
16													
17													
18													
19													

Figure 3.27: SUM smallest N numbers

Two tricks with SEQUENCE (ROW())

This is really interesting: the row on which you type this formula: **SEQUENCE(ROW())**, is the number of members in the array. As can be seen in *Figure 3.28*, if you type the formula on row no. 5, the formula creates the **SEQUENCE** : 1,2,3,4,5. If you type the formula on row no. 8, the array: 1,2,...8 is generated.

Create a SEQUENCE of n Rows starting from Row(n)

If we type the formula: **=SEQUENCE(ROW())** in any row, the formula generates a sequence of n integers, where n is equal to the row number. You do not need to specify the array's size.

For example, if we type the formula in row 5, we get a sequence of the first 5 natural numbers. If the formula is written in row 8, it yields a sequence of the first 8 natural numbers.

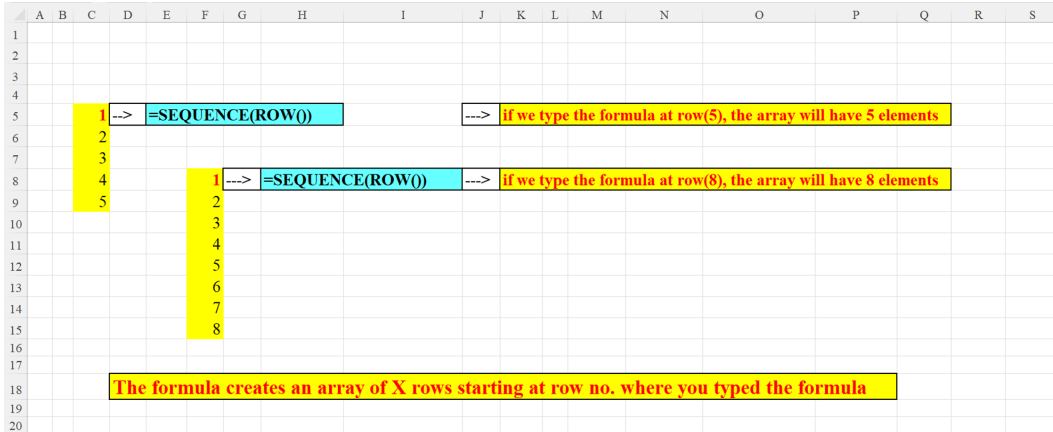


Figure 3.28: Create a SEQUENCE of n Rows starting from Row(n)

SUM a virtual array created by SEQUENCE (ROW())

Let us take the example in the previous subsection one step further. Now we want to SUM the array that the **SEQUENCE(ROW())** formula generated when typed on certain row. The result will be the sum of the array without a “physical” array! The array that is SUMmed is virtual: It exists only in Excel’s memory, it does not “reveal” itself on the spreadsheet!

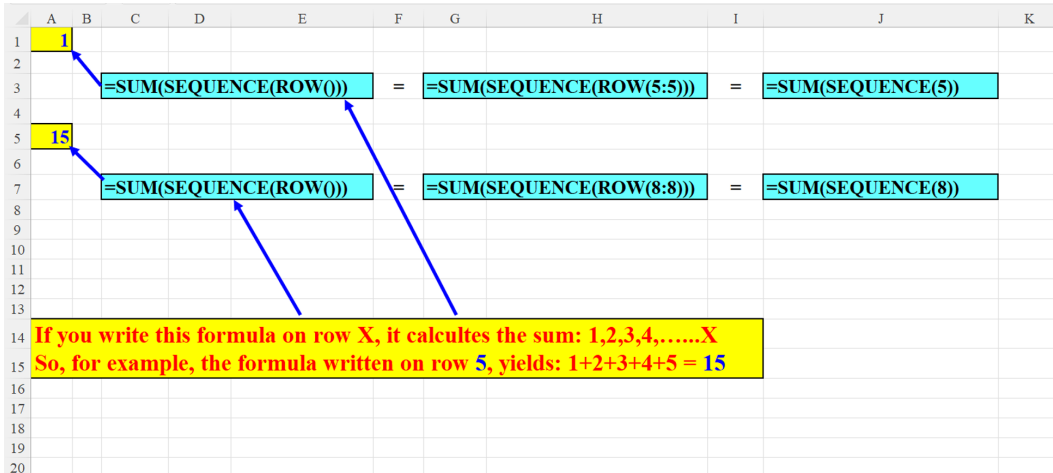


Figure 3.29: SUM a Virtual Array created by SEQUENCE (ROW())

SUM SEQUENCE (virtual array)

Another interesting demonstration of the unique feature of **SEQUENCE** manifested in the previous subsection: A *real* array is engendered by the **SEQUENCE** function

however it becomes *virtual* if we wrap it with the **SUM** function as shown in the following screenshot:

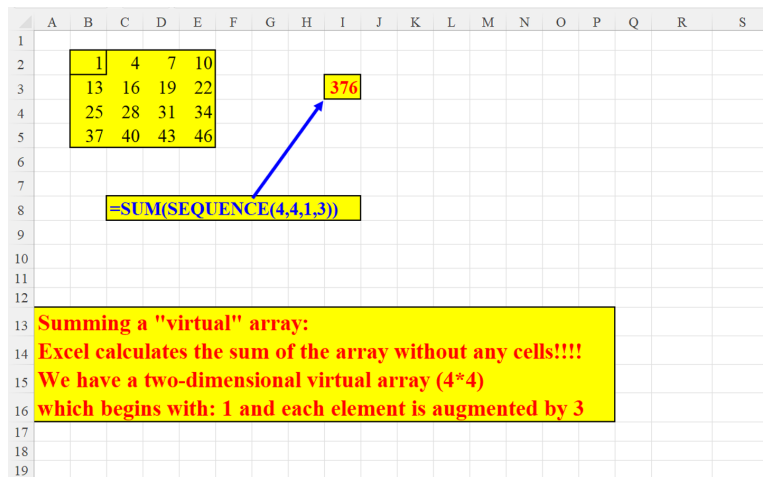


Figure 3.30: SUM&SEQUENCE (virtual array)

Alternate 1s and 0s

In cell B2 we have five zeroes (00000) preceded by five ones (11111). We engender an array of 1's and 0's, where the **TEXTJOIN** function joins the first five digits of cell B2 with the last five digits as shown in the following screenshot:

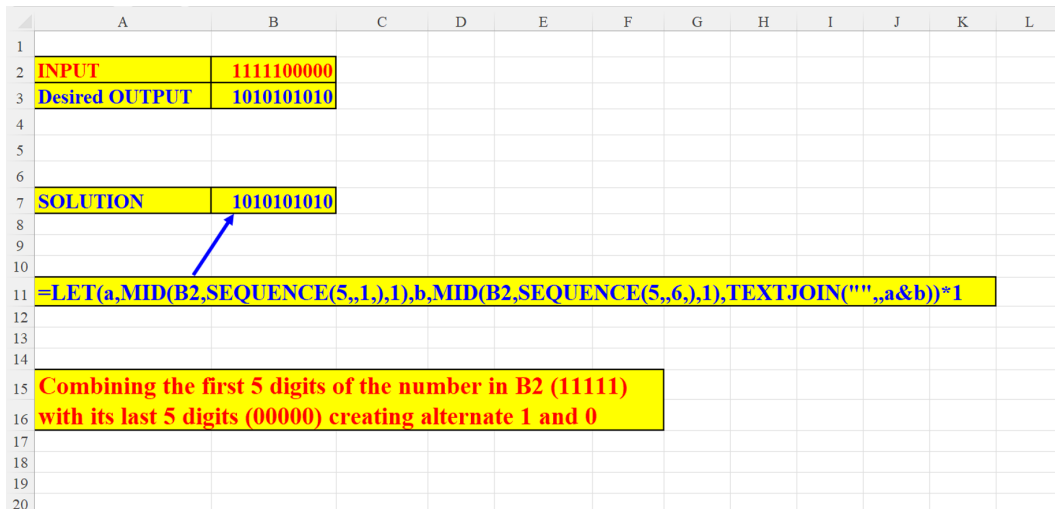


Figure 3.31: Alternate 1s and 0s

Dynamic SEQUENCE

Three parameters shape this array: the first (in cell G1) sets the number of items, the second (in cell J1) sets the starting number and the third (in cell M1) assigns the increment as shown in the following screenshot:

	B	C	D	E	F	G	H	I	J	K	L	M	N
1		15			How many numbers:	12		starting from:	15		increment:	1	
2		16											
3		17											
4		18											
5		19											
6		20											
7		21											
8		22											
9		23		=SEQUENCE(G1,J1,M1)									
10		24											
11		25											
12		26											
13													
14													
15													
16													

Figure 3.32: Dynamic SEQUENCE

Two methods to extract a number from the string's end

This section will demonstrate two methods of extracting a number that appears at the end of a string, and preceded by non-digits.

Method 1

The formula finds the cell's first integer location (in our case, the 4th character) and then extracts all the digits from that location to the end of the string. The assumption here is that all the cell's digits appear at the cell's end:

	A	B	C	D	E	F	G	H	I	J	K	L	M
1		bbb124											
2													
3													
4						124							
5													
6													
7													
8					=MID(SBS2,MIN(FIND(SEQUENCE(1,10,0,1),CONCAT(SBS2,SEQUENCE(10,0))))),LEN(SBS2))								
9													
10													
11													
12													
13													
14													
15													
16													
17													
18													

Figure 3.33: Extract a number from the string's end – Method 1

Method 2

This screenshot exemplifies the use of two new DAF function, **TEXTAFTER** and **VSTACK**, (explained in *Chapter 1, A short introduction to Dynamic Array Functions in Excel 365*). The **VSTACK** function *piles up* all non-numeric characters (preceding 0, CHAR(48) and succeeding 9, CHAR(57)). Then the **TEXTAFTER** function returns the text after the last occurrence of the non-digits (the third argument to the function is: -1):

	A	B	C	D	E	F	G	H	I	J	K
1											
2		bbb124									
3											
4						124					
5											
6											
7											
8		=TEXTAFTER(B2,VSTACK(CHAR(SEQUENCE(47)),CHAR(SEQUENCE(198,,58))),-1)									
9											
10											
11											
12											
13											
14											
15											
16		Extract number from end of string - Method 2									
17											
18											
19											

Figure 3.34: Extract a number from the string's end – Method 2

Two methods to extract a number from the string's start

This section will demonstrate two methods of extracting a number that appears at the beginning of a string before any non-numeric characters.

Method 1

The **MATCH** function retrieves the location of the first non-digit character (in our case: 4). Then the **LEFT** function extracts $4-1 = 3$ characters from the string's start as shown in the following screenshot:

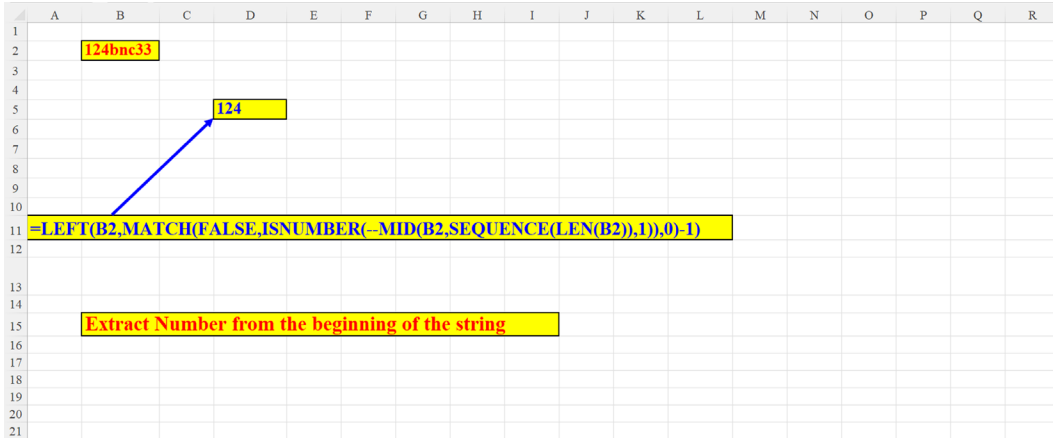


Figure 3.35: Extract a number from the string's start – Method 1

Method 2

The technique used here is similar to the one used in the previous section, method 2. But instead of using the **TEXTAFTER** function, we use the **TEXTBEFORE** (this function is explained in: *Chapter 1, A short introduction to Dynamic Array Functions in Excel 365*). The third argument to the function is omitted, since we use the default: instance number=1 as shown in the following screenshot:

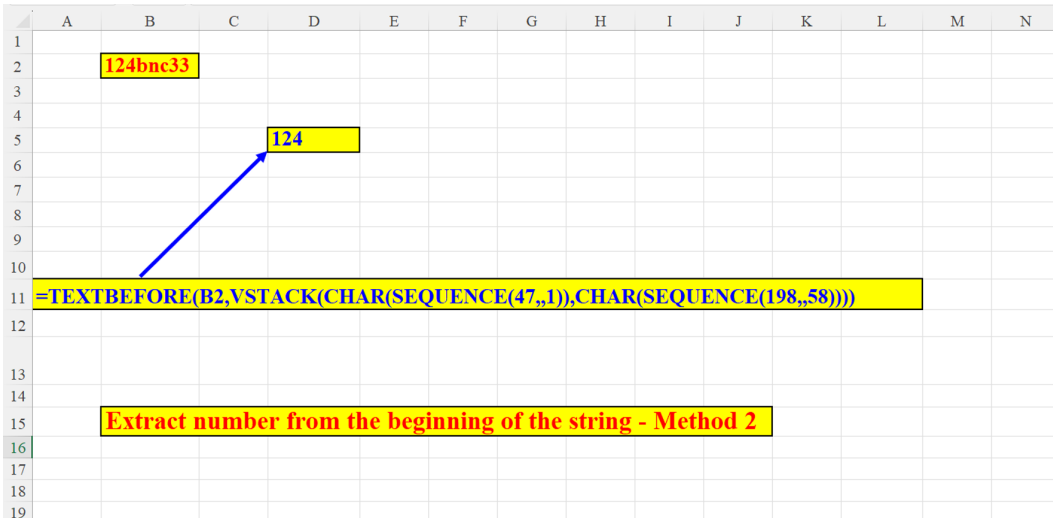


Figure 3.36: Extract a number from the string's start – Method 2

Find N largest numbers (Ascending)

The **LARGE** function extracts the 3 (parameter in cell G1) largest numbers from the dataset in Column B. Since the last parameter of **SEQUENCE** is -1, the result shown in ascending order (from small to large) is illustrated in the following screenshot:

	A	B	C	D	E	F	G	H	I	J	K	L
1						numbers to extract	3					
2												
3		Numbers array		largest numbers								
4		12		509								
5		15		567								
6		128		1044								
7		8										
8		78										
9		567		=LARGE(B4:B16,SEQUENCE(G1,G1,-1))								
10		89										
11		98										
12		419										
13		509										
14		233										
15		65										
16		1044										
17												
18				Find n (parameter) largest numbers in a range. The result: in ascending order								
19												
20												

Figure 3.37: Find N largest Numbers(Asc.)

Find N largest numbers (Descending)

The **LARGE** function extracts the 3 (parameter in cell G1) largest numbers from the dataset in Column B. Since the last parameter of **SEQUENCE** is 1, the result is shown in descending order (from large to small) as shown in the following screenshot:

	B	C	D	E	F	G	H	I
1					numbers to extract	3		
2								
3	Numbers array		largest numbers					
4	12		1044					
5	15		567					
6	128		509					
7	8							
8	78							
9	567		=LARGE(B4:B16,SEQUENCE(G1))					
10	89							
11	98							
12	419							
13	509							
14	233							
15	65							
16	1044							
17								
18			Find n (parameter) largest numbers in a range. The result: in descending order					
19								
20								

Figure 3.38: Find N largest Numbers(Desc.)

How many columns in a sheet

This formula is quite complicated and deserves an explanation:

First, the MID function with the SEQUENCE function yields the 3-character array, "X";"F";"D" since the string length in A2 is 3, and 3-LEN(A2)= 0. When concatenated with A2, we get the string in cell A2. Then, the **CODE** function *translates* each letter of the array into its ASCII code: X=88, F=70, D=68. Further, we subtract 64 from each of these values since the ASCII codes for uppercase English letters starts with A=65. So, the result is this numeric array: {24; 6; 4}.

Finally, we multiply each number of the array by: 26^2 , 26^1 and 26^0 and sum the result:

$$24*26^2 + 6*26^1 + 4*26^0 = 16224 + 156 + 4 = 16384.$$

	A	B	C	D	E	F	G	H	I	J	K	L
1	Last column's name	Last column's number										
2	XFD	16384										
3												
4												
5												
6												
7												
8												
9	=SUM((CODE(MID(REPT("@",3-LEN(A2))&A2,SEQUENCE(3,1))-64)*SORT(26^SEQUENCE(3,0,1),-1))											
10												
11												
12												
13												
14												
15	How many columns are there in an Excel Worksheet											
16												
17												
18												
19												
20												
21												
22												

Figure 3.39: How many columns in a sheet

How many digits

The **MID** function in unison with the **SEQUENCE** function extracts only digits from the string in cell A2. Then, these digits are concatenated into a single number, and

finally the LEN function returns this number's length as shown in the following screenshot:

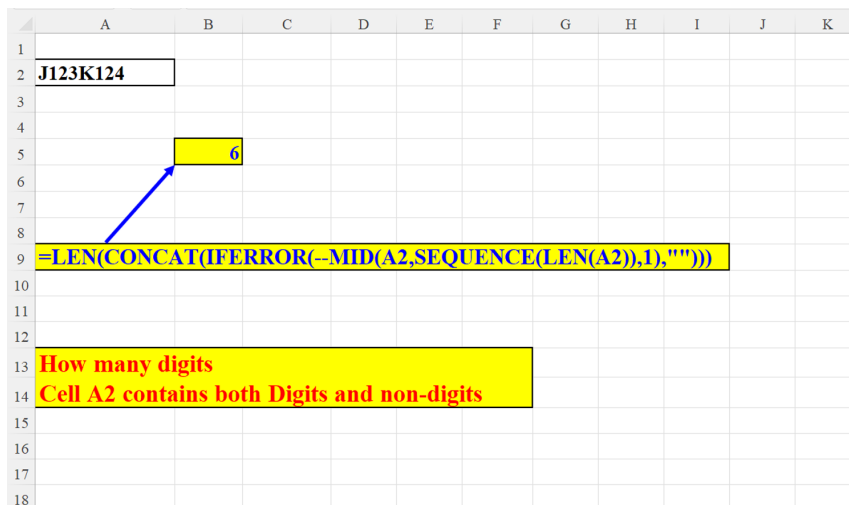


Figure 3.40: How many digits

Reverse numbers horizontally by a parameter

The formula suggested here is a simpler procedure to reverse a number (compare subsections *Descending Sequence - method 1*, Figure 3.11) and *Descending Sequence - method 2*, Figure 3.12). A parameter (in cell K1) sets the array's size, both the original one (in A2) and the reversed one (in A10) as shown in the following screenshot:

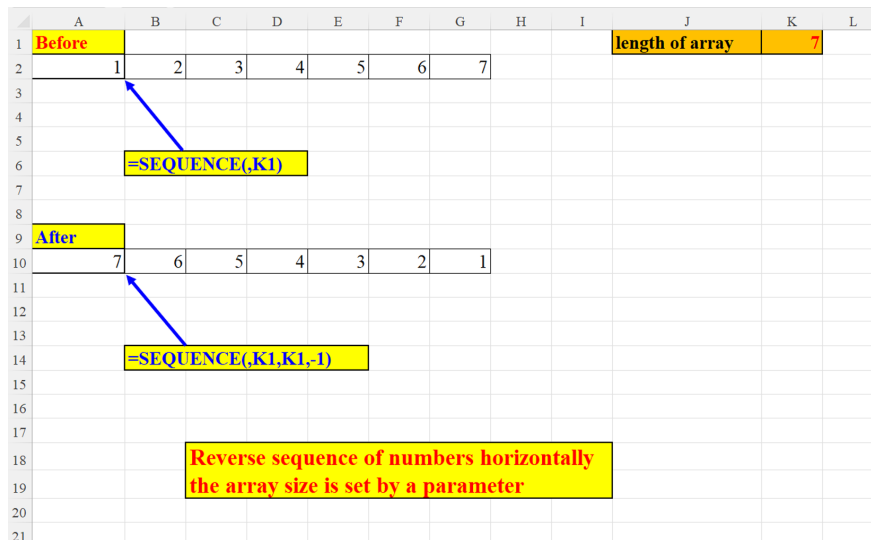


Figure 3.41: Reverse numbers horizontally by a parameter

Reverse order of a SEQUENCE of numbers

A dynamic table in column A (*Tnum*) is a table of natural numbers (integers greater than 0), ordered consecutively in ascending order. The array contrived in column C reverses the order of the original table. Since we are dealing with a dynamic table, adding new numbers to the table will automatically update the array in column C as shown in the following screenshot:

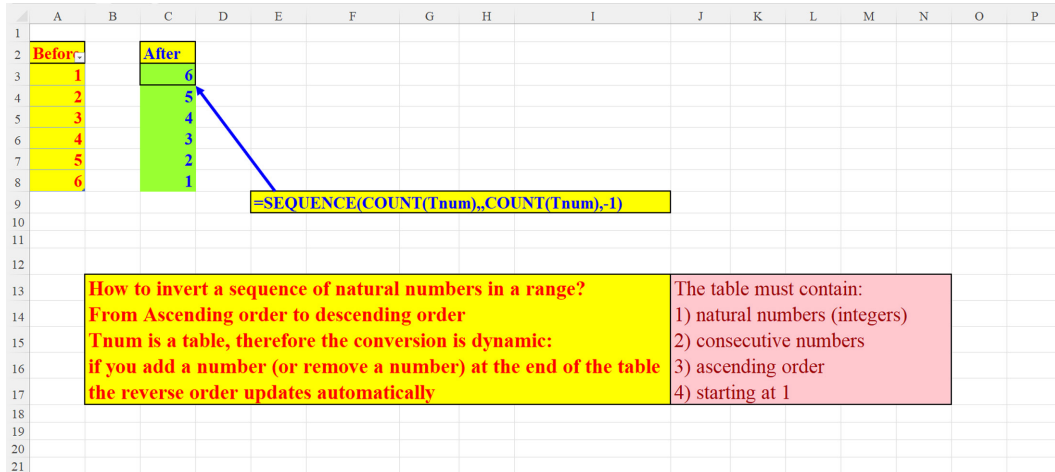


Figure 3.42: Reverse order of a SEQUENCE of numbers

Subject with the highest score

This is basically an **INDEX-MATCH** formula, where the *traditional* **MATCH** is replaced by the **SUMPRODUCT** function: the search for the largest number in the range A2:D13 generates an bi-dimensional (4*12) array in which only one value is **TRUE (1)**. This array is multiplied by the one-dimensional array created by the **SEQUENCE** function (4 elements, since we have 4 columns). The result of this multiplication is the column

where we have the highest score: the 4th column, which is the Sciences subject as shown in the following screenshot:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Math	English	History	Sciences											
2	84	43	76	36											
3	67	95	51	81											
4	92	93	49	99		Sciences									
5	79	36	75	66											
6	85	84	87	64											
7	45	26	80	92											
8	54	10	35	81											
9	58	70	54	88											
10	65	66	97	71											
11	97	12	35	19											
12	91	81	80	39											
13	34	39	10	47											
14															
15															
16															
17															
18															
19															
20															
21															
22															

=INDEX(A1:D1,SUMPRODUCT(--(A2:D13=MAX(A2:D13))*(SEQUENCE(4))))

In which subject did I get the highest score?

Figure 3.43: Subject with the highest score

SEQUENCE based on number of unique values

For each unique item in list 1 (sorted in ascending order, column B) we add an index number (column G). There are seven items in unique List1 (column F), so the array produced by the **SEQUENCE** function has seven index numbers as shown in the following screenshot:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1		List 1				List 1	Index								
2		491				491	1								
3		491				493	2								
4		491				495	3								
5		493				520	4								
6		493				521	5								
7		495				536	6								
8		495				681	7								
9		520													
10		520													
11		520													
12		521													
13		521													
14		536													
15		681													
16		681													
17		681													
18															
19															
20															

=UNIQUE(B2:B17)

=SEQUENCE(COUNT(F:F))

The array generated in column G depends on the number of unique items in column B (an ordered dataset)

Figure 3.44: SEQUENCE based on number of unique values

Dynamic frequency based on dynamic bins

The **FREQUENCY** function in Excel is used to sum vertically the number of items for each bin in the bins array (Column E). The bins array is built dynamically according to three parameters in: H1, J1, L1 which are, respectively, the number of bins (10), the start of the array (50) and the step/increment for item in the array. The **FREQUENCY** function (in Column C) counts all the items for each bin. For example, in the first bin (50) there is only one number (45) which is less than or equal to 50. In the 100th bin we have 4 numbers: 96, 96, 98, 99, and so on.

Please pay attention to the notation of the second argument of the **FREQUENCY** function. The # in the E2# denotes the Spilled Range Operator. E2# is the array created by the **SEQUENCE** function as shown in the following screenshot:

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Numbers		frequency		bins		No. of bins	10	from	50	step	10	
2	98		1		50								
3	64		1		60								
4	96		4		70								
5	139		1		80								
6	130		2		90								
7	128		4		100								
8	96		1		110								
9	81		2		120								
10	104		4		130								
11	64		2		140								
12	134		0										
13	119												
14	45												
15	67												
16	77												
17	114												
18	90												
19	99												
20	129												
21	64												
22	125												
23	54												
24													

=SEQUENCE(H1,,J1,L1)

=FREQUENCY(A:A,E2#)

**Dynamic Frequency (col. C)
The dynamic Bins (col. E) are set by 3 parameters
in: H1, J1 & L1**

Figure 3.45: Dynamic frequency based on dynamic bins

SEQUENCE column

Both formulae (in A1 and in A10) yield the same result: the number 10 is *spilled* all over row 1 and row 10. The size of the upper and lower arrays is the same: all columns of the worksheet as shown in the following figure:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
2																		
3																		
4																		
5																		
6																		
7																		
8																		
9																		
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
11																		
12																		
13																		
14																		
15																		
16																		
17																		
18																		
19																		
20																		
21																		
22																		

Figure 3.46: SEQUENCE column

SEQUENCE and COLUMNS

The result of: **COLUMNS(1:1)** is the number of the worksheet's columns (=16384). When *wrapping* this number with the **SEQUENCE** function, we obtain a vertical array of 16384 rows as shown in the following screenshot:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	1																	
2	2																	
3	3																	
4	4																	
5	5																	
6	6																	
7	7																	
8	8																	
9	9																	
10	10																	
11	11																	
12	12																	
13	13																	
14	14																	
15	15																	
16	16																	
17	17																	
18	18																	
19	19																	
20	20																	
21	21																	
22	22																	

Figure 3.47: SEQUENCE and COLUMNS

Building a chessboard in three steps

This section is going to explain (in three steps) how to build a chessboard in Excel.

Chessboard - Step 1

Step 1: The formula

=IF(ISODD(COLUMN()),BITAND(SEQUENCE(8),1),--NOT(BITAND(SEQUENCE(8),1))) creates a bi-dimensional array (8*8). It is written in Cell A1 and then dragged down to cell A8.

This formula generates alternate 1's and 0's so that next to each cell containing 1, there are only cells containing 0 (left, right, beneath and above). In odd Column numbers (A,C,E,G) the function **BITAND(SEQUENCE(8),1)** produces alternate 1's and 0's starting with 1: {1;0;1;0;1;0;1;0}. For each odd number in the array defined in the function's first argument: **SEQUENCE(8)**, the function generates 1 and for each even number in that array the function begets 0. That is how a **SEQUENCE** of 8 alternate 1's and 0's is created.

However, in even Column numbers (B,D,F,H) the function **NOT(BITAND(SEQUENCE(8),1))** produces the opposite array: {0;1;0;1;0;1;0;1}. Here the process is reversed with the **NOT** operator: for each 1 on an odd column number, a 0 is generated whereas for each 0 on such column number, a 1 is engendered.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	1	0	1	0	1	0	1	0						
2	0	1	0	1	0	1	0	1						
3	1	0	1	0	1	0	1	0						
4	0	1	0	1	0	1	0	1						
5	1	0	1	0	1	0	1	0						
6	0	1	0	1	0	1	0	1						
7	1	0	1	0	1	0	1	0						
8	0	1	0	1	0	1	0	1						
9														
10														
11	=IF(ISODD(COLUMN()),BITAND(SEQUENCE(8),1),--NOT(BITAND(SEQUENCE(8),1)))													
12														
13														
14														
15														
16	Phase 1 - Creating a two-dimensional array (8*8) of alternate 1's and 0's by dragging the formula in A1 all the way to A8													
17														
18														
19														
20														

Figure 3.48: Chessboard - Step 1

Chessboard - Step 2

Step 2: We resize the array originated in *Step 1* to create 64 square cells: the height of each cell is equal to its width as shown in the following screenshot:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	1	0	1	0	1	0	1	0								
2	0	1	0	1	0	1	0	1								
3	1	0	1	0	1	0	1	0								
4	0	1	0	1	0	1	0	1								
5	1	0	1	0	1	0	1	0								
6	0	1	0	1	0	1	0	1								
7	1	0	1	0	1	0	1	0								
8	0	1	0	1	0	1	0	1								
9																
10																
11																
12																
13																
14																
15																
16	Phase 2 - select cells A1:H8,															
17	Set their Height to 18.75,															
18	Set their Width to 2.22															
19																
20																
21																
22																
23																
24																

Figure 3.49: Chessboard - step 2

Chessboard - Step 3

Step 3: First, we employ the Conditional Formatting feature in order to *paint* black each cell containing 1. Then, to *clean* the contents of all the cells, we use the following trick with format cells: in custom number formatting we define the type: ;;; (three semicolons). This will hide the contents of all the chessboard's cells.

Now we have a chessboard in cells A1:H8 as shown in the following screenshot:

Phase 3:

- 1) select cells A1:H8
- 2) In Conditional Formatting define the formula: =A1=1
- 3) In Number Formatting define ::: (3 semicolons)

Figure 3.50: Chessboard - step 3

Creating N-digit number with the same digit repeated N times

The **SEQUENCE** function creates an array of 5 times the digit 5. The **CONCAT** function then combines all these 5 digits together and creates a 5-digit number as shown in the following screenshot:

Creating an N-digit number with the same digit

Figure 3.51: Creating N-digit number with the same digit repeated N times

Conclusion

This chapter exemplified and demonstrated many use-cases where the **SEQUENCE** is applied in numeric operations or in generating numbers. One very interesting feature that we encountered in this chapter is the fact that this function can generate results even with a *virtual* array.

The next chapter will discuss the employment of **SEQUENCE** with arrays.

Points to remember

- SEQUENCE is very useful in creating ascending or descending arrays of numbers, starting from any number with any increment (either positive or negative). If the increment is 0 (as, for example, in sub-section *Extract a number from the string's start – Method 2*) the same digit/number is generated in the array.
- SEQUENCE can easily be applied to extract numbers or digits from a string.
- SEQUENCE combined with SUMPRODUCT, helps summing certain *chunks* of data, such as part of a list of numbers (for example: *SUM every Nth row, SUM the largest N numbers, SUM the smallest N numbers*)
- SEQUENCE combined with SUMPRODUCT, can imitate the MATCH function (for example: section *SUM SEQUENCE (virtual array)*)
- SEQUENCE can generate an array of alternate 0's and 1's (for example: section *Alternate 1s and 0s*)

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



CHAPTER 4

SEQUENCE in Arrays

Introduction

As we already know, the SEQUENCE is one of the new Dynamic Array Functions of Excel 365. So, what would be more appropriate than demonstrating its seamless conjunction with arrays? An Excel array is a set of similar and related items arranged contiguously. Each item in the array can be accessed directly, by an index. That is why the SEQUENCE function is, so to speak, built-in to the very concept of arrays. Not only can it create an array, but it can also easily create a subset of an array which can hold one or more items.

In this chapter, we will display and explain the advantages of using the SEQUENCE function when arrays are the focus. The SEQUENCE function helps us generate arrays: unidimensional as well as bidimensional, dynamic arrays, flipped arrays (upside-down or “mirror”), etc.

Structure

In this chapter, we will discuss the following topics:

- Examples of SEQUENCE with arrays
 - Creating an array of identical numbers - Two methods
 - Creating an array of ascending numbers - Three methods

- From one cell to a vertical array
- How many active months
- Build a dynamic array - horizontal or vertical
- Flip columns horizontally
- Flip vertical array (with and without SEQUENCE)
- Flip part of vertical array using a parameter
- Create a two-dimensional array using 4 parameters
- Flexible LARGE
- MMULT with static ranges and with dynamic Arrays
- Four useful tricks with VLOOKUP
- From vertical to horizontal – Two Methods
- Four two-dimensional arrays generated by two parameters
- Select columns by parameters
- Transpose a vertical array without knowing its size beforehand
- Fetching multiple results for a search value
- Vertical to horizontal without TRANSPOSE

Objectives

Upon concluding this chapter, you should be able to implement the **SEQUENCE** function when operating on arrays of data, either numeric or non-numeric. **SEQUENCE** is a good companion to many other useful functions. Applying it to those functions will enhance your versatility in Excel.

Examples of SEQUENCE with arrays

As the chapter's name implies, we are going to exhibit the **SEQUENCE** function when employing array operations.

We will manifest some of the advantages of using **SEQUENCE** with arrays. For example, creating an array of identical numbers, creating an array of descending numbers, flipping/creating vertical arrays from horizontal arrays and vice versa, several methods for fetching specific data items from a **VLOOKUP** table, selecting specific columns from a data set, fetching multiple results for a search value and more.

Creating an array of identical numbers - Two methods

We demonstrate here two methods to create an array of identical numbers. It is worth noting that using a parameter is always a preferred procedure. If you want to change the array's size, you will not have to *tamper with* the formula itself. Another point to be stressed is that in the second formula, we can omit the third argument (starting number), but we should explicitly indicate the fourth argument (step/increment) as 0, otherwise, the formula will use the default value (1) and the array created will not consist of identical members as shown in the following screenshot:

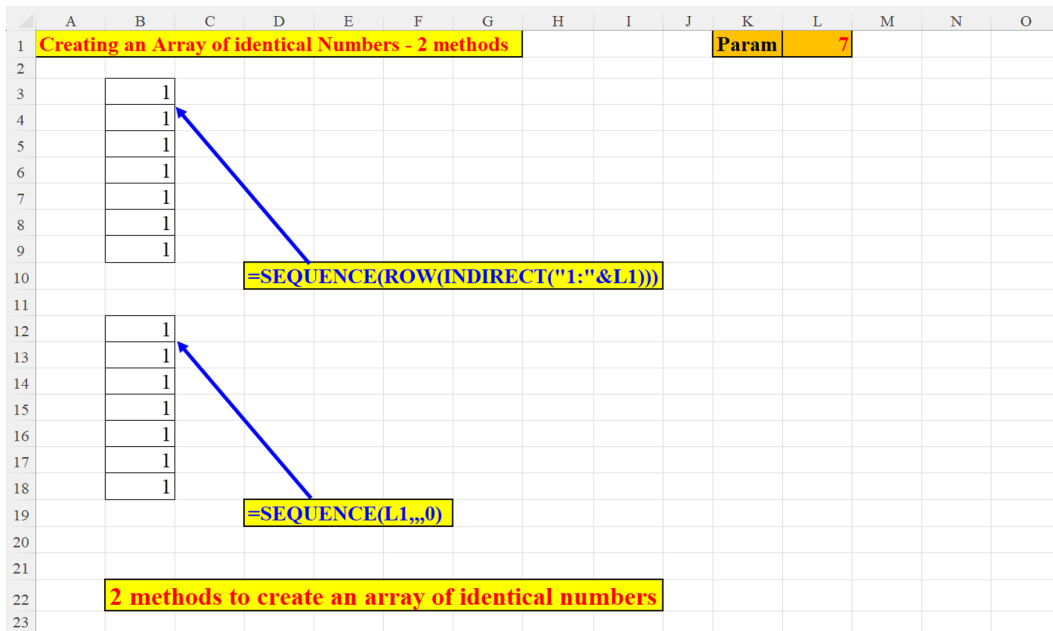


Figure 4.1: Creating an Array of identical Numbers - Two Methods

Creating an array of ascending numbers – three methods

As we all know, there is (almost) always more than one way to solve a problem in Excel. Figure 4.2 exhibits three solutions for creating an array of ascending values:

- First, we have the older method before the Dynamic Array Functions. Here, we use the **ROW** function combined with **INDIRECT**. Although **INDIRECT** is a *volatile* function (which means that it recalculates every time a change is made to the workbook), the impact of a very small number of *volatile* functions is negligible.

- Secondly, we have a mixture of the **ROWS** function with the **INDIRECT**, wrapped with our function. The **ROWS** function returns the number of elements in the array (**ROWS(1:7)**, which is 7). This number (7) is the argument to the **SEQUENCE** function which then generates a one-dimensional array of seven natural numbers, from 1 to 7.
- However, the shortest, most obvious solution is on the bottom left: **SEQUENCE** with only one argument, the parameter in L1 as shown in the following screenshot:

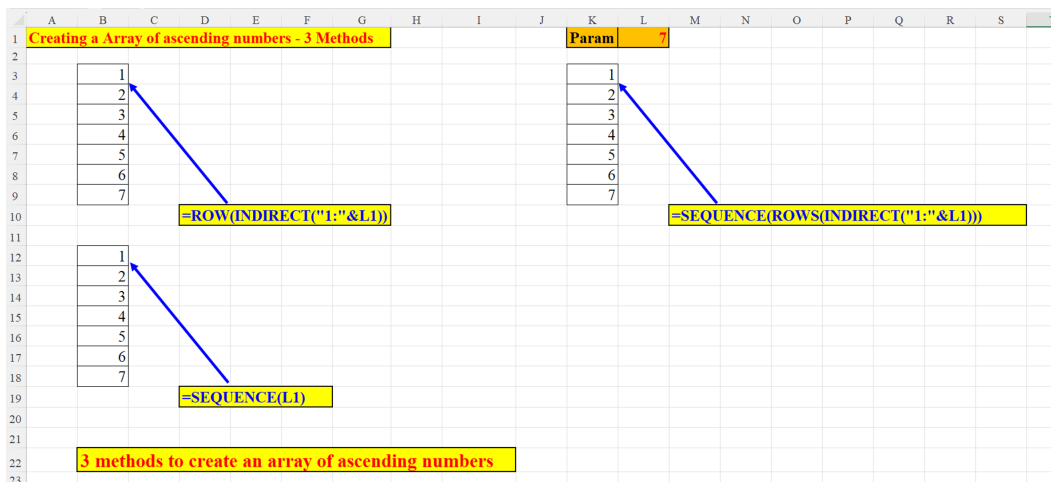


Figure 4.2: Creating an array of ascending numbers - 3 methods

From one cell to a vertical array

Here is an example of how to split a string into its components. We use a text function (**MID**) that separates the name to the characters composing it. The **SEQUENCE** function creates a vertical array of the components of the name in A1. The length of the array is, of course, identical to the original string's length as shown in the following screenshot:

	A	B	C	D	E	F	G	H	I	J	K
1	MARADONA		M								
2			A								
3			R								
4			A								
5			D								
6			O								
7			N								
8			A								
9											
10											
11											
12											
13			=MID(A1,SEQUENCE(LEN(A1)),1)								
14											
15											
16											
17			From one cell to a vertical array								
18											
19											
20											

Figure 4.3: From one cell to a vertical array

How many active months

The challenge is this: We have a string of 12 pairs of numbers, each pair representing a month: the first – January, the second – February etc. If the pair's numeric value is greater than 0, it means that in the corresponding month some transactions were made, and we must count this month as a profitable month. The formula:

=SUM(IF(MID(A1,SEQUENCE(,12,1,2),2)*1>0,1,0))

sums all these pairs to find out how many lucrative months we had in that year:

	A	B	C	D	E	F	G	H	I	J	K	L
1	000230000409112030240205											
2												
3												
4												
5												
6												
7												
8												
9												
10												
11												
12												
13												
14												
15												
16												
17												

Figure 4.4: How many active months

Build a dynamic array - horizontal or vertical

The formula in the following figure implements four elements:

- Whether we want a horizontal or vertical array. If cell B1 is “R”, then we want a vertical array (by Row), otherwise – we want a horizontal array.
- Cell D1 stores the array’s size: how many rows (if it is a vertical array) or columns (if it is a horizontal array).
- Cell F1 sets the starting value.
- Cell H1 determines the step/increment. It is 1 by default, so we can omit it if we want an increment of 1 between the array’s members:

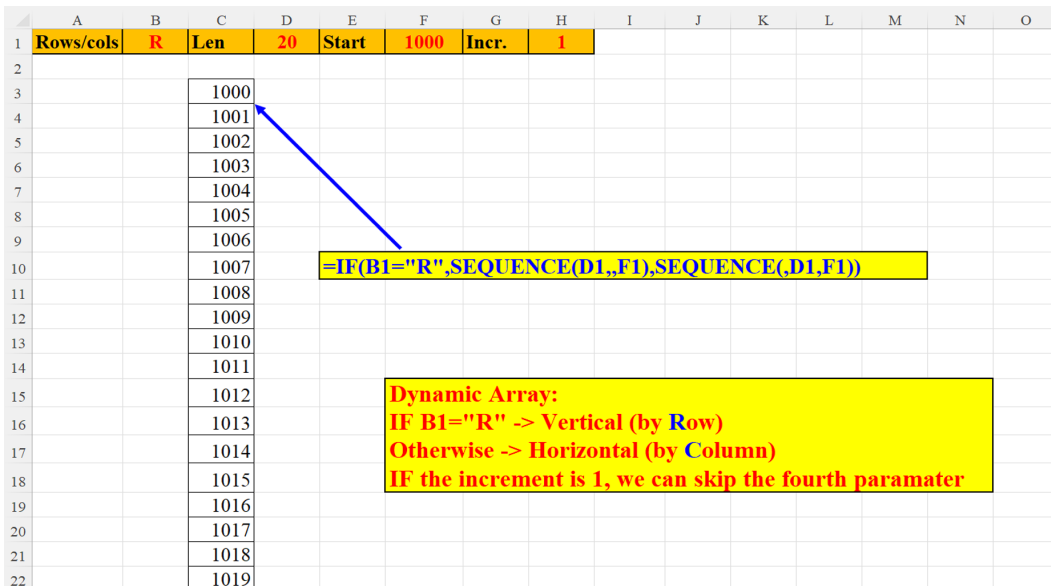


Figure 4.5: Build a dynamic array - Horizontal or Vertical

Flip columns horizontally

This section will display two methods to flip columns horizontally: the first one uses a dynamic range (a parameter) whereas the second one uses *hard coded* references. Both examples use advanced DAF, as explained in *Chapter 1, Introduction to Dynamic Array Functions in Excel 365*.

Method 1

The parameter (in K1) specifies the range to be flipped where we are using **INDIRECT**. This function *translates* the text in K1 into a valid reference in Excel. As explained above, it is not recommended to use many instances of the **INDIRECT** function in the

worksheet (see the section: *Creating an array of ascending numbers - three methods*), but here we use it only once to establish the size of the array to flip. The **SORTBY** function sorts the columns to flip (determined by the **SEQUENCE**) in descending order (its last argument is: -1) and the result: a *mirrored* array is shown in the following screenshot:

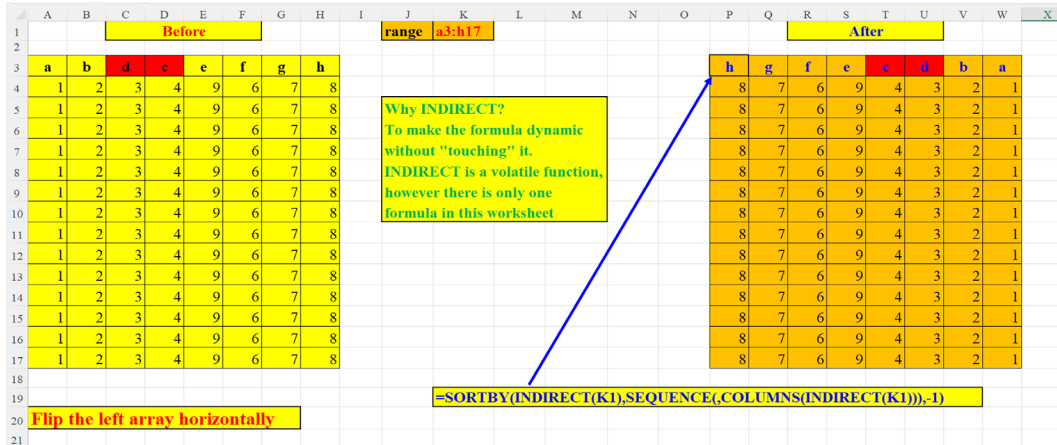


Figure 4.6: Flip Columns horizontally – Method 1

Method 2

Using the new DAF function **CHOOSECOLS** with our **SEQUENCE**, yields the desired *mirror* array. The **COLUMNS** function (appearing twice within the **SEQUENCE** function) returns both the number of columns to flip (3) and the column to start with. As in the previous example, the last argument to the **CHOOSECOLS** is negative to indicate the reverse order of columns in the resulting array. This is shown in the following screenshot:

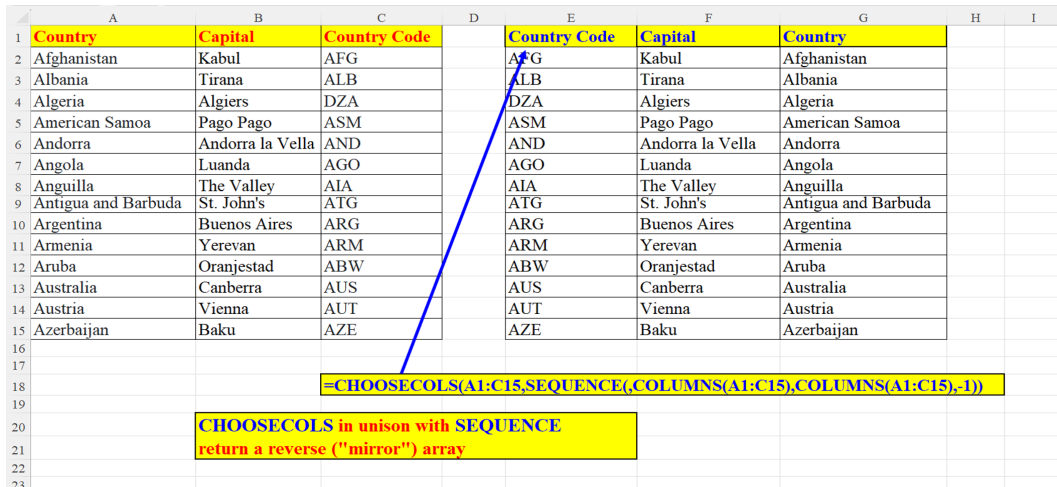


Figure 4.7: Flip columns horizontally – Method 2

Flip vertical array (with and without SEQUENCE)

The following figure shows two methods of flipping a vertical array *upside down*. On the right-hand side, we have a solution that uses the **OFFSET** function in tandem with the **COUNTA** function. This, of course, is not a dynamic solution. We need to drag the formula all the way down to row 13, the last row of the array. The solution suggested on the left-hand side is, of course, a dynamic solution that turns the array topsy-turvy in one go:

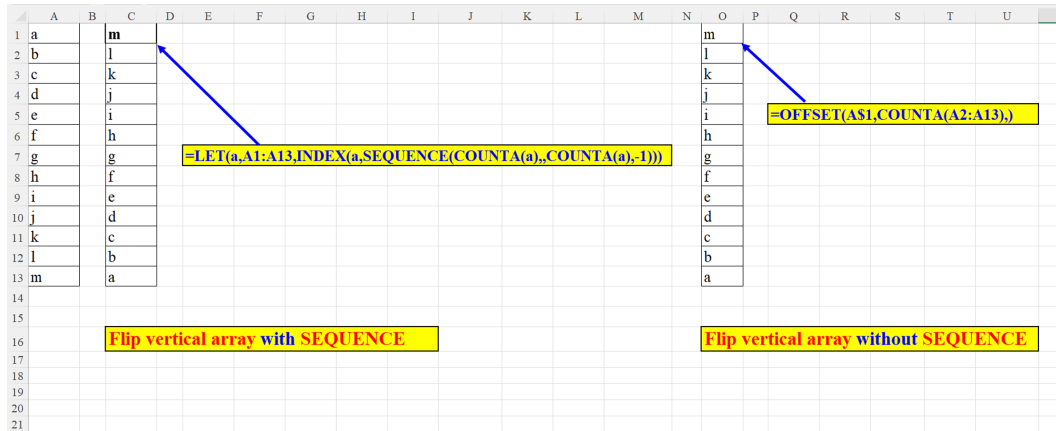


Figure 4.8: Flip vertical array (with and without SEQUENCE)

Flip part of vertical array using a parameter

The parameter (in I1) enables us to choose only part of the array to be flipped without *hard-coding* the range within the formula. This example is similar to the previous one, however the preceding figure (Figure 4.8) suggests flipping the whole array. The current example flips only a part of the array. This is done with the **INDIRECT** function. By using the **LET** function (which enables us to define variables of the formula) we can significantly shorten the original formula as shown in the following screenshot:

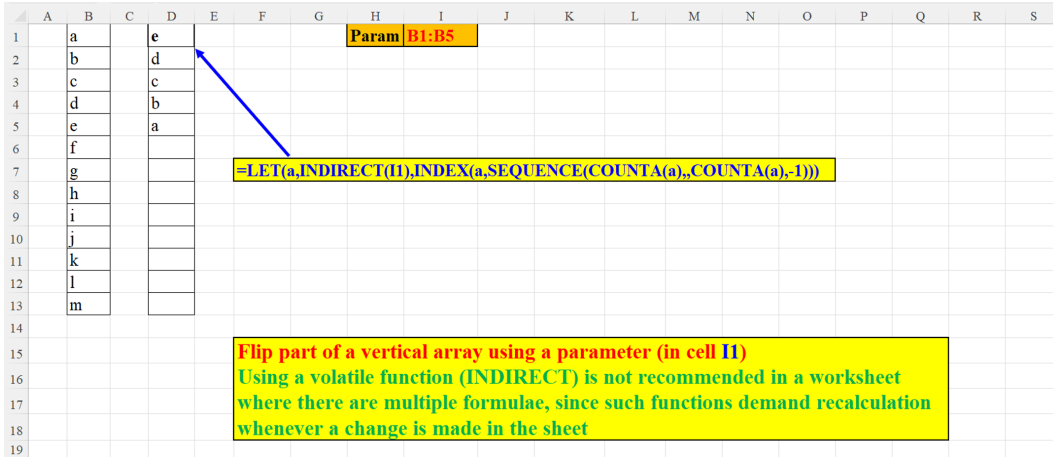


Figure 4.9: Flip part of vertical array using a Parameter

Create a two-dimensional array using four parameters

The following screenshot demonstrates creating a two-dimensional array. The number of rows is set dynamically: we divide the total number of items in the array (parameter in cell I1) by the desired number of columns (parameter in cell R1). The starting number is defined in cell L1, and the solution is flexible enough to allow us an array with increments greater than 1 or even smaller than 1 (a descending array). The increment is defined in cell O1:

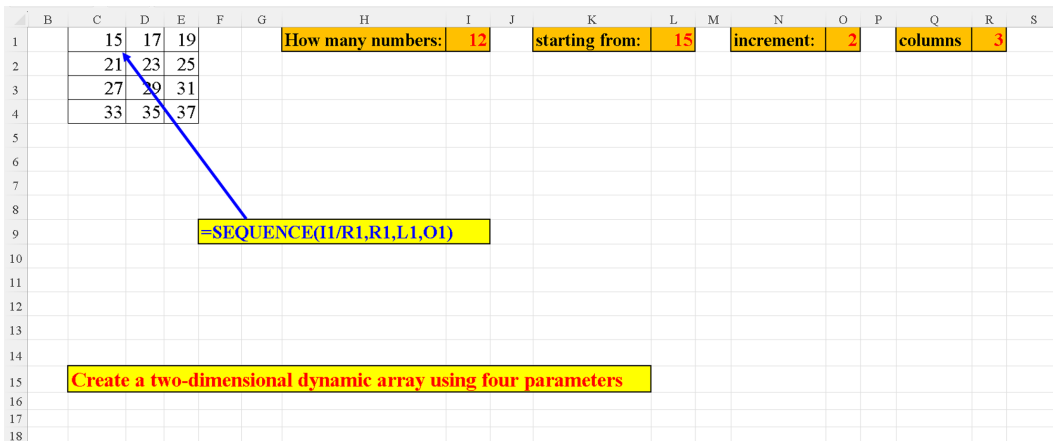


Figure 4.10: Create a two-dimensional array using four parameters

Flexible LARGE

The parameter in H1 defines the number of agents to display. This number is in fact the second argument of the **LARGE** function. It lets us create a dynamic header of the report (in cell D2) as shown in the following screenshot:

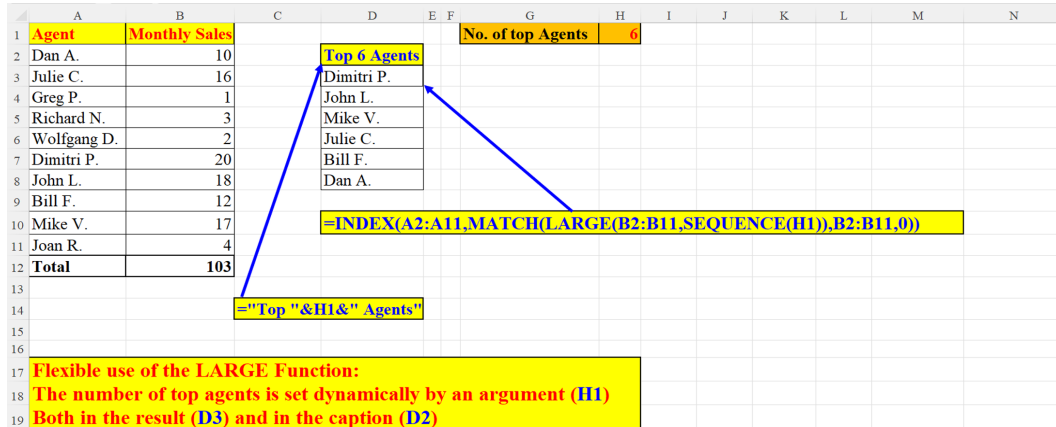


Figure 4.11: Flexible LARGE

MMULT with static ranges and with dynamic Arrays

The **MMULT** function returns the result of multiplication of two arrays. On the left-hand side formula, we use multiplication of ranges whereas on the right-hand side formula we multiply arrays. In the right-hand side formula, we need to specify explicitly all four arguments of the second **SEQUENCE** function: we need to know not only the size of the rectangular array (3 rows by three columns) but also its starting number (9) and its step/increment (-1) as shown in the following screenshot:

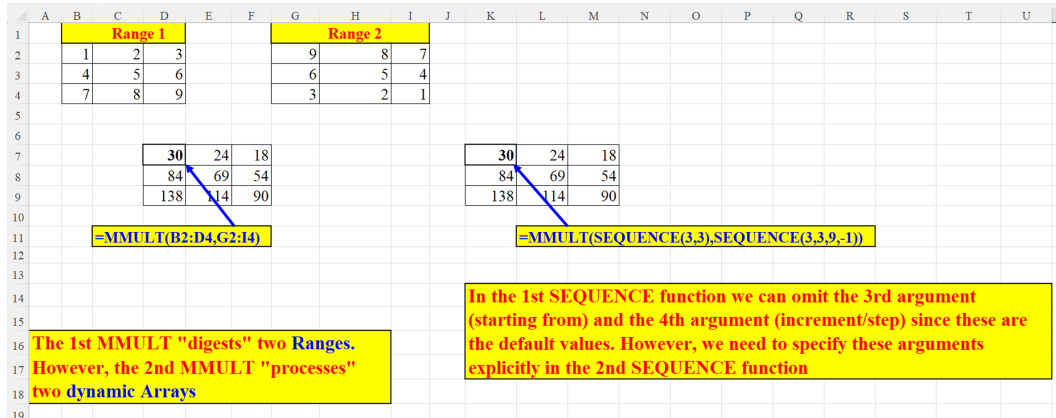


Figure 4.12: MMULT with static ranges and with dynamic arrays

Four useful tricks with VLOOKUP

When used with **VLOOKUP**, **SEQUENCE** can return multiple results in various forms. In the next four examples we present four such cases: fetching all columns for the search key, fetching all columns in reverse order, fetching data from the last two (out of the three) columns and finally, fetching data from the first and last (third) columns.

VLOOKUP - Fetch all columns of an item searched

The following figure demonstrates how to fetch all columns of a searched item. The second argument to the **SEQUENCE** function specifies the number of columns to be returned by the **VLOOKUP** function for the item specified in cell G1:

	A	B	C	D	E	F	G	H
1						Param	111	
2	Code	Tariff	Service					
3	111	1920.00	Morning					
4	112	1680.00	Noon					
5	113	2150.00	Evening					
6	114	2640.00	Premium					
7	115	1550.00	Youth					
8	116	1650.00	Student					
9	117	3350.00	Couple					
10	118	4250.00	Family					
11								
12	111	1920	Morning					
13								
14								
15								
16	=VLOOKUP(G1,\$A\$3:\$C\$10,SEQUENCE(,3),0)							
17								
18								
19	Fetch all data per lookup key							
20								

Figure 4.13: VLOOKUP - Fetch all columns of item searched

VLOOKUP - Fetch all data per lookup key in reverse order

In the following example, **SEQUENCE** tells **VLOOKUP** to return backwards the data found for the search key (parameter in cell G1): fetch three columns (second

argument) starting from the third column (third argument) in reverse order (fourth argument):

	A	B	C	D	E	F	G	H	I	J
1						Param	115			
2	Code	Tariff	Service							
3	111	1920.00	Morning							
4	112	1680.00	Noon							
5	113	2150.00	Evening							
6	114	2640.00	Premium							
7	115	1550.00	Youth		Youth	1550	115			
8	116	1650.00	Student							
9	117	3350.00	Couple							
10	118	4250.00	Family							
11										
12						=VLOOKUP(G1,\$A\$3:\$C\$10,SEQUENCE(,3,3,-1),0)				
13										
14										
15										
16										
17						Fetch all data per lookup key in reverse order				
18										

Figure 4.14: VLOOKUP - Fetch all data per lookup key in reverse order

VLOOKUP - Fetch last two columns for a search key

The following screenshot illustrates how to fetch the last two columns for the key defined in cell G1: start from the second column (SEQUENCE's argument number 2) and get two columns (argument number 3):

	A	B	C	D	E	F	G	H	I	J
1						Param	115			
2	Code	Tariff	Service							
3	111	1920.00	Morning							
4	112	1680.00	Noon							
5	113	2150.00	Evening							
6	114	2640.00	Premium							
7	115	1550.00	Youth		1550	Youth				
8	116	1650.00	Student							
9	117	3350.00	Couple							
10	118	4250.00	Family							
11										
12						=VLOOKUP(G1,\$A\$3:\$C\$10,SEQUENCE(,2,2),0)				
13										
14										
15										
16						Fetch last two columns per lookup key				
17										

Figure 4.15: VLOOKUP - Fetch last two columns for a search key

VLOOKUP - Fetch the first and third data items per lookup key

The following screenshot depicts a case where we fetch only the first and third columns of the data items for the key in cell G1. **VLOOKUP**'s second argument (table array defined in the **SEQUENCE** function) returns two columns, starting with the first table's array column and skipping the second column (**SEQUENCE**'s last argument is: 2, which means: *Step 2* columns):

	A	B	C	D	E	F	G	H	I	J	K	L
1						Param	111					
2	Code	Tariff	Service									
3	111	1920.00	Morning		111	Morning						
4	112	1680.00	Noon									
5	113	2150.00	Evening									
6	114	2640.00	Premium									
7	115	1550.00	Youth		=VLOOKUP(G1,\$A\$3:\$C\$10,SEQUENCE(2,1,2),0)							
8	116	1650.00	Student									
9	117	3350.00	Couple									
10	118	4250.00	Family									
11												
12												
13												
14												
15	Fetch the 1st and 3rd data items per lookup key											
16												
17												

Figure 4.16: VLOOKUP - Fetch the first and third data items per lookup key

From vertical to horizontal – Two Methods

The challenge here is to convert data in one column (where there are gaps between the contiguous *chunks* of data) into a two-column array (in the first example) or three-column array (in the second example).

Method 1

The first method first filters all the data (non-empty) cells, while the **SEQUENCE** function sets an array of three rows and two columns, to host the data filtered: 6 data rows divided into two columns, as shown in the following screenshot. The

LET function allows you to assign a name to a variable. In our example, the *r* is a substitute (shorthand) to the expression B2:B9 (in our case, the range: B2:B9):

	A	B	C	D	E	F	G	
1		Before			After			
2		Abraham Lincoln			Abraham Lincoln	February 12, 1809 – April 15, 1865		
3		February 12, 1809 – April 15, 1865			John Fitzgerald Kennedy	May 29, 1917 – November 22, 1963		
4					Franklin Delano Roosevelt	January 30, 1882 – April 12, 1945		
5		John Fitzgerald Kennedy						
6		May 29, 1917 – November 22, 1963						
7								
8		Franklin Delano Roosevelt			=LET(r,B2:B9,INDEX(FILTER(r,r<>""),SEQUENCE(COUNTA(r)/2,2),1))			
9		January 30, 1882 – April 12, 1945						
10								
11								
12								
13								
14		From vertical to horizontal.						
15		Each two rows are joined into one row in two columns						
16								
17								

Figure 4.17: From vertical to horizontal - Method 1

Method 2

The second method splits the data in column A (rows 2-17) into a two-dimensional 4*4 array. This array is then *cleaned* of its 4th rows (which are empty) and the resulting array consists of 4 names in three columns. Again, we use here (as in our previous sub-section) the **LET** function to define a “shorthand” for a lengthy expression: **INDEX(A:A,SEQUENCE(16/4,4,2))**.

The **SEQUENCE** function defines a bi-dimensional array: 4 rows (there are 16 rows in the range (2:17) for each person, and we want to *transpose* each such 4 rows into 4 columns. Since the **INDEX** function returns 0 for an empty cell (each fourth row becomes the fourth column), we then remove these zeroes with the **IF** function. The result (on the right-hand side of the picture) displays three data items per person in each row:

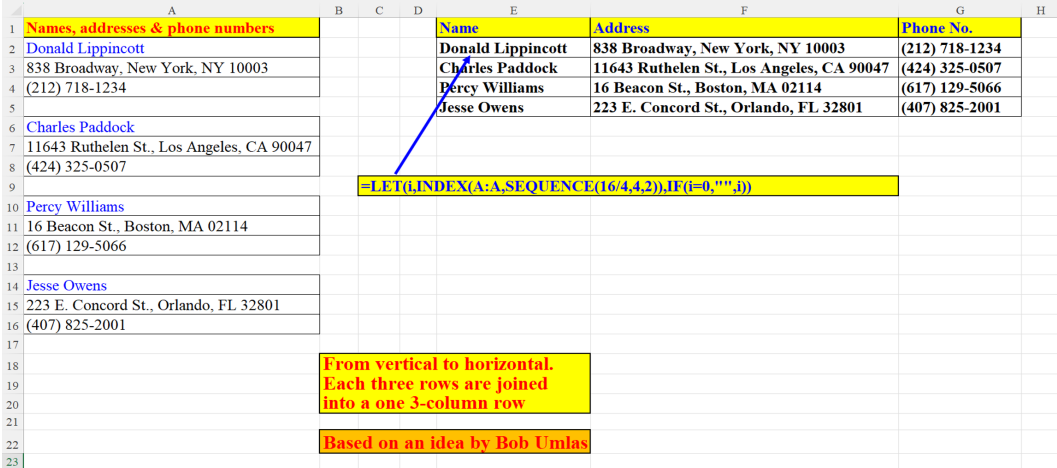


Figure 4.18: From vertical to horizontal - Method 2

Four two-dimensional arrays generated by two parameters

The two parameters in L1 and L2, enable us to generate four different kinds of arrays. The two arrays on the left-hand side consist of a 2-row by 6-column array and a 6-row by 2-column array. The two arrays on the right-hand side are in fact the swapping of each dimension of their respective array on the left: the array in P2:Q8 manifests the shift from 2 rows by 6 columns into 6 rows by two columns, and the array in P11:U12 shows the shift from 6 rows by two columns into 6 columns by two rows.

You need to pay heed to the special notation of a spilled array in the formula in P2 and P11: the # sign following the cells address (B2), denotes a spilled array which starts at B2 as shown in the following screenshot:

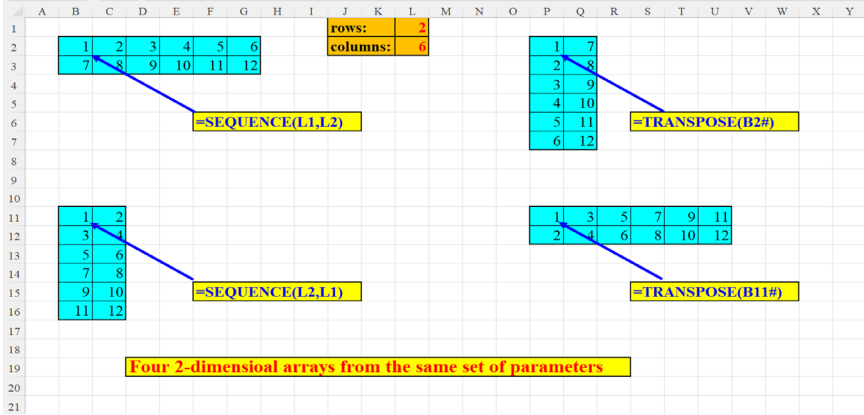


Figure 4.19: Four two-dimensional arrays generated by two parameters

Select columns by parameters

The **CHOOSECOLS** function (mentioned and explained in *Chapter 1, Introduction to Dynamic Array Functions in Excel 365*) enables us to choose certain columns from an array. In this case, we want to select the last two columns (third and fourth) of the original array. The parameters in L3 and L4 (which are data validation lists whose values are 1-4) determine respectively the numbers of columns to display and the starting column number. The **IF** function within the **SEQUENCE** verifies that no error is produced when selecting incompatible values from the parameters as shown in the following screenshot:

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Country	Capital	Country Code	Continent									
2	Afghanistan	Kabul	AFG	Asia		Kabul	AFG	Asia					
3	Albania	Tirana	ALB	Europe		Tirana	ALB	Europe					
4	Algeria	Algiers	DZA	Africa		Algiers	DZA	Africa				no. of col.	3
5	American Samoa	Pago Pago	ASM	America		Pago Pago	ASM	America				from col.	2
6	Andorra	Andorra la Vella	AND	Europe		Andorra la Vella	AND	Europe					
7	Angola	Luanda	AGO	Africa		Luanda	AGO	Africa					
8	Anguilla	The Valley	AIA	America		The Valley	AIA	America					
9	Antigua and Barbuda	St. John's	ATG	America		St. John's	ATG	America					
10	Argentina	Buenos Aires	ARG	America		Buenos Aires	ARG	America					
11	Armenia	Yerevan	ARM	Asia		Yerevan	ARM	Asia					
12	Aruba	Oranjestad	ABW	America		Oranjestad	ABW	America					
13	Australia	Canberra	AUS	Australia		Canberra	AUS	Australia					
14	Austria	Vienna	AUT	Europe		Vienna	AUT	Europe					
15	Azerbaijan	Baku	AZE	Asia		Baku	AZE	Asia					
16													
17			=CHOOSECOLS(A2:D15,SEQUENCE(L3,IF(L4+L3>5,(5-L3),L4)))										
18													
19													
20													
21													
22													
23													

Return columns from an array: the number of columns returned as well as the starting column, are set by the parameters in two data validation cells: L3 & L4

Figure 4.20: Select columns by parameters

Transpose a vertical array without knowing its size beforehand

If we do not know in advance the size of the source array, but know where it starts, then the **COUNTA** function (the second argument: number of columns which should be equal to the number of rows in the original array) will help up set up the size of the output array as shown in the following screenshot:

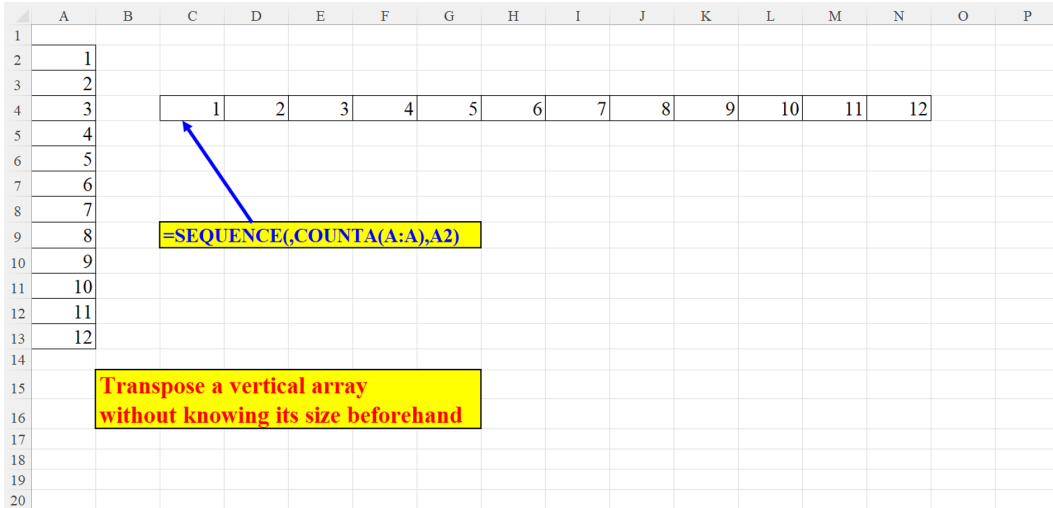


Figure 4.21: Transpose a vertical array without knowing its size beforehand

Fetching multiple results for a search value

How do we fetch multiple results for a certain agent who has multiple records in the input array?

First, we create a unique list of agents (in cells J2:J7). Then, we can select the agent from the drop-down list of the data validation of cell F2. The **INDEX** conjoined with the **SEQUENCE** delivers some empty non-valid values which are eliminated from the list with the assistance of the **TOCOL** function as shown in the following screenshot:

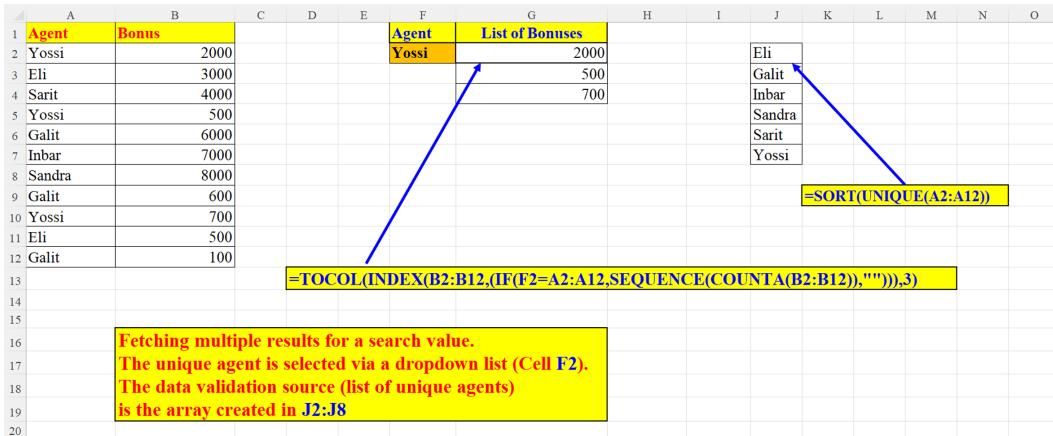


Figure 4.22: Fetching multiple results for a search value

Vertical to horizontal without TRANSPOSE

The vertical array of months (A2:A13) is converted into a horizontal array. The **SEQUENCE** function yields a 12-item horizontal array which is *populated* by the **INDEX** function as shown in the following screenshot:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Months		January	February	March	April	May	June	July	August	September	October	November	December	
2	January														
3	February														
4	March														
5	April														
6	May														
7	June														
8	July														
9	August														
10	September														
11	October														
12	November														
13	December														
14															
15															
16															
17															
18															

Figure 4.23: Vertical to horizontal without TRANSPOSE

Conclusion

In this chapter we referred to cases where the **SEQUENCE** can be easily used when dealing with arrays. Since **SEQUENCE** is a Dynamic Array Function (see explanation in *Chapter 1, Introduction to Dynamic Array Functions in Excel 365*) it is clear that this chapter perfectly illustrates its role as an array formula.

The next chapter will explain in detail the **SEQUENCE** function in the context of dates and times.

Points to remember

- SEQUENCE adapts itself flawlessly to working with arrays.
- SEQUENCE can build arrays on the fly, either horizontally or vertically.
- SEQUENCE easily generates one- or bi-dimensional arrays.
- SEQUENCE can return multiple results in various forms if you use it with VLOOKUP.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



CHAPTER 5

SEQUENCE in

Date and

Time Operations

Introduction

Excel is indispensable when dealing with time operations. It helps us calculate periods of time (seconds, minutes, hours, days, weeks, months and years). We use time in calendars, for scheduling, measuring duration of an activity, calculations involving timespan and of course in myriad of other cases and scenarios. This chapter will focus on all these features. It is my opinion that when it comes to dealing with dates and times in Excel, there is no function as advantageous as **SEQUENCE**. Multiple examples in this chapter will exemplify **SEQUENCE**'s versatility and its ease of use in this context. We will demonstrate (among other things) not only how to calculate durations of time, set schedules, and so on, but also how to build dynamic and flexible monthly and yearly calendars.

Structure

In this chapter, we will discuss the following topics:

- Examples of **SEQUENCE** with date and time
 - 12 months with each month's first day
 - The year's months with the last day of each month - three methods
 - N consecutive dates starting from today (two methods)

- Display month names without a specific date
- 2 Methods to display the weekday names
- Adding minutes to time
- Adding seconds to time
- **SEQUENCE** of days in a given month
- A **SEQUENCE** of dates (between start and end dates)
- Extract only time – four traditional methods
- Extract only time – a new method
- Presence in class by month
- How many Mondays are there in a given month
- How many Saturdays between two dates
- Display only dates of Wednesdays in 2020
- How many Wednesdays are there in 2020
- **SEQUENCE** of the month's last date for each month (two methods)
- A horizontal **SEQUENCE** of descending dates: first of each month
- A substitute for NETWORKDAYS.INTL (for a certain month)
- A substitute to the NETWORKDAYS.INTL (any period)
- A substitute to NETWORKDAYS.INTL - with/without weekends
- How many working days are there in each month of a given period
- How many eligibilities days
- The doctor's schedule (2 versions)
- The doctor's schedule (version 1)
- The doctor's schedule (version 2)
- Monthly calendar – classic vs. non-classic
- Monthly calendar in 20 languages
- Two methods for creating a list of the month's days
- Yearly calendar – good vs. bad
- Dynamic yearly calendar – in one formula
- Dynamic yearly calendar – Conditional Formatting
- Dynamic yearly calendar - by month
- Dynamic yearly calendar - by week
- Yearly horizontal calendar with highlighted weekday (two examples)

- Yearly horizontal calendar
- Yearly vertical calendar with highlighted weekday (two examples)
- Conditional Formatting – each weekday is formatted differently

Objectives

If you read this chapter thoroughly and exercise the examples given in it, you will be able to implement the **SEQUENCE** function in a huge range of related situations and problems.

Examples of SEQUENCE with date and time

The **SEQUENCE** function can be of great assistance when applied to date and time. We are going to exhibit its abilities in this territory, from the tiniest (second) to the immense (year).

12 months with each month's first day

This simple formula displays the dates of the first of each month for the year 2021. Of course, the Year argument is superfluous, but did you know that if you omit it, you will get the list of months for the year 1900? (which in Excel is Year 0). This is illustrated in the following screenshot:

	A	B	C	D	E	F	G	H
1	01/01/2021			Year	2021			
2	01/02/2021							
3	01/03/2021							
4	01/04/2021							
5	01/05/2021	=DATE(E1,SEQUENCE(12),1)						
6	01/06/2021							
7	01/07/2021							
8	01/08/2021							
9	01/09/2021							
10	01/10/2021							
11	01/11/2021							
12	01/12/2021							
13								
14								
15								
16								
17								
18								

Figure 5.1: Sequence of the year's 12 months: each with the month's first day

The year's months with the last day of each month - three methods

There are three methods mentioned in this section that will show you how to create an array of the year's months, with each month's last day. In this section, we will learn how to display the last date of each month of the year.

Method 1

The first method uses a trick by which you do not need to specify the last argument of the **DATE** function (that is, the **DAY**): If the month's number is **current+1** (for example, 2 for January 3 for February, and so on) then the formula automatically generates the last day of the month. You can omit the **DAY** argument as shown in the following screenshot:

	A	B	C	D	E	F
1				Year	2020	
2						
3	31/01/2020	Method 1				
4	29/02/2020					
5	31/03/2020					
6	30/04/2020					
7	31/05/2020	=DATE(SES1,SEQUENCE(12,,2),)				
8	30/06/2020					
9	31/07/2020					
10	31/08/2020					
11	30/09/2020					
12	31/10/2020					
13	30/11/2020					
14	31/12/2020	Method 1				
15		Sequence of the year's dates of each LAST day of the month				
16						
17	An additional trick: if the month is current +1, it automatically gets the last month's date. No need to specify the day argument of the DATE function					
18						
19						
20						

Figure 5.2: Display the last date of each month of the year – method 1

Method 2

The second method uses the following trick: the **SEQUENCE(12,0)** generates the array: 0,1,2,...11. So, the dates array created by the **DATE** function is the following: 01/12/2019, 01/01/2020, ...,01/11/2020. Now, adding 1 month (last argument of the **EOMONTH** function) creates the desired array as shown in the following screenshot:

	A	B	C	D	E	F
1				Year	2020	
2						
3	31/01/2020		Method 2			
4	29/02/2020					
5	31/03/2020					
6	30/04/2020					
7	31/05/2020		=EOMONTH(DATE(SES1,SEQUENCE(12,,0),1),1)			
8	30/06/2020					
9	31/07/2020					
10	31/08/2020					
11	30/09/2020					
12	31/10/2020					
13	30/11/2020					
14	31/12/2020		Method 2			
15			Sequence of the year's dates of each LAST day of the month			
16						
17						
18						
19						
20						

Figure 5.3: Display the last date of each month of the year – method 2

Method 3

This method is a little simpler than the previous one, yet it uses the same functions to generate the sought-after result. Here, the **DATE** function yields the first dates of each month in 2020. The second argument of the **EOMONTH** (**0**), converts the date into the month's last date as shown in the following screenshot:

	A	B	C	D	E	F	G
1				Year	2020		
2							
3	31/01/2020		Method 3				
4	29/02/2020						
5	31/03/2020						
6	30/04/2020						
7	31/05/2020		=EOMONTH(DATE(E1,SEQUENCE(12),1),0)				
8	30/06/2020						
9	31/07/2020						
10	31/08/2020						
11	30/09/2020						
12	31/10/2020						
13	30/11/2020						
14	31/12/2020		Method 3				
15			Sequence of the year's dates of each LAST day of the month				
16							
17							

Figure 5.4: Display the last date of each month of the year – method 3

N consecutive dates starting from today (two methods)

Method 1

SEQUENCE's first parameter is the number of days (taken from the parameter in cell G1). **SEQUENCE**'s last parameter (value to start with) is today.

One can, of course, start with any date. In such a case, we would add the starting date as an additional parameter as shown in the following screenshot:

	A	B	C	D	E	F	G	H	I
1	21/02/2023					Param	20		
2	22/02/2023								
3	23/02/2023			Method 1					
4	24/02/2023								
5	25/02/2023								
6	26/02/2023			=SEQUENCE(G1,TODAY())					
7	27/02/2023								
8	28/02/2023								
9	01/03/2023								
10	02/03/2023								
11	03/03/2023								
12	04/03/2023								
13	05/03/2023								
14	06/03/2023								
15	07/03/2023								
16	08/03/2023								
17	09/03/2023								
18	10/03/2023								
19	11/03/2023			Method 1					
20	12/03/2023			Sequence of N consecutive dates starting from Today					
21									
22									
23									

Figure 5.5: N consecutive dates starting from today – method 1

Method 2

In this example, the starting date (Today) is not an argument to the **SEQUENCE** function. It is the *starting point* from which we add the **SEQUENCE** of days, according to the parameter in G1 as shown in the following screenshot:

	A	B	C	D	E	F	G	H	I
1	13/02/2023					Param	20		
2	14/02/2023								
3	15/02/2023								
4	16/02/2023								
5	17/02/2023								
6	18/02/2023								
7	19/02/2023								
8	20/02/2023								
9	21/02/2023								
10	22/02/2023								
11	23/02/2023								
12	24/02/2023								
13	25/02/2023								
14	26/02/2023								
15	27/02/2023								
16	28/02/2023								
17	01/03/2023								
18	02/03/2023								
19	03/03/2023								
20	04/03/2023								
21									
22									

Figure 5.6: N consecutive dates starting from today – method 2

Display month names without a specific date

The formula in Figure 5.7 does not need a date to display the months names, nor does it need any other function. The **DATE** function uses the same trick as in Figure 5.2: if the month is current+1, it automatically generates the month's last date (so we do not need the **DAY** argument of the **DATE** function). Moreover, we can omit **DATE**'s first argument (**YEAR**) and excel interprets the missing argument as 0: year 0 in Excel in 1900, so we have created a *legal* date that Excel can *understand*. The **DATE** function then generates a sequence of numbers which are the cumulative days for each month from the beginning of the year. If you wonder why **b1** is the prefix of the **TEXT**'s second argument: Since my Windows default language is not English, if I omitted the **b1** and specified only **mmmm**, Excel would display the result in my native language. To have the month names in

English this is the shortest method. Of course, if your Windows' default language is English, then the **b1** prefix is redundant as shown in the following screenshot:

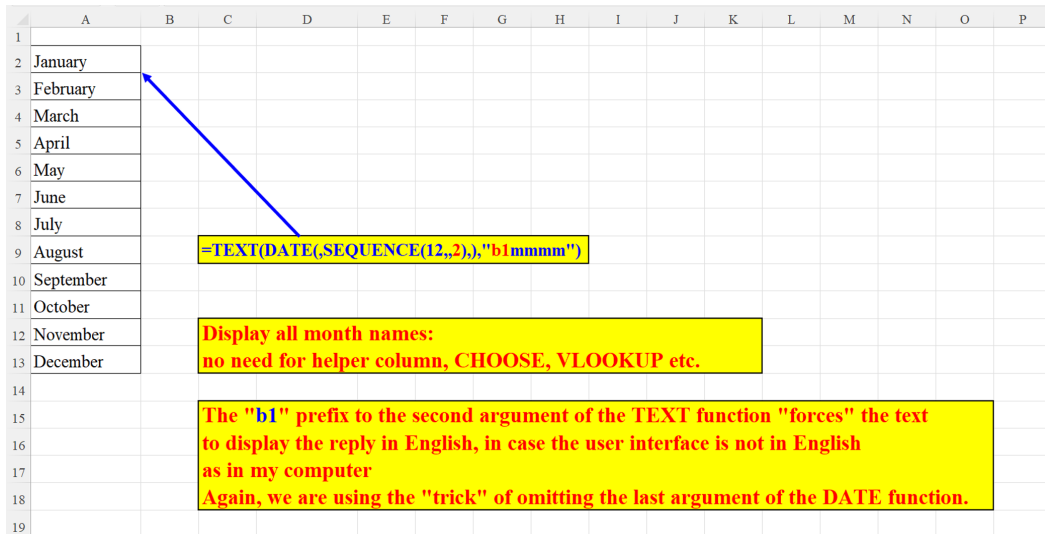


Figure 5.7: Display month names without a specified date

Two methods to display the weekday names

The **TEXT** function transforms an array of seven numbers into weekday names. Please refer to the previous section, *Display month names without a specific date* for an explanation of the **b1** prefix of **TEXT**'s second argument as shown in the following screenshot:

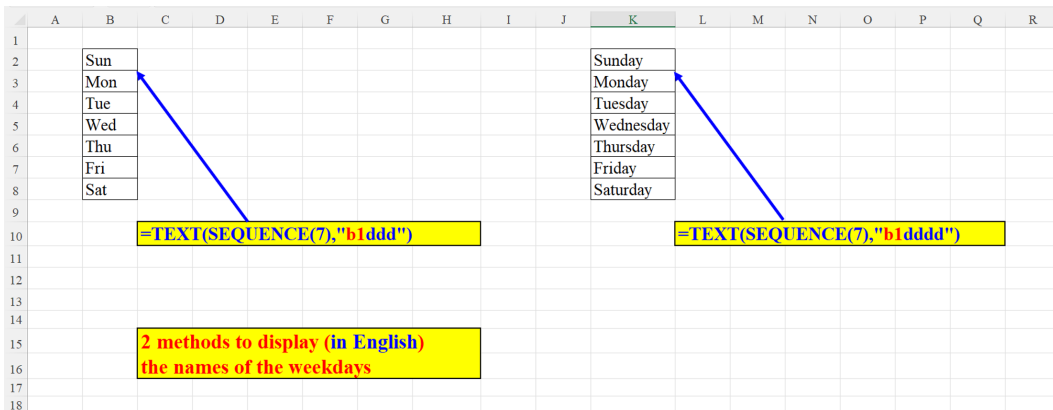


Figure 5.8: Two methods to display the weekday names

Adding minutes to time

The formula builds a dynamic array of time gaps within an hour. The time gap is given as a parameter in cell H1. The parameter for the starting hour is defined in cell E1. The size of the array is determined by the gap size. For example, if it is 4, then there are: $60/4 = 15$ gaps as shown in the following screenshot:

	A	B	C	D	E	F	G	H	I	J
1				Hour	12		Diff in minutes	4		
2		12:04:00								
3		12:08:00								
4		12:12:00								
5		12:16:00								
6		12:20:00								
7		12:24:00								
8		12:28:00								
9		12:32:00								
10		12:36:00								
11		12:40:00								
12		12:44:00								
13		12:48:00								
14		12:52:00								
15		12:56:00								
16		13:00:00								
17										
18										
19										

**Adding minutes (H1) to an hour (E1)
to build a dynamic time gap for one hour**

Figure 5.9: Adding minutes to time

Adding seconds to time

In the following example we will create a time gap in seconds. The gap size is given as a parameter in cell H1. The parameter for the starting hour is defined in cell E1, and the size of the array is set in cell K1 as shown in the following screenshot:

	A	B	C	D	E	F	G	H	I	J	K	L
1				Hour	12		Seconds	4		Array size	10	
2		12:00:04										
3		12:00:08										
4		12:00:12										
5		12:00:16										
6		12:00:20										
7		12:00:24										
8		12:00:28										
9		12:00:32										
10		12:00:36										
11		12:00:40										
12												
13												
14												
15												
16												
17												
18												
19												

**Adding seconds (H1) to an hour (E1)
The formula displays an array whose size is K1 items**

Figure 5.10: Adding seconds to time

Sequence of days in a given month

In this section, we will learn how to create the array of days in a given month (cell I2) and year (cell I1). Here, we will again use the *trick* of omitting the **DAY** argument from the **DATE** function (see preceding section: *Display month names without a specific date*) as shown in the following screenshot:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	1							Year	2021					
2	2							Month	2					
3	3													
4	4													
5	5													
6	6													
7	7													
8	8						=SEQUENCE(DAY(DATE(I1,I2+1,)))							
9	9													
10	10													
11	11													
12	12													
13	13													
14	14													
15	15													
16	16													
17	17													
18	18													
19	19													
20	20													
21	21													
22	22													
23	23													
24	24													
25	25													
26	26													
27	27													
28	28													

Figure 5.11: SEQUENCE of days in a given month

A SEQUENCE of dates (between start and end dates)

In this section we will generate an array of dates where only the start date (cell C1) and the end date (cell C2) are known. The sequence is not limited to the scope of only one month. It can span over more than one month as shown in the following screenshot:

	A	B	C	D	E	F	G	H	I	J	K	L	M
1		start date	21/01/2021		21/01/2021								
2		end date	02/02/2021		22/01/2021								
3					23/01/2021								
4					24/01/2021								
5					25/01/2021								
6					26/01/2021	=SEQUENCE(C2+1-C1,C1,1)							
7					27/01/2021								
8					28/01/2021								
9					29/01/2021								
10					30/01/2021								
11					31/01/2021								
12					01/02/2021								
13					02/02/2021								
14													
15													
16		Create a sequence of dates where only the first date (C1) and last date (C2) are given											
17													
18													
19													

Figure 5.12: A Sequence of dates (between start and end date)

Extract only time –four traditional methods

Here are four traditional procedures to extract only the time from cells holding both dates and times:

- The first one (in cell C2) is using Excel's **TIME** function.
- The second one (cell C7) extracts the time (which is a fraction) by using the **MOD** function.
- The third method subtracts the **INT** (integer), which is the date portion of cell A12.
- And the fourth one uses the **TRUNC** function (which removes the fractional part of the number in C17, so that after the subtraction we are left with the fraction

All four methods are shown in the following screenshot:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Date and Time		Only Time											
2	20/11/2022 14:45		14:45			Method 1								
3														
4			=TIME(HOUR(A2),MINUTE(A2),)											
5														
6	Date and Time		Only Time											
7	20/11/2022 14:45		14:45			Method 2								
8														
9			=MOD(A7,1)											
10														
11	Date and Time		Only Time											
12	20/11/2022 14:45		14:45			Method 3								
13														
14			=A12-INT(A12)											
15														
16	Date and Time		Only Time											
17	20/11/2022 14:45		14:45			Method 4								
18														
19			=A17-TRUNC(A17)											
20														
21														
22	4 traditional ways to extract only the time from a cell formatted as Date & Time													
23														
24														

Figure 5.13: Extract only time – four traditional methods

Extract only time – a new method

In addition to the four known techniques to draw out the time part of a Date and Time cell, here is another method using **SEQUENCE**. First, it locates the “.” (the separator denoting the decimal place). Then, it retrieves the decimal part and converts it from a string to a number. Since the cell holding the result is formatted as “hh:mm” the result appears as an hour as shown in the following screenshot:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1														
2														
3	Date and Time		Only Time											
4	20/11/2022 14:45		14:45											
5														
6														
7														
8			=MID(A4,MATCH(1,FIND(".",MID(A4,SEQUENCE(LEN(A4),1)),0),10)*1											
9														
10														
11														
12														
13														
14														
15	A unique method to extract only the time from a cell formatted as Date & Time													
16														
17														
18														

Figure 5.14: Extract only time – a new method

Presence in class by month

The following figure exemplifies the use of the **FREQUENCY** function concomitantly with our **SEQUENCE**. It shows the number of times the student showed up during the Academic year, month by month. Please note that the list of dates need not be sorted, as it is done automatically by the **FREQUENCY** function as shown in the following screenshot:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Dates when present			Frequency	Month									
2	01/07/2022			0	January									
3	02/07/2022			0	February									
4	04/07/2022			0	March									
5	02/04/2022			2	April									
6	03/04/2022			0	May									
7	09/07/2022			1	June									
8	06/06/2022			5	July									
9	12/07/2022			5	August									
10	12/08/2022			4	September									
11	14/08/2022			4	October									
12	15/08/2022			0	November									
13	16/08/2022			0	December									
14	20/08/2022			0										
15	27/09/2022													
16	28/09/2022													
17	29/09/2022													
18	30/09/2022													
19	01/10/2022													
20	02/10/2022													
21	03/10/2022													
22	06/10/2022													

Formula in cell G7: `=TEXT(DATE(SEQUENCE(12,,2)), "b1mmmm")`

Formula in cell C16: `=FREQUENCY(MONTH(B2:B22), SEQUENCE(12))`

Text in cell M21: **How many times did the student show up during the year in each month? Please note that the presence dates need not be arranged**

Figure 5.15: Presence in class by month

How many Mondays are there in a given month

In this section, we will learn to calculate the number of Mondays in a given month.

The formula receives two arguments:

- The desired month
- The wanted weekday

How many Mondays (parameter in cell B2) were there in January 2022 (cell A2). You are free to choose any month or weekday, according to your needs as shown in the following screenshot:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Date	Day of Week												
2	01/01/2000	2												
3														
4														
5			5											
6														
7														
8														
9														
10	=LET(m,SEQUENCE((EOMONTH(A2,0)-A2+1),,A2,1),COUNT(FILTER(m,WEEKDAY(m)=B2)))													
11														
12														
13														
14														
15														
16														
17														
18														
19														
20														
21														

Figure 5.16: How many Mondays are there in a given month

How many Saturdays between two dates?

In this section we will read about two methods that will help us find how many Saturdays there are between two days:

Method 1

This example illustrates Saturday (weekday is a parameter defined in cell G2). The parameters for the start date and the end date are in cells E2 and F2, respectively. These, of course, can be modified for any other period or any other weekday. Besides, the **SUM** function used in this example can be perfectly replaced by the **SUMPRODUCT** function:

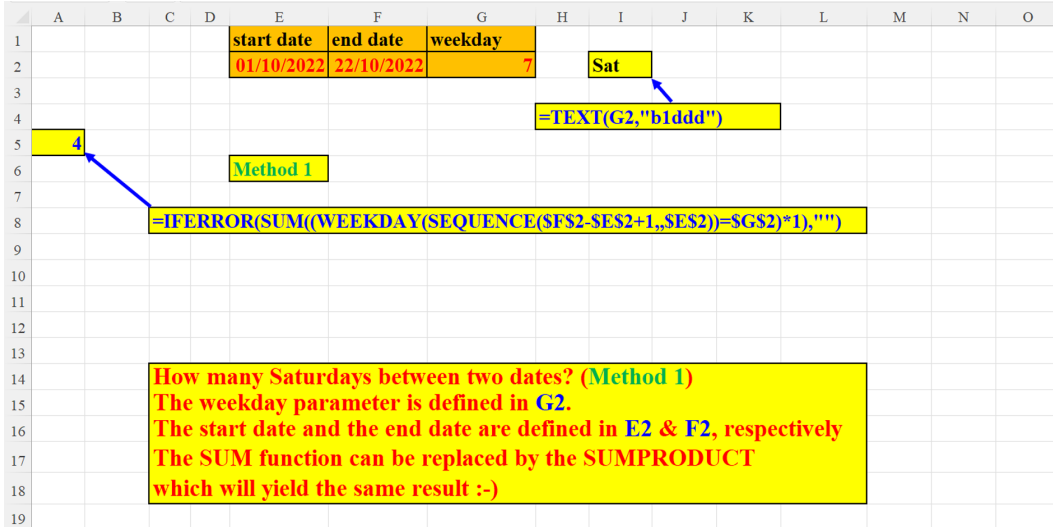


Figure 5.17: How many Saturdays in a given period – method 1

Method 2

Another procedure to achieve the same goal: find how many Saturdays there are in a certain period of dates.

The method exhibited here is quite similar to the one described in the preceding section: *How many Mondays are there in a given month* as shown in the following screenshot:

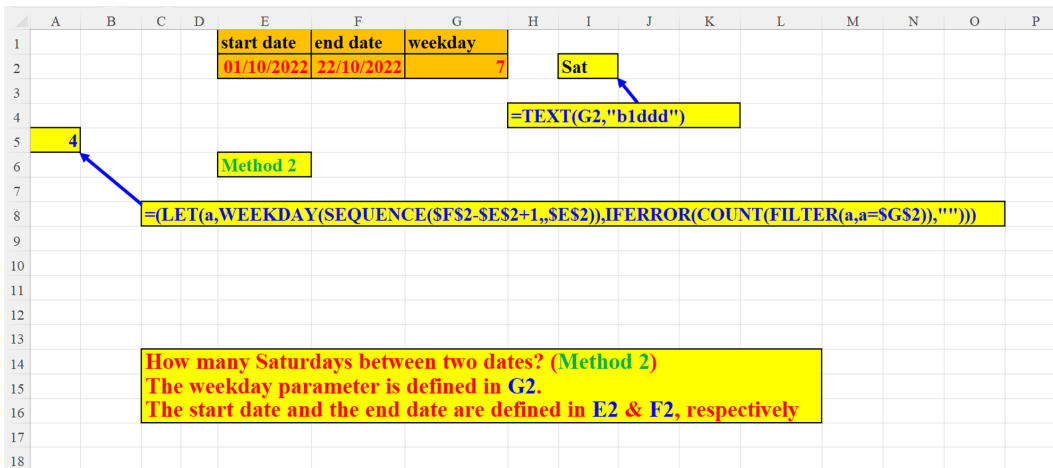


Figure 5.18: How many Saturdays in a given period – method 2

Display only dates of Wednesdays in 2020

In this example we want to find out all the dates of Wednesdays in the year 2020. The first part of the formula determines whether the parameter year (in cell G2) is a leap year. The second parameter filters out all the Wednesdays (parameter in cell H2) as shown in the following screenshot:

(Calendar image by: www.blank-calendar.com):

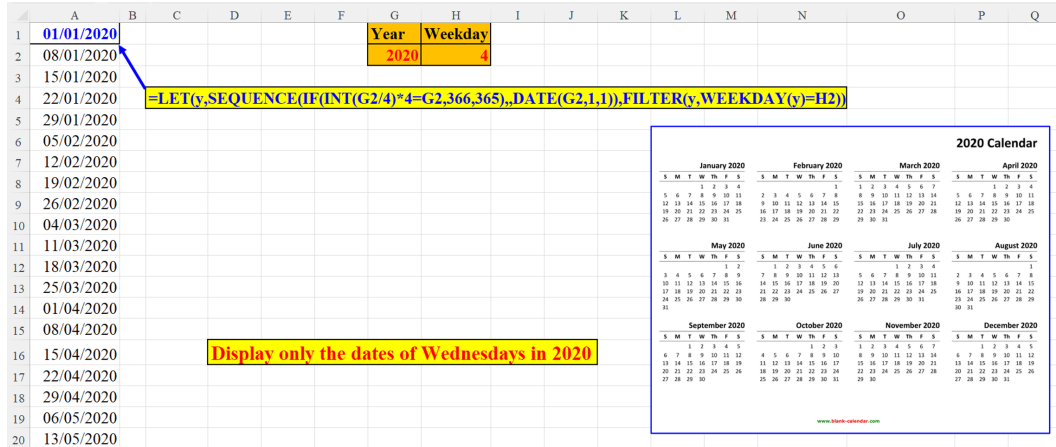


Figure 5.19: Display only dates of Wednesdays in 2020

How many Wednesdays are there in 2020?

In this section we will find the number of Wednesdays (parameter in cell H2) in the year 2020 (parameter in cell G2). The procedure to determine whether this is a leap year is slightly different from the one used in the previous section: *Display only dates of Wednesday in 2020* as shown in the following screenshot:

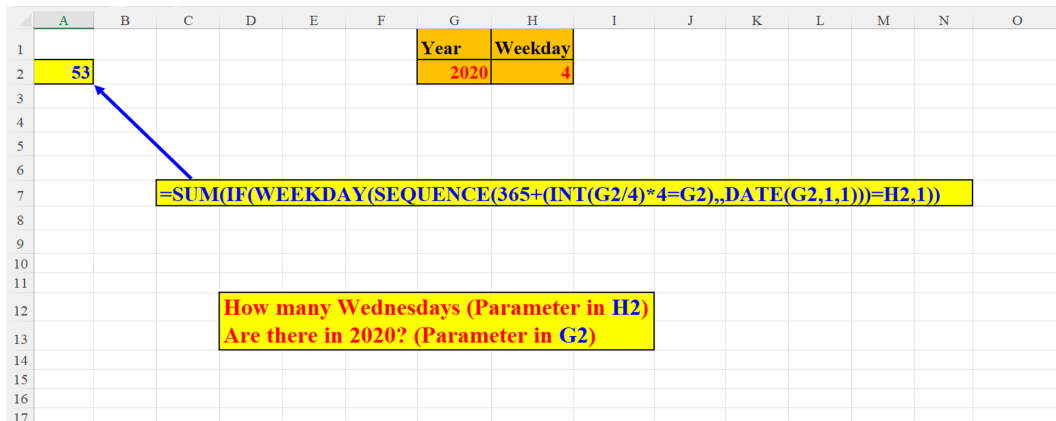


Figure 5.20: How many Wednesdays are there in 2020

Sequence of the month's last date for each month (two methods)

Method 1

The formula in this example uses the same trick as already shown in the preceding section: *The year's months with the last day of each month - three methods (method 1)*, if the month argument to the **DATE** function is current+1, then we can skip the **Day** argument, because the **DATE** function calculates the last day of the current month. **SEQUENCE**'s last argument is 0, so we start with the second month (2+0). Since the **Day** argument is missing, we get the date of January's last day as shown in the following screenshot:

	B	C	D	E	F	G	H	I	J	K
1	31/01/2022						Year	2022		
2	28/02/2022									
3	31/03/2022									
4	30/04/2022									
5	31/05/2022									
6	30/06/2022									
7	31/07/2022									
8	31/08/2022									
9	30/09/2022									
10	31/10/2022									
11	30/11/2022									
12	31/12/2022									
13										
14										
15	An array of dates: last date of each month of the year									
16										
17										

Figure 5.21: Sequence of the month's last date for each month (method 1)

Method 2

A similar approach to the one demonstrated in the previous subsection (as can be seen in *Figure 5.21*), but here, **SEQUENCE**'s last argument is not 0 but 1 (not explicitly specified) as shown in the following screenshot:

	B	C	D	E	F	G	H	I	J	
1	31/01/2022				Year	2022				
2	28/02/2022									
3	31/03/2022									
4	30/04/2022									
5	31/05/2022				Method 2					
6	30/06/2022									
7	31/07/2022									
8	31/08/2022	=DATE(G1,SEQUENCE(12,,2),)								
9	30/09/2022									
10	31/10/2022									
11	30/11/2022									
12	31/12/2022									
13										
14										
15	An array of dates: last date of each month of the year									
16										
17										
18										
19										

Figure 5.22: Sequence of the month's last date for each month (method 2)

A horizontal SEQUENCE of descending dates - First of each month

In this section, we will discuss the sequence of descending dates starting from 01/06/2020 (parameter defined in cell H2). The size of the sequence (six consecutive months) is set in cell I2. The minus sign preceding the **SEQUENCE** function *forces* the formula to generate a sequence of numbers in descending order starting with 0. Then, the **EDATE** adds this sequence to the starting date thus creating a sequence of dates in descending order.

To create a vertical sequence, all you have to do is switch places between **SEQUENCE**'s first two arguments:

SEQUENCE(, \$I\$2, 0) instead of **SEQUENCE(\$I\$2, , 0)**

	A	B	C	D	E	F	G	H	I	J
1	01/06/2020	01/05/2020	01/04/2020	01/03/2020	01/02/2020	01/01/2020		Date	How many months	
2								01/06/2020	6	
3										
4										
5										
6										
7										
8										
9										
10										
11										
12										
13										
14										
15										
16										
17										

Figure 5.23: Sequence of descending dates: 1st of each month

A substitute for NETWORKDAYS.INTL (for a certain month)

The following formula is my *replacement* for Excel's built-in **NETWORKDAYS.INTL** function. As the name suggests, it calculates the number of working days in a given period, excluding any combination of weekends (specified in cells D2:D3) and holidays (specified in cells J2:J10) [you can add as many holidays as you need in column J]. This function is very flexible: it can exclude either only the weekends or only the holidays, both or none. My solution takes the original idea one step further: it lists all the net working days for the specified month.

To make the formula shorter, we defined a Named Range *MonCal* as can be seen in *Figure 5.25*. The formula within that Named Range takes the month (parameter in cell G2) and generates the dates of all the month's days. This is a relatively simple formula. The main formula, however, is quite complicated and an explanation is due. It has three steps:

Step 1: Filter only dates which do not fall on the weekend days: neither on Friday (cell D6) nor on Saturday (cell D3).

Step 2: *Stacking* the filtered dates from *Step 1* with the list of holidays dates.

Step 3: Now that we have the holidays twice (one in *MonCal* and one from *Step 2*), the **UNIQUE** function removes those duplicate values to give us the desired result: list of days that are neither weekends nor holidays. The third argument of the **UNIQUE** function is: 1, which tells the function to return rows that occur exactly once in the array.

To visualize the result, we have added the month's calendar (in the middle of the following figure):

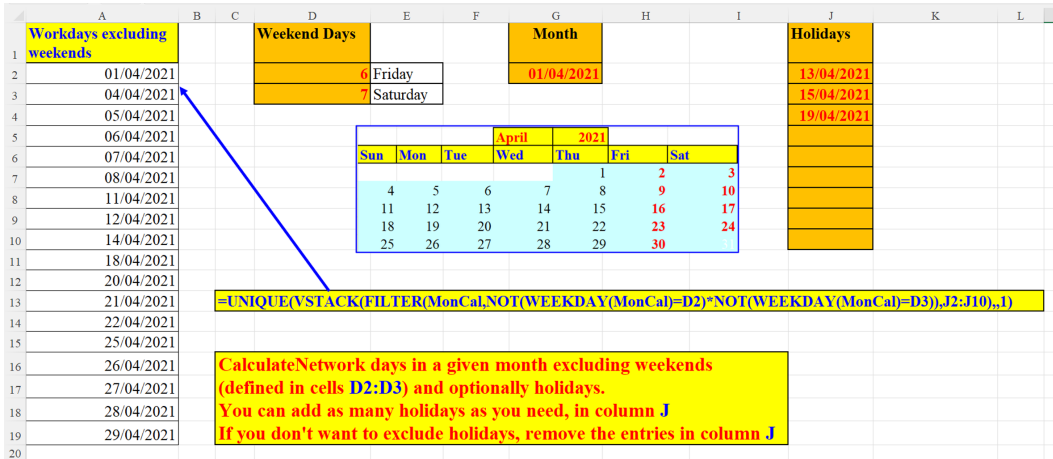


Figure 5.24: Instead of NETWORKDAYS.INTL (for a certain month)

The MonCal Named Range

The formula within the MonCal Named Range takes the month (parameter in cell G2) and generates the dates of all the month's days:

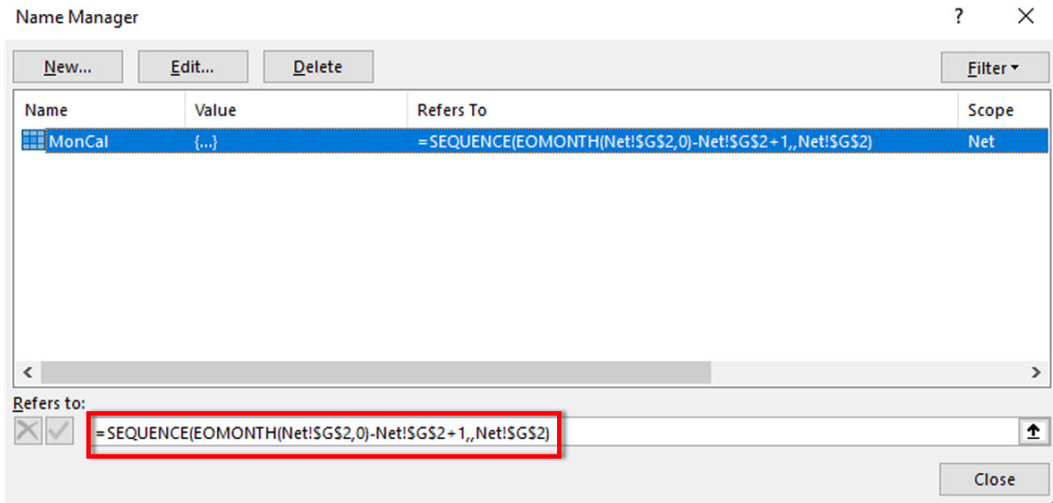


Figure 5.25: The MonCal Named Range "begets" the array of the month's dates

A substitute for the NETWORKDAYS.INTL (any period)

This is another version of the **NETWORKDAYS.INTL**, however here we have some differences when compared to the previous section:

Instead of an array of net working days, here, we calculate the number of net working days (the same as Excel's **NETWORKDAYS.INTL**).

The holidays parameter (column O) can be ignored altogether if cell J2 is set to 0 or left blank.

This formula can calculate any range of days, not necessarily one whole month.

The **SUM** function uses the name **wk** (defined by the **LET** function to shorten the formula), to extract only dates which do not fall on either Friday (cell G2) or Saturday (cell G3). Of course, you can define any weekend days. If you do not want to exclude any weekend days, just leave these cells empty (or with 0) as shown in the following screenshot:

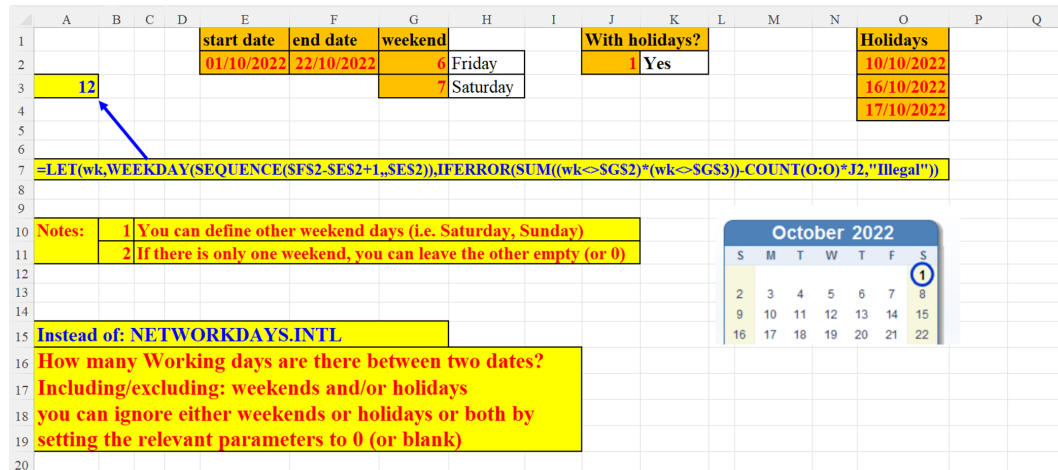


Figure 5.26: Instead of NETWORKDAYS.INTL (any period)

A substitute for NETWORKDAYS.INTL - with/without weekends

The last version of **NETWORKDAYS.INTL**'s replacement can exclude only weekends. These are defined in cells E2 and E3 and can be replaced or ignored completely, according to your needs. Here again, we employ a Named Range to keep the formula terse. As in the previous section, A substitute for **NETWORKDAYS.INTL** (for a certain month), the formula in it calculates the entire monthly calendar. As opposed to the

two previous versions, this formula displays both the array of working days for the chosen month (parameter in cell G2) along with the number of total working days as shown in the following screenshot:

	A	B	C	D	E	F	G	H	I	J	K
1		Workdays excluding weekends			Weekend Days		Month				
2		01/04/2021			6	Fri	01/04/2021				
3		04/04/2021			7	Sat					
4		05/04/2021									
5		06/04/2021									
6		07/04/2021	=FILTER(MonCal(WEEKDAY(MonCal)<=>\$E\$2)*(WEEKDAY(MonCal)<>\$E\$3))								
7		08/04/2021									
8		11/04/2021									
9		12/04/2021			Total work days		21				
10		13/04/2021									
11		14/04/2021					=COUNT(B:B)				
12		15/04/2021									
13		18/04/2021									
14		19/04/2021									
15		20/04/2021									
16		21/04/2021									
17		22/04/2021									
18		25/04/2021									
19		26/04/2021									
20		27/04/2021									
21		28/04/2021									
22		29/04/2021									

Figure 5.27: Instead of NETWORKDAYS.INTL (with/without weekends)

The Definition of MonCal

The formula within the MonCal Named Range takes the month (parameter in cell G2) and generates the dates of all the month's days as shown in the following screenshot:

Edit Name		?	×
Name:	MonCal		
Scope:	Sheet2		
Comment:			
Refers to:	=SEQUENCE(EOMONTH(Sheet2!\$G\$2,0)-Sheet2!\$G\$2+1,,Sheet2!\$G\$2)		
		OK	Cancel

Figure 5.28: Definition of MonCal

How many working days are there in each month of a given period?

This formula *imitates* the NETWORKDAYS . INTL function. It calculates the net workdays for each month in the month's parameter (cells: M1:M4) for the period stated in cells E2:F2. The weekend days parameter (cells G2:G3) can be modified, according to

your country's weekends. If you do not want to exclude any weekend days, just leave those cells empty (or with 0) as shown in the following screenshot:

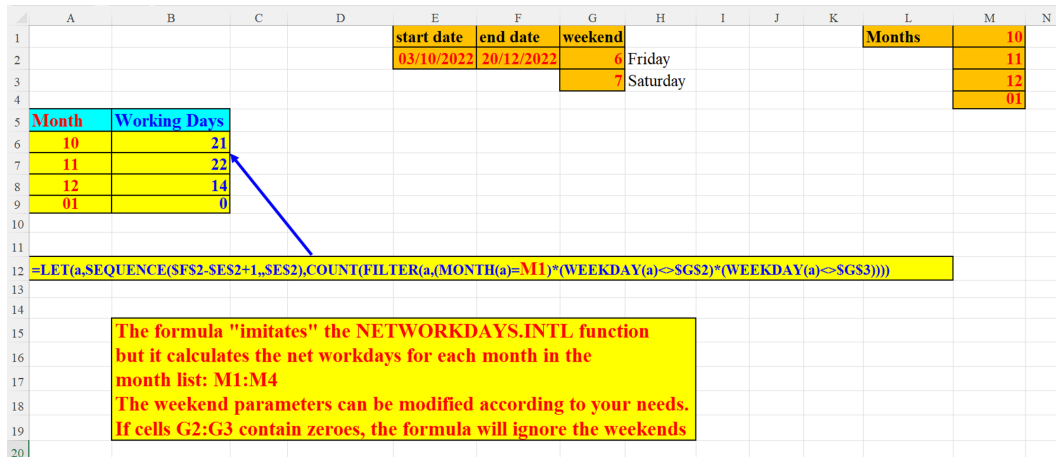


Figure 5.29: How many working days are there in each month of a given period

How many eligibility days?

This example demonstrates an imaginary social security office. Its customers are supposed to report every month on certain weekdays for them to be eligible for social security monthly allowances. The reporting period is from 01/08/2021 (cell B1) to 31/08/2021 (cell E1). In cells A4:A8 we have a list of customers. As stated above, each customer has certain days in a week in which he/she must report at the agency (cells B4:B8).

For example, Bridget has to report on Sundays and Mondays (1,2) whereas Catherine is supposed to show up at the agency every Sunday, Monday, Wednesday and Thursday (1,2,4,5) during the month.

In cells C4:C8 we calculate for each attendee the number of times they must report during the reporting period. So, for example, since August 2021 has five Sundays and five Mondays, Bridget must report 10 times in that month. Jessica, on the other hand, is required to arrive at the agency every day of the week, so she must show up 31 times in that month. A calendar of August 2021 is added (on the right-hand

side of the figure) to assist in checking the accuracy of the solution as shown in the following screenshot:

Reporting month	01/08/2021	End of Month	31/08/2021
Customer	Weekdays to report	total reporting days for current month	
Bridget	12	10	
Catherine	1245	18	
Angelina	25	9	
Jessica	1234567	31	
Julie	45	8	

August 2021

Week Su Mo Tu We Th Fr Sa

31 1 2 3 4 5 6 7

32 8 9 10 11 12 13 14

33 15 16 17 18 19 20 21

34 22 23 24 25 26 27 28

35 29 30 31

=SUM(N(WEEKDAY(SEQUENCE(DAY(\$E\$1))--(MID(\$B4,SEQUENCE(LEN(\$B4),1))))))

Each customer has certain days in a week where he/she has to report. For example: Bridget has to report on Sunday and Monday While Jessica must report on all weekdays, from Sunday to Saturday

Column C (total reporting days) calculates for each customer how many days he/she is supposed to report in this reporting month. Since Aug. 2021 has 5 Sundays and 5 Mondays, Bridget (for example) has to report 10 times in that month

Figure 5.30: How many eligibility days

The doctor's schedule (two versions)

The following two examples exhibit two settings of a schedule for a doctor's clinic. Viewing these two alternatives, each physician is free to choose their preferred timetable.

The doctor's schedule (version 1)

This is the first version of the doctor's schedule. We have two parameters: the opening hours of the clinic (cells J1:K1) and the time allotted per patient (cell E1). The formula builds the timetable accordingly (in Column A) as shown in the following screenshot:

Time gap allotted per patient	00:30	Office hours:	08:00	16:00
08:00				
08:30				
09:00				
09:30				
10:00				
10:30				
11:00				
11:30				
12:00				
12:30				
13:00				
13:30				
14:00				
14:30				
15:00				
15:30				
16:00				

= \$A\$1+SEQUENCE((K1-J1)/E1)*\$E\$1

The Doctor's Schedule

Figure 5.31: The doctor's schedule (version 1)

The doctor's schedule (version 2)

This is the second version of the doctor's schedule. Instead of defining the opening and closing hours, we offer a slightly different approach: We set the number of daily open hours (in cell D1), the time allotted per patient (in cell F1) and the opening hour of the clinic (in cell H1):

	A	B	C	D	E	F	G	H	I	J	K
1			No. of clinic hours	7	minutes per patient	45	opening time	9			
2											
3			9:00								
4			9:45								
5			10:30								
6			11:15								
7			12:00								
8			12:45								
9			13:30								
10			14:15								
11			15:00								
12			15:45								
13			16:30								
14											
15											
16											
17											
18											
19											
20											

A Dental Clinic's Schedule: 3 Parameters	
No. of daily open hours	D1
Time allotted per patient	F1
Clinic opens at	H1

The rule: A patient's treatment won't be interrupted even if the closing time has come
--

Figure 5.32: The doctor's schedule (version 2)

Monthly calendar – classic versus non-classic

Two monthly calendars: One - classic (Where the week starts on Sunday) and the second – Non-classic (where the week starts on the weekday of the month's first day). Both methods use the same formula, without any modification.

Monthly calendar - classic

This subsection illustrates the *Classic* monthly calendar: it starts on Sunday. The year (2017 parameter in cell E1) and the month (April chosen via data validation in cell E2) were selected for this illustration because April 2017 was a rare month: It had 6 weeks. Since this is a *classic* monthly calendar, the parameter in cell E3 is 1.

The calendar, as mentioned above, has only one formula (can be seen in cell A20). The other two formulae are set to create the weekday names (cells A11:G11) and generate the month name (cell D10).

The *trick* behind this calendar, along with the formula, is its Conditional Formatting technique, as can be seen in Figure 5.34.

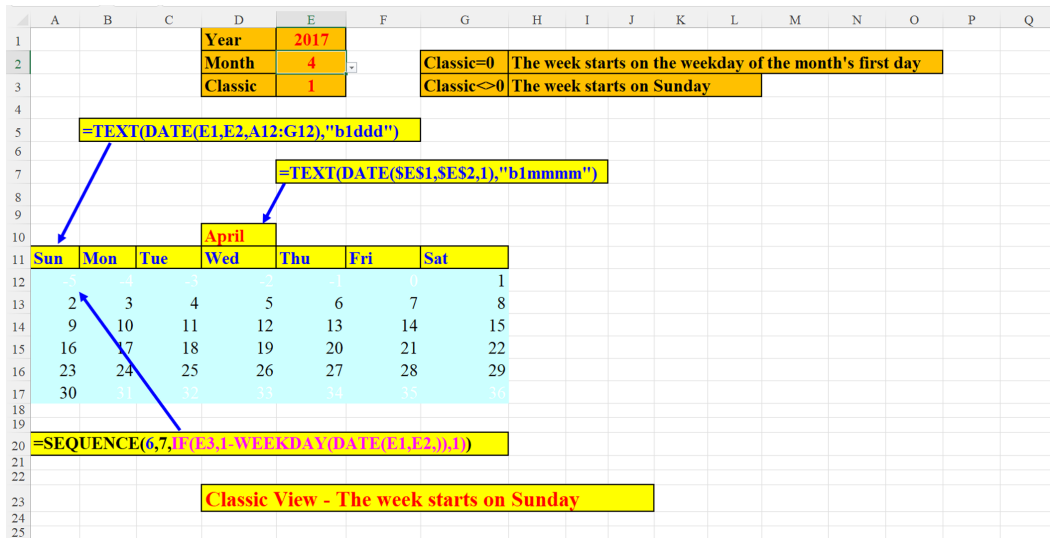


Figure 5.33: Monthly calendar – classic

The technique used in the following figure is the *trick* behind this calendar. Along with the formula, this Conditional Formatting technique, can be seen in Figure 5.34. The entire range of the calendar (cells A12:G17) has a light background color. The cells in row 12 preceding the 1st of April (in cell G12) contain numbers less than 1 and we want to hide them. The same applies to the cells succeeding the last month's day, April 30th (in cell A17): all these cells hold values greater than 30, so we want to hide them too. So, for the first case we have rule no.1: `=A12<1` and for second one we have rule no.2: `=A12>DAY(DATE(E1,E2+1,))`. These rules are formatted with a transparent font, so they are invisible:

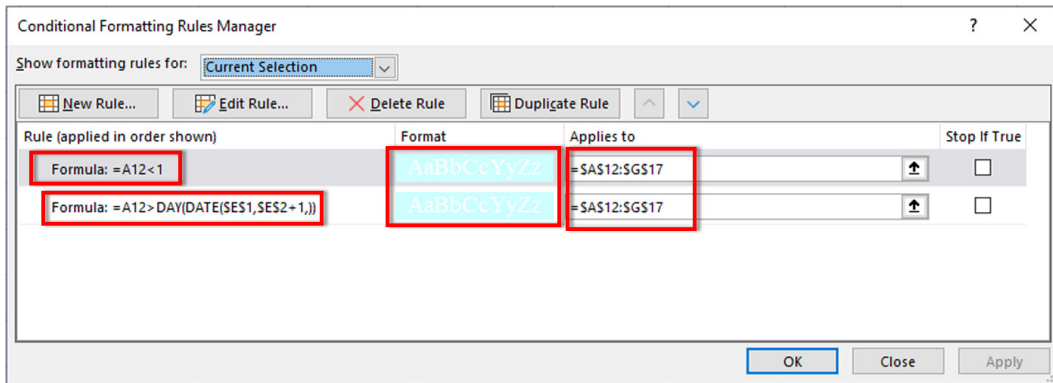


Figure 5.34: Monthly Calendar – Classic – Conditional Formatting

Monthly calendar – non-classic

The non-classic calendar is identical in all details but one to the classic calendar, the difference in the value defined in cell E3. Now, the parameter is *non-classic* which means: the week starts on the weekday of the month's first day.

This slight change has an impact in two places:

- The weekdays (cells A11:G11) begin on Saturday and not on Sunday.
- The calendar adapts itself automatically to the new situation: the calendar now has only five rows, and not six rows as in the *classic* calendar. (However, we already know that actually six rows are displayed. Thanks to the Conditional Formatting mechanism, the superfluous cells are hidden because their font is transparent) as shown in the following screenshot:

Year	2017	Month	4	Classic=0	The week starts on the weekday of the month's first day	
Classic	0	Classic<>0	The week starts on Sunday			
=TEXT(DATE(E1,E2,A12:G12),"b1ddd")						
=TEXT(DATE(SES1,SES2,1),"b1mmmm")						
April						
Sat	Sun	Mon	Tue	Wed	Thu	Fri
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	32	33	34	35
36	37	38	39	40	41	42
=SEQUENCE(6,7,0+(E3,1-WEEKDAY(DATE(E1,E2,0),1))						
Non-Classic View - The week starts on the weekday of the month's first day						

Figure 5.35: Monthly calendar – non-classic

Monthly calendar in 20 languages

In this section, we will learn how to display 20 calendars in 20 different languages.

First, there are three parameters:

- the year (in cell E1),
- the month (in cell E2) (a drop-down list whose values are defined in cells R2:R13), (see Figure 5.37), and
- the language (in cell E3) (a drop-down list whose values are taken from cells M2:M22, (see Figure 5.38).

Then, we have three formulae:

- which sets the weekdays names (cells A11:G11) according to the language chosen in cell E3. It searches the table defined in columns M:O by the language name and fetches the day format (from Column O) to be implemented in the weekday names.
- which sets the month name (cell D9) according to the language chosen in cell E3. It searches the table defined in columns M:P by the language name and fetches the day format for that language (from Column P) to be implemented in the month name.
- which populates the calendar array (cells A11:G15). In this example, we will use the *non-classic* monthly calendar:

A	B	C	D	E	F	G	H	I	J
			Year	2022					
			Month	9					
			Language	Greek					
=TEXT(WEEKDAY(DATE(SES1,SES2,COLUMN())),XLOOKUP(SES3,\$MS2:\$MS22,\$OS2:\$OS22))									
=TEXT(DATE(SES1,SES2,1),XLOOKUP(SES3,\$MS2:\$MS22,\$PS2:\$PS22))									
Σεπτέμβριος									
Πέμπτη	Παρασκευή	Σάββατο	Κυριακή	Δευτέρα	Τρίτη	Τετάρτη			
	1	2	3	4	5	6	7		
	8	9	10	11	12	13	14		
	15	16	17	18	19	20	21		
	22	23	24	25	26	27	28		
	29	30							
=SEQUENCE(ROUNDUP(DAY(DATE(SES1,ES2+1,)))/7,0),7,)									

Figure 5.36: Monthly calendar in 20 languages

Monthly calendar in 20 languages – list of languages and formats

The dataset introduced in the following screenshot represents the table used in the previous screenshot to fetch the language (defined in cell E3) by the **XLOOKUP** function. The language (in our case: Greek) is searched in cells M2:M22. **XLOOKUP** returns the month format (cells P2:P22) for the month name in Greek (cell D9) and for the weekday names (cells O2:O22) which will be displayed in cells A10:G10.

The formula to fetch the month name in Greek can be seen in cell E7, and the formula to return the Greek weekday names is displayed in cell A5 (See Figure 5.36).

	M	N	O	P
1	Language	Short Day Format	Long Day Format	Month Name Format
2	Italian	[\$-it-IT]ddd	[\$-it-IT]dddd	[\$-it-IT]mmmm
3	Arabic	[\$-ar-EG]ddd	[\$-ar-EG]dddd	[\$-ar-EG]mmmm
4	Hebrew	[\$-he-IL]ddd	[\$-he-IL]dddd	[\$-he-IL]mmmm
5	English (Br.)	[\$-en-BR]ddd	[\$-en-BR]dddd	[\$-en-BR]mmmm
6	English (US)	[\$-en-US]ddd	[\$-en-US]dddd	[\$-en-US]mmmm
7	German	[\$-de-DE]ddd	[\$-de-DE]dddd	[\$-de-DE]mmmm
8	French	[\$-fr-FR]ddd	[\$-fr-FR]dddd	[\$-fr-FR]mmmm
9	Greek	[\$-el-GR]ddd	[\$-el-GR]dddd	[\$-el-GR]mmmm
10	Chinese	[\$-zh-CN]ddd	[\$-zh-CN]dddd	[\$-zh-CN]mmmm
11	Japanese	[\$-ja-JP]ddd	[\$-ja-JP]dddd	[\$-ja-JP]mmmm
12	Hungarian	[\$-hu-HU]ddd	[\$-hu-HU]dddd	[\$-hu-HU]mmmm
13	Spanish	[\$-es-ES]ddd	[\$-es-ES]dddd	[\$-es-ES]mmmm
14	Portuguese	[\$-pt-PT]ddd	[\$-pt-PT]dddd	[\$-pt-PT]mmmm
15	Russian	[\$-ru-RU]ddd	[\$-ru-RU]dddd	[\$-ru-RU]mmmm
16	Czech	[\$-cs-CZ]ddd	[\$-cs-CZ]dddd	[\$-cs-CZ]mmmm
17	Dutch	[\$-nl-NL]ddd	[\$-nl-NL]dddd	[\$-nl-NL]mmmm
18	Turkish	[\$-tr-TR]ddd	[\$-tr-TR]dddd	[\$-tr-TR]mmmm
19	Bulgarian	[\$-bg-BG]ddd	[\$-bg-BG]dddd	[\$-bg-BG]mmmm
20	Danish	[\$-da-DK]ddd	[\$-da-DK]dddd	[\$-da-DK]mmmm
21	Hindi	[\$-hi-IN]ddd	[\$-hi-IN]dddd	[\$-hi-IN]mmmm
22	Polish	[\$-pl-PL]ddd	[\$-pl-PL]dddd	[\$-pl-PL]mmmm
23				

Figure 5.37: List of languages and their corresponding day- and month- formats

Monthly calendar in 20 languages – list of month numbers

This is the list of months used in the data validation of the preceding section: *Monthly calendar in 20 languages* (See Figure 5.36, cell E2).

	R
1	Months
2	1
3	2
4	3
5	4
6	5
7	6
8	7
9	8
10	9
11	10
12	11
13	12
14	

Figure 5.38: List of the month numbers (drop-down list to choose the month from)

Two methods for creating a list of the month's days

In this section, we will discuss two methods for creating a list of the month's days.

You should avoid the first one, which will yield wrong results (see *Figure 5.39*). It is better to use the second, dynamic one (see *Figure 5.40*). It assures us that we always get the correct number of days for the chosen month (February).

Monthly calendar – a bad attitude

This is an example of how not to create a dynamic monthly calendar: not all months have 31 days as can be seen in the following screenshot:

	A	B	C	D	E	F	G
1	01/02/2020				Date		
2	02/02/2020				01/02/2020		
3	03/02/2020						
4	04/02/2020						
5	05/02/2020						
6	06/02/2020						
7	07/02/2020						
8	08/02/2020		Don't do this	✘	=SEQUENCE(31,,E2)		
9	09/02/2020						
10	10/02/2020						
11	11/02/2020						
12	12/02/2020						
13	13/02/2020						
14	14/02/2020						
15	15/02/2020						
16	16/02/2020						
17	17/02/2020						
18	18/02/2020						
19	19/02/2020						
20	20/02/2020						
21	21/02/2020						
22	22/02/2020						
23	23/02/2020						
24	24/02/2020						
25	25/02/2020						
26	26/02/2020						
27	27/02/2020						
28	28/02/2020						
29	29/02/2020						
30	01/03/2020						
31	02/03/2020						

Figure 5.39: Monthly calendar – a bad attitude

Monthly calendar – a good attitude

This is a better approach to build a dynamic monthly calendar. The number of days in a given year and month can easily be found by using the **EOMONTH** function which

returns the desired month's last date. From that result, we extract the number of days, which is the first argument to the **SEQUENCE** function.

This way we can be sure that we have the correct number of days in the chosen month's calendar:

	A	B	C	D	E	F	G	H	I
1	01/02/2020				Date				
2	02/02/2020				01/02/2020				
3	03/02/2020								
4	04/02/2020								
5	05/02/2020								
6	06/02/2020								
7	07/02/2020								
8	08/02/2020	Do this	✓		=SEQUENCE(DAY(EOMONTH(E2,0)),E2)				
9	09/02/2020								
10	10/02/2020								
11	11/02/2020								
12	12/02/2020								
13	13/02/2020								
14	14/02/2020								
15	15/02/2020								
16	16/02/2020								
17	17/02/2020								
18	18/02/2020								
19	19/02/2020								
20	20/02/2020								
21	21/02/2020								
22	22/02/2020								
23	23/02/2020								
24	24/02/2020								
25	25/02/2020								
26	26/02/2020								
27	27/02/2020								
28	28/02/2020								
29	29/02/2020								

Figure 5.40: Monthly Calendar – a better attitude

Yearly calendar – good versus bad

Here again, we can see two attitudes to displaying the entire year's days. This time, the first one is better because it considers the fact that the chosen year (in cell E2) might be a leap year, a year in which there are 366 days and not 365:

	A	B	C	D	E	F	G	H	I	J	K
1	Yearly Calendar				Date						
2	01/01/2020				01/01/2020						
3	02/01/2020										
4	03/01/2020										
5	04/01/2020										
6	05/01/2020										
7	06/01/2020										
8	07/01/2020										
9	08/01/2020	Do this	✓		=SEQUENCE(365+(DAY(DATE(YEAR(E2),3,1))=29)*1,,E2)						
10	09/01/2020										
11	10/01/2020										
12	11/01/2020										
13	12/01/2020										
14	13/01/2020										
15	14/01/2020										
16	15/01/2020										
17	16/01/2020										
18	17/01/2020	Don't do this	✗		=SEQUENCE(365,,E2)						
19	18/01/2020										
20	19/01/2020										
21	20/01/2020										
22	21/01/2020										
23	22/01/2020										

Figure 5.41: Yearly calendar – good versus bad

Dynamic yearly calendar – in one formula

This yearly calendar is dynamic. It uses the “classic” method (see preceding section: *Monthly calendar – classic versus non-classic*, Figure 5.33). It accepts only one parameter (the year, in cell K1) and builds dynamically both the weekday names (in cells A1:G1) and the calendar itself (cells A2:G55) as shown in the following screenshot:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	Sun	Mon	Tue	Wed	Thu	Fri	Sat			Year	2022									
2								01/01/2022												
3	02/01/2022	03/01/2022	04/01/2022	05/01/2022	06/01/2022	07/01/2022	08/01/2022													
4	09/01/2022	10/01/2022	11/01/2022	12/01/2022	13/01/2022	14/01/2022	15/01/2022													
5	16/01/2022	17/01/2022	18/01/2022	19/01/2022	20/01/2022	21/01/2022	22/01/2022													
6	23/01/2022	24/01/2022	25/01/2022	26/01/2022	27/01/2022	28/01/2022	29/01/2022													
7	30/01/2022	31/01/2022	01/02/2022	02/02/2022	03/02/2022	04/02/2022	05/02/2022													
8	06/02/2022	07/02/2022	08/02/2022	09/02/2022	10/02/2022	11/02/2022	12/02/2022													
9	13/02/2022	14/02/2022	15/02/2022	16/02/2022	17/02/2022	18/02/2022	19/02/2022													
10	20/02/2022	21/02/2022	22/02/2022	23/02/2022	24/02/2022	25/02/2022	26/02/2022													
11	27/02/2022	28/02/2022	01/03/2022	02/03/2022	03/03/2022	04/03/2022	05/03/2022													
12	06/03/2022	07/03/2022	08/03/2022	09/03/2022	10/03/2022	11/03/2022	12/03/2022													
13	13/03/2022	14/03/2022	15/03/2022	16/03/2022	17/03/2022	18/03/2022	19/03/2022													
14	20/03/2022	21/03/2022	22/03/2022	23/03/2022	24/03/2022	25/03/2022	26/03/2022													
15	27/03/2022	28/03/2022	29/03/2022	30/03/2022	31/03/2022	01/04/2022	02/04/2022													
16	03/04/2022	04/04/2022	05/04/2022	06/04/2022	07/04/2022	08/04/2022	09/04/2022													
17	10/04/2022	11/04/2022	12/04/2022	13/04/2022	14/04/2022	15/04/2022	16/04/2022													
18	17/04/2022	18/04/2022	19/04/2022	20/04/2022	21/04/2022	22/04/2022	23/04/2022													
19	24/04/2022	25/04/2022	26/04/2022	27/04/2022	28/04/2022	29/04/2022	30/04/2022													
20	01/05/2022	02/05/2022	03/05/2022	04/05/2022	05/05/2022	06/05/2022	07/05/2022													
21	08/05/2022	09/05/2022	10/05/2022	11/05/2022	12/05/2022	13/05/2022	14/05/2022													
22	15/05/2022	16/05/2022	17/05/2022	18/05/2022	19/05/2022	20/05/2022	21/05/2022													

Figure 5.42: Dynamic yearly calendar - in one formula

Dynamic yearly calendar – Conditional Formatting

Similar to the *Monthly classic* calendar, this solution also uses Conditional Formatting with two rules (see Figure 5.43).

The first caters for cases where the year does not start on a Sunday.

The second takes care of instances where the year does not end on a Saturday.

As can be seen in Figure 5.43, these two rules use white (transparent) font color in such cases:

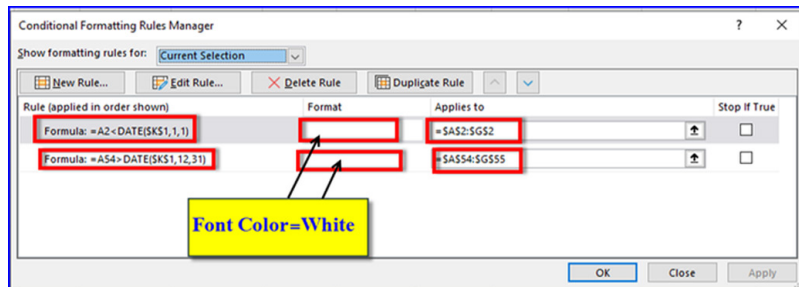


Figure 5.43: Dynamic yearly Calendar – Conditional Formatting

Dynamic yearly calendar - by month

The formula that builds the yearly calendar takes only one parameter: year (in cell O1).

The formula in cell A1 (shown in cell O3) propagates the month names in the 12 cells: A1:L1.

We build the dynamic calendar by writing the formula in cell A2 (displayed in cell O5) and then dragging it horizontally all the way to cell L2 as shown in the following screenshot:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
1	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec		year	2016							
2	1	1	1	1	1	1	1	1	1	1	1	1		A1	=TEXT(DATE(SEQUENCE(,12)), "b1mmm")							
3	2	2	2	2	2	2	2	2	2	2	2	2		A2	=SEQUENCE(DAY(DATE(SOSI,COLUMN()-1,)),1,)							
4	3	3	3	3	3	3	3	3	3	3	3	3										
5	4	4	4	4	4	4	4	4	4	4	4	4										
6	5	5	5	5	5	5	5	5	5	5	5	5										
7	6	6	6	6	6	6	6	6	6	6	6	6										
8	7	7	7	7	7	7	7	7	7	7	7	7										
9	8	8	8	8	8	8	8	8	8	8	8	8										
10	9	9	9	9	9	9	9	9	9	9	9	9										
11	10	10	10	10	10	10	10	10	10	10	10	10										
12	11	11	11	11	11	11	11	11	11	11	11	11										
13	12	12	12	12	12	12	12	12	12	12	12	12										
14	13	13	13	13	13	13	13	13	13	13	13	13										
15	14	14	14	14	14	14	14	14	14	14	14	14										
16	15	15	15	15	15	15	15	15	15	15	15	15										
17	16	16	16	16	16	16	16	16	16	16	16	16										
18	17	17	17	17	17	17	17	17	17	17	17	17										
19	18	18	18	18	18	18	18	18	18	18	18	18										
20	19	19	19	19	19	19	19	19	19	19	19	19										
21	20	20	20	20	20	20	20	20	20	20	20	20										
22	21	21	21	21	21	21	21	21	21	21	21	21										
23	22	22	22	22	22	22	22	22	22	22	22	22										
24	23	23	23	23	23	23	23	23	23	23	23	23										
25	24	24	24	24	24	24	24	24	24	24	24	24										
26	25	25	25	25	25	25	25	25	25	25	25	25										
27	26	26	26	26	26	26	26	26	26	26	26	26										
28	27	27	27	27	27	27	27	27	27	27	27	27										
29	28	28	28	28	28	28	28	28	28	28	28	28										
30	29	29	29	29	29	29	29	29	29	29	29	29										

Figure 5.44: Dynamic yearly calendar – by month

Dynamic yearly calendar - by week

This yearly calendar is arranged by week. In each column (weekday) only the dates that fall on that weekday are displayed. The year's parameter is defined in cell N1.

The formula creating the calendar is defined in cell A2. The *trick* here is that the formula in cell A1 (which generates the weekday names) can be activated only after the formula in cell A2 has been *launched*. Please note that this is a *non-classic* calendar, which means that the week does not always start on Sunday (please see preceding

section: *Monthly calendar – classic versus non-classic*, Figure 5.35) as shown in the following screenshot:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Wed	Thu	Fri	Sat	Sun	Mon	Tue						Year	2020			
2	01/01/2020	02/01/2020	03/01/2020	04/01/2020	05/01/2020	06/01/2020	07/01/2020										
3	08/01/2020	09/01/2020	10/01/2020	11/01/2020	12/01/2020	13/01/2020	14/01/2020										
4	15/01/2020	16/01/2020	17/01/2020	18/01/2020	19/01/2020	20/01/2020	21/01/2020										
5	22/01/2020	23/01/2020	24/01/2020	25/01/2020	26/01/2020	27/01/2020	28/01/2020										
6	29/01/2020	30/01/2020	31/01/2020	01/02/2020	02/02/2020	03/02/2020	04/02/2020										
7	05/02/2020	06/02/2020	07/02/2020	08/02/2020	09/02/2020	10/02/2020	11/02/2020										
8	12/02/2020	13/02/2020	14/02/2020	15/02/2020	16/02/2020	17/02/2020	18/02/2020										
9	19/02/2020	20/02/2020	21/02/2020	22/02/2020	23/02/2020	24/02/2020	25/02/2020										
10	26/02/2020	27/02/2020	28/02/2020	29/02/2020	01/03/2020	02/03/2020	03/03/2020										
11	04/03/2020	05/03/2020	06/03/2020	07/03/2020	08/03/2020	09/03/2020	10/03/2020										
12	11/03/2020	12/03/2020	13/03/2020	14/03/2020	15/03/2020	16/03/2020	17/03/2020										
13	18/03/2020	19/03/2020	20/03/2020	21/03/2020	22/03/2020	23/03/2020	24/03/2020										
14	25/03/2020	26/03/2020	27/03/2020	28/03/2020	29/03/2020	30/03/2020	31/03/2020										
15	01/04/2020	02/04/2020	03/04/2020	04/04/2020	05/04/2020	06/04/2020	07/04/2020										
16	08/04/2020	09/04/2020	10/04/2020	11/04/2020	12/04/2020	13/04/2020	14/04/2020										
17	15/04/2020	16/04/2020	17/04/2020	18/04/2020	19/04/2020	20/04/2020	21/04/2020										
18	22/04/2020	23/04/2020	24/04/2020	25/04/2020	26/04/2020	27/04/2020	28/04/2020										
19	29/04/2020	30/04/2020	01/05/2020	02/05/2020	03/05/2020	04/05/2020	05/05/2020										
20	06/05/2020	07/05/2020	08/05/2020	09/05/2020	10/05/2020	11/05/2020	12/05/2020										
21	13/05/2020	14/05/2020	15/05/2020	16/05/2020	17/05/2020	18/05/2020	19/05/2020										
22	20/05/2020	21/05/2020	22/05/2020	23/05/2020	24/05/2020	25/05/2020	26/05/2020										

Figure 5.45: Dynamic yearly calendar – by week

Yearly Horizontal calendar with highlighted weekday (two examples)

This is a horizontal calendar in only one formula, but with two Conditional Formatting rules. The first rule refers to the calendar itself (cells B4:AF15, see Figure 5.48) and *controls* the color of all the calendar's cells whose dates fall on the same day as the weekday specified in the parameter Weekday (cell L1).

The second Conditional Formatting rule is *responsible* to display (in column AH) the name of the weekday chosen in L1 and *colored* by the first rule. The color, as can be seen in the following two figures (Figure 5.46 and Figure 5.47) are the same, both for the highlighted dates in the calendar and the weekday name displayed in column AH. Another *trick* shown here is that each weekday name is displayed in a different cell: Sunday – in cell AH4, Monday – in cell H5.... and Saturday – in cell AH10.

The formula (in cell B4) creates a horizontal calendar, in which each month of the year parameter (in cell I1) *stretches* to the right according to the month's size (number of days). It is copied (without any alterations) to cells B5:B15.

The months names (in column A) are generated in a manner similar to the one shown in Figure 5.7: *Display month names without a specified date*

Yearly horizontal calendar

Example 1 (weekday chosen: Sunday)

In the following screenshot only Sundays (cell L1 is equal to 1) in the year 2022 are *painted*. The calendar is horizontal: Month names in A4:A15 and the dates in B4:AF15:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH										
1								year	2022	Weekday	1							Horizontal Calendar in one Formula																										
2																		The formula for each month is determined automatically																										
3																																												
4	Jan	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31			Sunday									
5	Feb	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28															
6	Mar	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31												
7	Apr	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30													
8	May	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31												
9	Jun	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30													
10	Jul	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31												
11	Aug	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31												
12	Sep	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30													
13	Oct	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31												
14	Nov	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30													
15	Dec	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31												
16																																												
17																																												
18																																												

Figure 5.46: Yearly horizontal calendar – example 1

Example 2 (weekday chosen: Saturday)

In the following screenshot only Saturdays (cell L1 is equal to 7) in the year 2022 are *painted*. The calendar is horizontal: Month names in A4:A15 and the dates in B4:AF15:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH											
1								year	2022	Weekday	7							Horizontal Calendar in one Formula																											
2																		The formula for each month is determined automatically																											
3																																													
4	Jan	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31													
5	Feb	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28																
6	Mar	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31													
7	Apr	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30														
8	May	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31													
9	Jun	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30														
10	Jul	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31					Saturday								
11	Aug	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31													
12	Sep	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30														
13	Oct	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31													
14	Nov	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30														
15	Dec	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31													
16																																													
17																																													
18																																													

Figure 5.47: Yearly horizontal calendar – example 2

Yearly horizontal calendar - Conditional Formatting - calendar

The following figure depicts the Conditional Formatting pane which exhibits the rules “behind” the weekday names, in *Figure 5.46* and *Figure 5.47*. These rules apply to the calendar (cells B4:AF15).

Each cell in the dataset of yearly weekdays (B4:AF15) is checked to see if it matches the weekday in the parameter (cell L1): Sunday – in *Figure 5.46* and Saturday - in *Figure 5.47*. If it does match, then that cell is *painted* accordingly by the colors defined in the format: magenta – for Sunday (marked with label 1, as shown in *Figure 5.48*), orange – for Monday (marked with label 2, as shown in *Figure 5.48*), red – for Tuesday (marked with label 3, as shown in *Figure 5.48*), and so on:

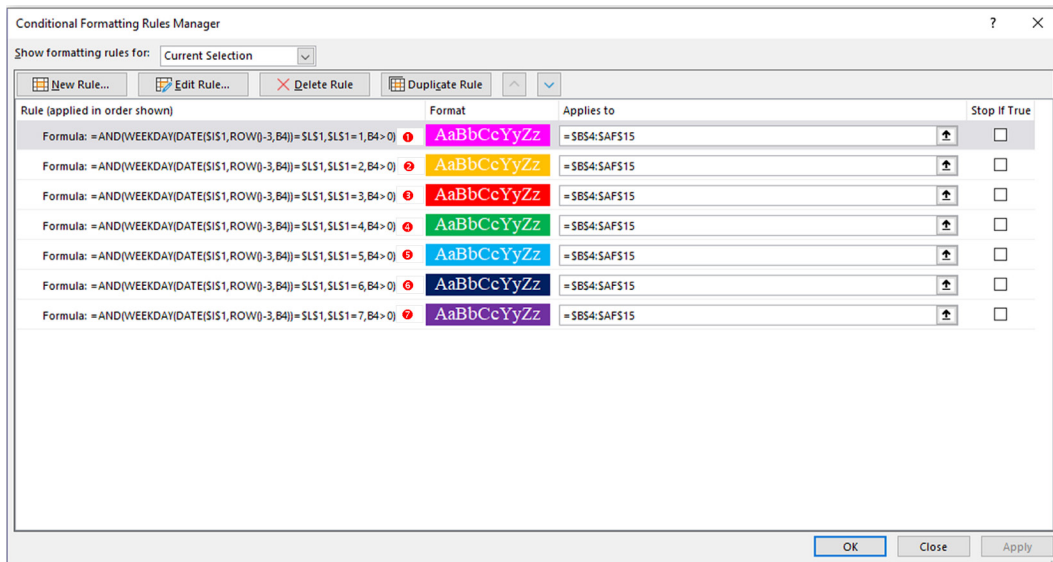


Figure 5.48: Yearly horizontal calendar – Conditional Formatting – calendar

Yearly horizontal calendar - Conditional Formatting - weekday names

The following figure depicts the Conditional Formatting pane which exhibits the rules *behind* the weekday names, in *Figure 5.46* and *Figure 5.47*. These rules apply to cells (AH4:AH10).

For each weekday chosen in L1, the corresponding weekday name is displayed in cells: AH4:AH10 and is *painted* accordingly by the colors defined in the format:

magenta – for Sunday (marked with label 1, as shown in *figure 5.49*), orange – for Monday (marked with label 2, as shown in *figure 5.49*), red – for Tuesday (marked with label 3, as shown in *Figure 5.49*), and so on. If cell L1 contains 1 (Sunday) then cell AH4 is selected and painted in magenta (marked with label 1, as shown in *Figure 5.49*); if cell L1 contains 2 (Monday) then cell AH5 is selected and painted in orange (marked with label 2, as shown in *Figure 5.49*), and so on. Cells (AN4:AN10) which cannot be seen on *Figure 5.46* and *Figure 5.47* contain the weekday names: Sunday to Saturday.

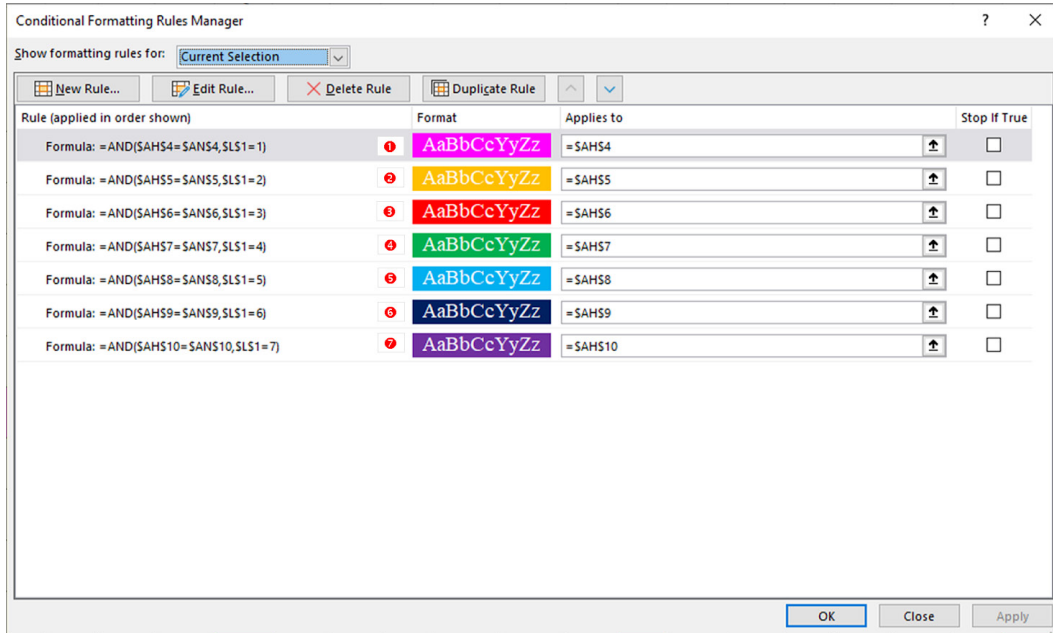


Figure 5.49: Yearly horizontal calendar – Conditional Formatting – weekday names

Yearly vertical calendar with highlighted weekday (2 examples)

The vertical calendar in this section is similar to the one in the previous section: Yearly horizontal calendar with highlighted weekday (two examples) (*Figure 5.46*).

The only significant difference is in the layout (vertical vs. horizontal) and, of course, in the formula. Here too, the formula is copied (vertically) from A4 to L4. Also, here we have two Conditional Formatting rules, one for the calendar (see *Figure 5.52*) and the second for the weekday names (see *Figure 5.53*).

Yearly vertical calendar – example 1

The calendar in the following screenshot is similar to the one in *Figure 5.46* but here it is vertical: Month names are in cells A3:L3. Only Sundays are painted since the weekday parameter (cell J1) is equal to 1:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1							year	2022	Weekday	1								
2																		
3	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec		Sunday				
4	1	1	1	1	1	1	1	1	1	1	1	1						
5	2	2	2	2	2	2	2	2	2	2	2	2						
6	3	3	3	3	3	3	3	3	3	3	3	3						
7	4	4	4	4	4	4	4	4	4	4	4	4						
8	5	5	5	5	5	5	5	5	5	5	5	5						
9	6	6	6	6	6	6	6	6	6	6	6	6						
10	7	7	7	7	7	7	7	7	7	7	7	7						
11	8	8	8	8	8	8	8	8	8	8	8	8						
12	9	9	9	9	9	9	9	9	9	9	9	9						
13	10	10	10	10	10	10	10	10	10	10	10	10						
14	11	11	11	11	11	11	11	11	11	11	11	11		=SEQUENCE(DAY(DATE(SH\$1,COLUMN()+L)),L)				
15	12	12	12	12	12	12	12	12	12	12	12	12						
16	13	13	13	13	13	13	13	13	13	13	13	13						
17	14	14	14	14	14	14	14	14	14	14	14	14						
18	15	15	15	15	15	15	15	15	15	15	15	15		Vertical Yearly Calendar				
19	16	16	16	16	16	16	16	16	16	16	16	16						
20	17	17	17	17	17	17	17	17	17	17	17	17						
21	18	18	18	18	18	18	18	18	18	18	18	18						
22	19	19	19	19	19	19	19	19	19	19	19	19						
23	20	20	20	20	20	20	20	20	20	20	20	20						

Figure 5.50: Yearly vertical calendar – example 1

Yearly vertical calendar – example 2

The calendar in the following screenshot is similar to the one in *Figure 5.47* but here it is vertical: Month names are in cells A3:L3. Only Saturdays are painted since the weekday parameter (cell J1) is equal to 7:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1							year	2022	Weekday	7								
2																		
3	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec						
4	1	1	1	1	1	1	1	1	1	1	1	1						
5	2	2	2	2	2	2	2	2	2	2	2	2						
6	3	3	3	3	3	3	3	3	3	3	3	3						
7	4	4	4	4	4	4	4	4	4	4	4	4						
8	5	5	5	5	5	5	5	5	5	5	5	5						
9	6	6	6	6	6	6	6	6	6	6	6	6		Saturday				
10	7	7	7	7	7	7	7	7	7	7	7	7						
11	8	8	8	8	8	8	8	8	8	8	8	8						
12	9	9	9	9	9	9	9	9	9	9	9	9						
13	10	10	10	10	10	10	10	10	10	10	10	10						
14	11	11	11	11	11	11	11	11	11	11	11	11		=SEQUENCE(DAY(DATE(SH\$1,COLUMN()+1)),L)				
15	12	12	12	12	12	12	12	12	12	12	12	12						
16	13	13	13	13	13	13	13	13	13	13	13	13						
17	14	14	14	14	14	14	14	14	14	14	14	14						
18	15	15	15	15	15	15	15	15	15	15	15	15		Vertical Yearly Calendar				
19	16	16	16	16	16	16	16	16	16	16	16	16						
20	17	17	17	17	17	17	17	17	17	17	17	17						
21	18	18	18	18	18	18	18	18	18	18	18	18						
22	19	19	19	19	19	19	19	19	19	19	19	19						
23	20	20	20	20	20	20	20	20	20	20	20	20						

Figure 5.51: Yearly vertical calendar – example 2

Yearly vertical calendar – Conditional Formatting – calendar

The following figure depicts the Conditional Formatting pane which exhibits the rules “behind” *Figure 5.50* and *Figure 5.51*. These rules apply to the vertical calendar (cells A4:L34)

Each cell in the dataset of yearly weekdays (A4:L34) is checked to see if it matches the weekday in the parameter (cell J1): Sunday – in *Figure 5.50* and Saturday - in *Figure 5.51*. If it does match, then that cell is “painted” accordingly by the colors defined in the format: magenta – for Sunday (marked with label 1, as shown in *figure 5.52*), orange – for Monday (marked with label 2, as shown in *figure 5.52*), red – for Tuesday etc.

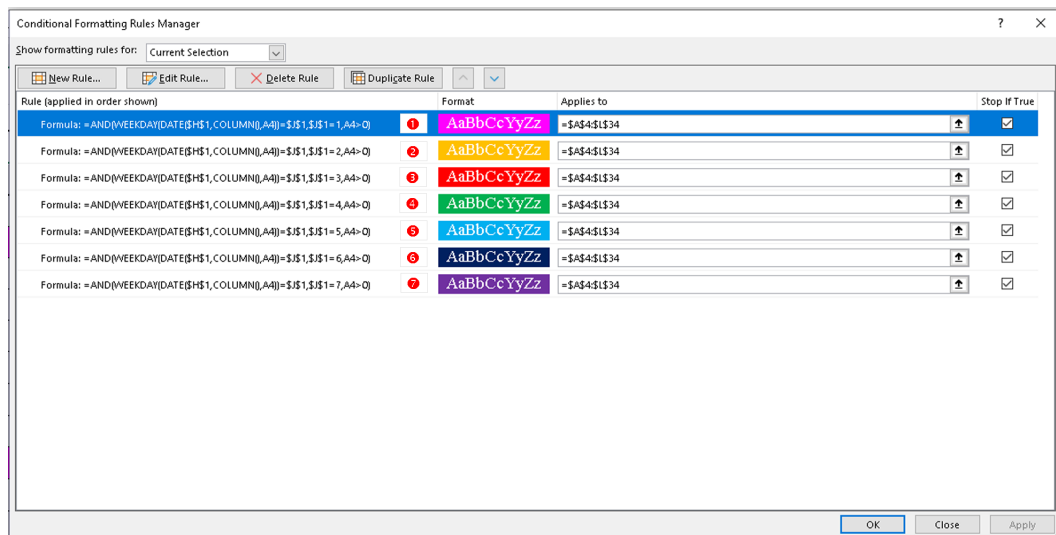


Figure 5.52: Yearly vertical calendar – Conditional Formatting – calendar

Yearly vertical calendar – Conditional Formatting – weekday names

The following screenshot (*Figure 5.53*) depicts the Conditional Formatting pane which exhibits the rules “behind” the weekday names, in *Figure 5.50* and *Figure 5.51* (cells N3 and N9, respectively). These rules apply to the weekday names (cells N3:N9)

For each weekday chosen in J1, the corresponding weekday name is displayed in cells N3:N9 and is painted accordingly with the same color of that weekday in the calendar (cells A4:L34): magenta – for Sunday (marked with label 1, as shown in *figure 5.53*), orange – for Monday (marked with label 2, as shown in *figure 5.53*), red – for Tuesday etc.

Cells (AC2:AC8) which cannot be seen on *Figure 5.50* and *Figure 5.51* contain the weekday names: Sunday to Saturday.

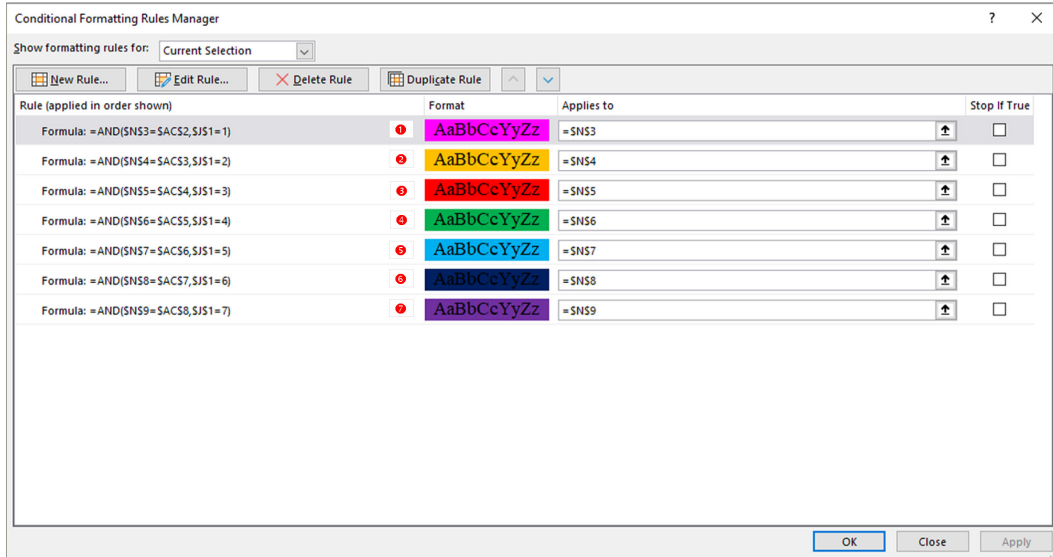


Figure 5.53: Yearly vertical calendar – Conditional Formatting – weekday names

Yearly calendar: one formula with Conditional Formatting

This is the last yearly calendar with one formula. Here we have only one Conditional Formatting rule, but since we have only the YEAR parameter (in cell H1) and no weekday to select, the calendar is formatted with each weekday having its own color (the color legend is displayed in cells N2:O8).

NOTE: If you feel that the calendar is too cluttered, you can change the Conditional Formatting rule (for example, to ignore Fridays and Saturdays).

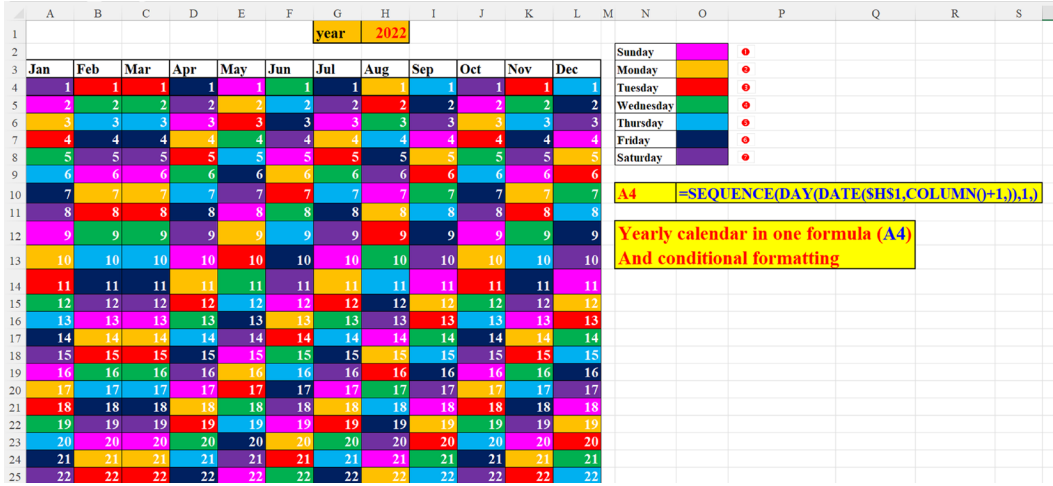


Figure 5.54: Yearly Calendar – One formula with Conditional Formatting

Conditional Formatting – each weekday is formatted differently

The following screenshot depicts the Conditional Formatting rules for Figure 5.54. Each cell in the calendar (A4:L34) has a different color: Sunday is painted in magenta (as represented by label 1 in Figure 5.55), Monday is painted in orange (as represented by label 2 in Figure 5.55), Tuesday is painted in red (as represented by label 3 in Figure 5.55). These, of course, correspond to the colors associated with the weekday names in the previous figure: Figure 5.54: Yearly Calendar - One formula with Conditional Formatting (cells N2:P8).

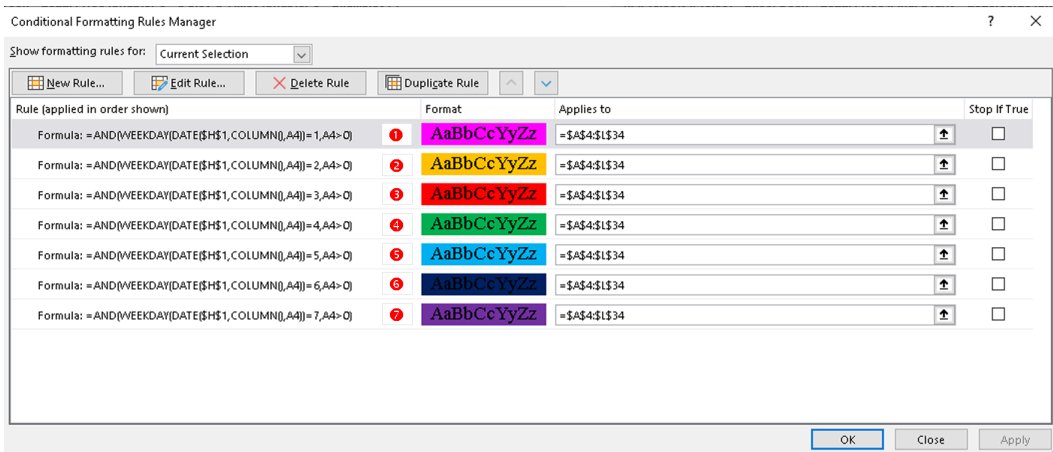


Figure 5.55: Conditional Formatting – each weekday is formatted differently

Conclusion

In this chapter, we have showcased **SEQUENCE** in more than 30 examples. This function has been used in various instances, on any scale of time: from second, minute, hour to week, month, and yearly operations. **SEQUENCE** is indispensable in this context. We demonstrated how to calculate time gaps (seconds, minutes, and so on), how to set timetables, how to compute net working days in a period of time with/without weekends and/or holidays, how to create a dynamic sequence of dates, how to build classic and non-classic monthly calendars, how to construct vertical and horizontal yearly calendars with/without formatting the weekdays, how to assemble presence tables, how to find out the number of a certain weekday in a particular period of time, and how to display your monthly calendar in other languages and many more.

In the next chapter, we will discuss some very interesting solutions related to financial challenges. We will see how the **SEQUENCE** function is requisite in shortening the process of solving fiscal problems.

Points to remember

- **SEQUENCE** is ideal when constructing calendars, either monthly or yearly. Clever use of Conditional Formatting makes your calendar both flexible and outstanding.
- By using **SEQUENCE**, you can easily set up a perfect substitution for the NETWORKDAYS.INTL both in scope and in flexibility.
- **SEQUENCE** easily generates the list of the first or last day of the months of a given year.
- **SEQUENCE** is phenomenal in setting timetables based on fixed gaps of time.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



CHAPTER 6

Financial Operations with SEQUENCE

Introduction

This chapter manifests the use of **SEQUENCE** in core financial functions. The implementation of this function in traditional fiscal functions is nothing short of a revolution. Now, one can quickly review several solutions at once. No need to code many formulae to view multiple results. Some finely chosen examples of business-oriented functions illustrate the advantages of using **SEQUENCE** in functions that hitherto were implemented without arrays, neither as input nor as output. These functions are the standard financial functions of Excel, used throughout the world of finance, especially in loans and investments.

Structure

In this chapter, we will discuss the following topics:

- Examples of **SEQUENCE** with financial functions
 - Loan return by payments per period
 - PMT - Periodic payment of a loan – traditional method
 - Periodic payment of a loan – a more flexible method
 - The Depreciation function in Excel – DB

- Equally divide a sum of money over a period of time
- One formula: How varying loan amounts impact the loan's installments
- NPV – No need for a Data table
- PDURATION - Multiple results
- The RATE function – multiple results
- RRI - calculate the average annual interest rate of an investment
- SEQUENCE and SUM

Objectives

The goal of this chapter is to give you, the reader of this book, some ideas on how to use “traditional” financial functions in a manner that will save you time as well as display several optional solutions simultaneously. This will make it easier for you to determine which is the scenario that best suits your needs, instead of wasting time running multiple simulations with a single argument and solution.

Examples of SEQUENCE with financial functions

The following sections manifest the use of **SEQUENCE** in core financial functions.

Loan return by payments per period

We need to return a loan, but cannot decide how to *spread* the installments over a period of time. The number of payments per period is defined in cell E1. The larger this number, the smaller the number of periods. In principle, it does not really matter whether the period is a month, a quarter, or a year. We just want to know the total amount of periods given the number of installments (defined in A4:A8) and the parameter in E1.

So, for example, if we are supposed to return 18 installments (cell A4) where, in each period we return six installments, then the total period to return that loan would be three periods (result in G4). If we have only nine installments, (cell A6) then we need two periods (G6), to return six payments in in the first period and three payments in the second period. This is illustrated in the following figure:

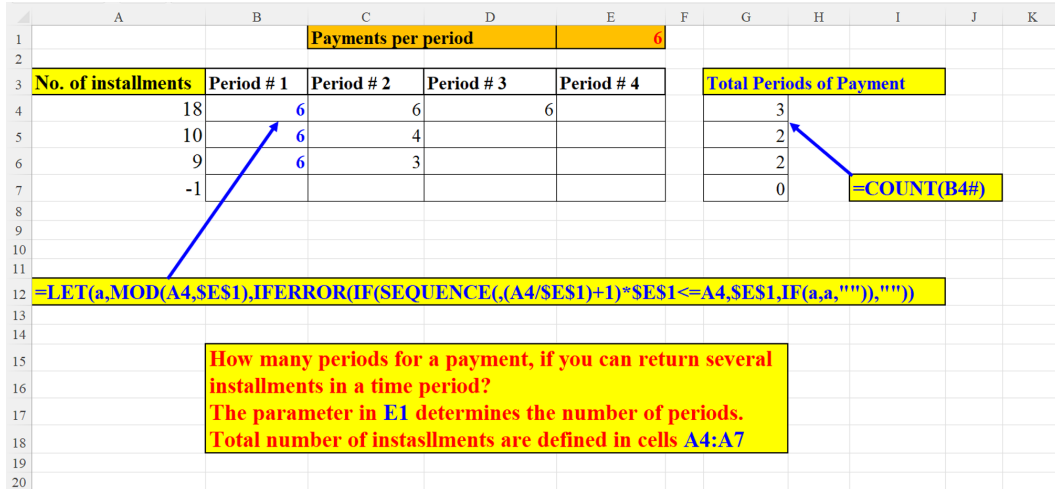


Figure 6.1: Loan Return by payments per period

PMT - Periodic payment of a loan – traditional method

When returning a loan, we want to know the amount of money we are supposed to pay on each installment, whether we return the money monthly, quarterly, or yearly. So far, we could calculate the PMT (periodic payment for constant payments and constant interest rate) for only one amount or only one interest rate, as explained in the example.

In the following example, we will demonstrate the monthly payment due for a fixed loan amount (PV in cell H2) and a fixed interest rate (4.40% = cell F2 + cell I2) for a period of 12 months (nper=number of periods, in cell G2). Since this is the traditional method, we get only result (in cell A2). To make the calculation more flexible, we added 2 spin buttons, one for the initial rate (in cell D2) and the second for the incremental rate (in cell K2). Cells E2 and J2, respectively, are helper cells, since the spin buttons can handle only integers, not fractions. F2 is the initial rate (4%). We divide cell E2 by 100 and cell I2 is the incremental rate (0.4%). We divide cell J2 by

1000. The first argument to the formula (B2) is the monthly rate ($=4.40\%/12$) since we return the loan monthly as shown in the following screenshot:

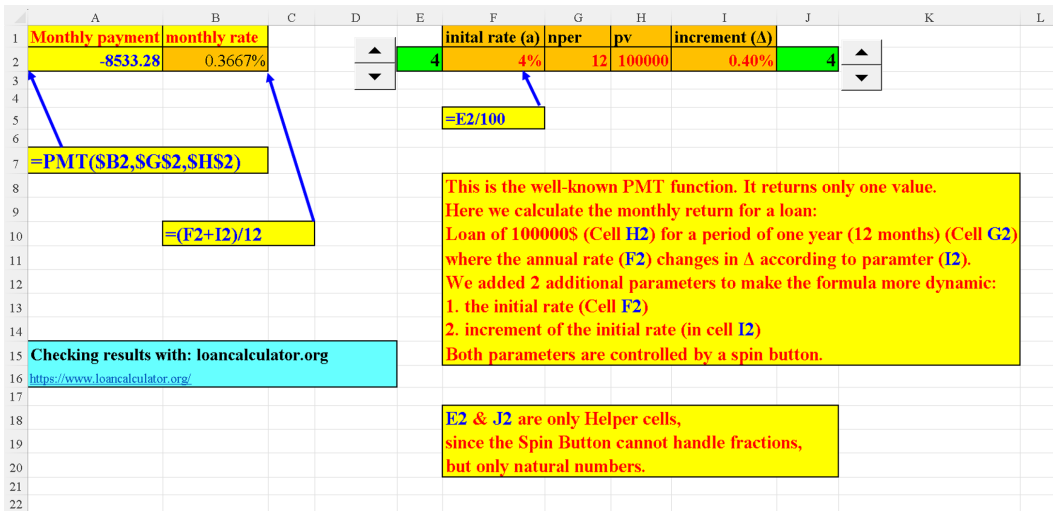


Figure 6.2: Monthly payment of a loan – “traditional” method

Now we can compare our calculation to the result in www.loancalculator.org. As can be clearly seen, our result in Figure 6.2 and the site’s result (in Figure 6.3) are identical:

Loan Calculator

The web's original easy-to-use JavaScript loan calculator
Automatically calculates your monthly loan payments

Loan Details	Amount
Loan Amount \$	100000
Loan Term Years	1
Interest Rate (APR %)	4.4
Payments Per Year Monthly	12
<input type="button" value="Reset"/>	<input type="button" value="Calculate"/>

Loan Payments	Amount
Interest Only Payment	\$366.67
Amortizing Payments	\$8533.28

Figure 6.3: Monthly payment of a loan – comparing the result with Loan Calculator site

In the following figure, we can see that the **Spin** button for the initial rate is linked to cell E2. Since, as stated previously, the **Spin** button cannot display fractions, the desired interest rate (4%) is achieved in cell F2 by dividing E2 by 100 as shown in the following screenshot:

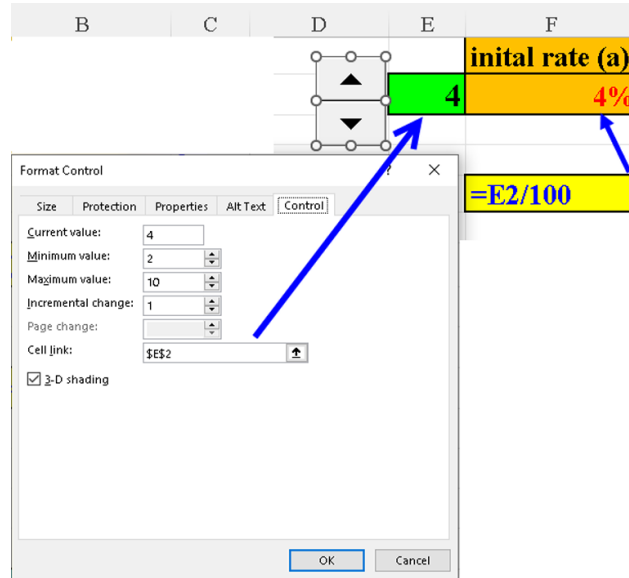


Figure 6.4: Monthly payment of a loan – Spin Button – Initial Rate

The following figure shows the **Spin** Button for the incremental rate. It is linked to cell J2. Since, as stated previously, the **Spin** button cannot display fractions, the desired incremental rate (0.4%) is achieved in cell I2 by dividing J2 by 1000 as shown in the following screenshot:

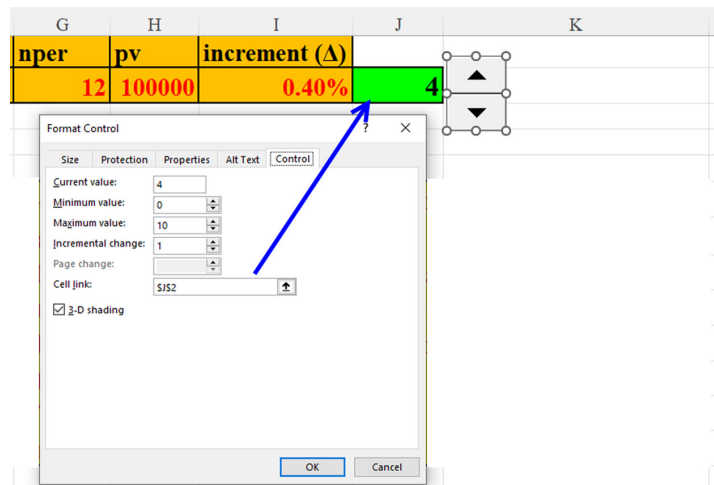


Figure 6.5: Monthly payment of a loan – Spin Button – Incremental Rate

Periodic payment of a loan – a more flexible method

As we have seen in the previous section, the traditional PMT function returns only one value. However, if we want to examine the PMT [=periodic payment] for several interest rates, we hereby suggest a more flexible method. The idea is that you can play with one or more arguments of this function, for example, the interest rate. Instead of feeding the formula with only one value, we can provide it with an array of value thanks to the **SEQUENCE** function. As can be seen in *Figure 6.6*, we have an array of 12 monthly interest rates (cells B2:B13), starting from 4% (parameter in F2) growing by the increment defined in I2. The **SEQUENCE** function within the PMT formula: **SEQUENCE(F5, F2, I2)/12** - creates the array shown in B2:B13 and enables us to generate a simulation of 12 monthly payments (as can be seen in the result's array, cells A2:A13) for 12 different monthly rates, as opposed to just one payment in Section, *Periodic payment of a loan – traditional method* above.

The same method of **Spin** buttons is used here, exactly as in our previous example (see *Figure 6.4* and *Figure 6.5*), so there is no need to display them again.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Monthly payment	monthly rate				initial rate	nper	pv	increment (Δ)				
2	-8514.99	0.3333%			4	4%	12	100000	0.40%	4			
3	-8533.28	0.3667%											
4	-8551.59	0.4000%				No. of simulations							
5	-8569.92	0.4333%				12							
6	-8588.27	0.4667%											
7	-8606.64	0.5000%											
8	-8625.04	0.5333%											
9	-8643.46	0.5667%											
10	-8661.90	0.6000%											
11	-8680.36	0.6333%											
12	-8698.84	0.6667%											
13	-8717.35	0.7000%											
14													
15	=PMT(SEQUENCE(F5,F2,I2)/12,G2,H2)												
16	Checking results with: calculator.net												
18	https://www.calculator.net/												

Using SEQUENCE to simulate loan installments:
 The usual PMT returns only one value (see previous example)
 Now you can "play" with the arguments and return many results by dynamically changing, for example, the yearly interest rate.
 We check the monthly return for a 12-month (G2) 100000\$ loan (H2) by changing the annual rate (F2) in Δ according to parameter (I2): 4%, 4.4%, 4.8%.....8.4%
 The annual rates are converted into monthly rates (in cells B2:B13) because the payments are monthly.
 Both F2 & I2 are controlled by spin buttons (in D2,K2 respectively).
 E2 & J2 are only Helper cells, since the Spin Buttons cannot handle fractions, but only natural numbers.

Figure 6.6: Periodic payment of a loan – a more flexible method

On comparing our calculation to the result in www.loancalculator.org, we can clearly see that the result (in *Figure 6.6*) (cell A2) and the site's result (in *Figure 6.7*) are identical:

Loan Calculator

Amortized Loan: Paying Back a Fixed Amount Periodically

Use this calculator for basic calculations of common loan types such as [mortgages](#), [auto loans](#), [student loans](#), or [personal loans](#), or click the links for more detail on each.


Loan Amount	\$100000	Results: Payment Every Month \$8,514.99 Total of 12 Payments \$102,179.89 Total Interest \$2,179.89 View Amortization Table
Loan Term	1 years	
	0 months	
Interest Rate	4 %	
Compound	Monthly (APR) ▾	
Pay Back	Every Month ▾	
Calculate ▶		

Figure 6.7: Periodic payment of a loan – compare result in cell A2 (Figure 6.6) with Loan Calculator site

Comparing our calculation to the result in www.loancalculator.org. As can be clearly seen, our result (in Figure 6.6) above (in cell A3) and the site's result (in Figure 6.8) are identical:

Loan Calculator

Amortized Loan: Paying Back a Fixed Amount Periodically

Use this calculator for basic calculations of common loan types such as [mortgages](#), [auto loans](#), [student loans](#), or [personal loans](#), or click the links for more detail on each.


Loan Amount	\$100000	Results: Payment Every Month \$8,533.28 Total of 12 Payments \$102,399.32 Total Interest \$2,399.32 View Amortization Table
Loan Term	1 years	
	0 months	
Interest Rate	4.4 %	
Compound	Monthly (APR) ▾	
Pay Back	Every Month ▾	
Calculate ▶		

Figure 6.8: Periodic payment of a loan – compare result in cell A3 (Figure 6.6) with Loan Calculator site

The following figure (Figure 6.9) shows the same result as in cell A13 (Figure 6.6). It is displayed to compare our results with the site's results:

Loan Calculator

Amortized Loan: Paying Back a Fixed Amount Periodically

Use this calculator for basic calculations of common loan types such as [mortgages](#), [auto loans](#), [student loans](#), or [personal loans](#), or click the links for more detail on each.


Loan Amount	\$100000	Results:	
Loan Term	1 years		Payment Every Month \$8,717.35
	0 months		Total of 12 Payments \$104,608.18
Interest Rate	8.4 %		Total Interest \$4,608.18
Compound	Monthly (APR)		View Amortization Table
Pay Back	Every Month		
Calculate			

Figure 6.9: Periodic payment of a loan – compare result in cell A13 (Figure 6.6) with Loan Calculator site

The Depreciation function in Excel – DB

Depreciation (do not mix it with depreciation) is the inevitable process by which an asset gradually reduces its value as time goes by. The original method requires a formula for each period (as can be seen in the upper part of Figure 6.10).

We propose a simpler solution where one formula (in cell C9) replaces five formulae. If the depreciation period is longer, then of course the advantage of this technique is even more conspicuous as shown in the following screenshot:

	A	B	C	D	E	F	G	H
1	Initial Cost	200,000						
2	Salvage value	25,000						
3	Life (depreciation period)	5	Year 1	Year 2	Year 3	Year 4	Year 5	
4	Period	1,2,3,4,5	1	2	3	4	5	
5								
6	Depreciation amount calculated per year	\$ 68,000.00	\$ 44,880.00	\$ 29,620.80	\$ 19,549.73	\$ 12,902.82		
7		=DB(B1,B2,B3,C4)	=DB(B1,B2,B3,D4)	=DB(B1,B2,B3,E4)	=DB(B1,B2,B3,F4)	=DB(B1,B2,B3,G4)		
8								
9	Depreciation amount calculated at once in one formula for all 5 years	\$ 68,000.00	Year 1					
10		\$ 44,880.00	Year 2					
11		\$ 29,620.80	Year 3					
12		\$ 19,549.73	Year 4					
13		\$ 12,902.82	Year 5					
14								
15			=DB(B1,B2,B3,SEQUENCE(5))					
16								
17								
18								
19								
20								
21								
22								

Calculating the declining-balance depreciation of an asset
The depreciation (DB) calculated separately per each year (5 formulae C6:G6) can be achieved with only one formula instead of 5 (cell C9)

Figure 6.10: The Depreciation function – one formula instead of five

Equally divide a sum of money over a period of time

The following figure indicates a simple procedure to equally divide a certain amount of money (defined in C2) into a defined number of months (in cell B2) starting from a certain date (visible in cell A2). The period's dates are constructed dynamically in row 1 (cells D1:O1) and the amounts per month are displayed in row 2 (cells D2:O2). The number of months defined in cell B2 cannot exceed 12:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Start Date	Months	Total	10-22	11-22	12-22	01-23	02-23	03-23	04-23	05-23	06-23	07-23	08-23	09-23	
2	13/10/2022	12	20400	1700.00	1700.00	1700.00	1700.00	1700.00	1700.00	1700.00	1700.00	1700.00	1700.00	1700.00	1700.00	
3																
4																
5																
6																
7																
8																
9																
10																
11																
12																
13																
14																
15																
16																
17																
18																
19																
20																
21																

Figure 6.11 shows a spreadsheet illustrating how to equally divide a sum of money over a given period of time. The spreadsheet has columns A through P and rows 1 through 21. Row 1 contains dates from 10-22 to 09-23. Row 2 contains monthly payments of 1700.00. Cell B2 contains 12, C2 contains 20400, and D2 contains 1700.00. Two formulas are shown: =SEQUENCE(,B2,SC\$2/SBS2,0) in cell D7 and =SEQUENCE(,B2,(A2),30) in cell E8. A text box explains: 'We want to equally divide the sum in C2 into B2 months Starting from the date in A2. A2, B2 and C2 are the 3 parameters to execute this task'.

Figure 6.11: Equally divide a sum of money over a given period of time

One formula - How varying loan amounts impact the loan's installments

The traditional PMT function can handle only one amount at a time. But what if we would like to simulate more than one loan **Present value (PV)**?

Well, **SEQUENCE** comes to the rescue again:

We simulate two payment frequencies:

- Quarterly (in Figure 6.12) and
- Monthly (in Figure 6.15)

This is controlled by the parameter in cell E2 (Data Validation drop-down list with 2 values: 4 – quarterly, 12 – monthly).

Figure 6.12 makes this evident. We have six arguments for this PMT simulation:

1. Payment Frequency (cell E2) = quarterly
2. Annual rate (in cell G2) = 1.50%

3. Number of payments (in cell H2) = 24
4. Loan amount (in cell I2) = 100,000
5. Increment (in cell J2) = 10,000
6. Upper limit (in cell K2) = 250,000

The last two arguments are *dedicated* for the dynamic solution: in the *traditional* method, we needed only the first four arguments. The new arguments (number 5 and 6) are arguments to the **SEQUENCE** function (as can be seen in cell D10): we start with the loan amount (I2) with increments of (J2) for a number of instances calculated in the expression: $(\$K\$2 - \$I\$2) / \$J\$2 + 1$. So, in column B (cells B2:B17) we can see the sequence of loan amounts, which correspond to the (quarterly) payments in column A (cells A2:A17).

With only one formula we clearly see how varying loan amounts are reflected in the periodic payments as shown in the following screenshot:

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Quarterly Payment	loan sum		payment freq.			ann. rate	nper (mon.)	pv	increment	upper limit		
2	-12711.86	100,000		quarterly	4		1.50%	24	100000	10000	250000		
3	-13983.05	110,000											
4	-15254.23	120,000											
5	-16525.42	130,000											
6	-17796.60	140,000											
7	-19067.79	150,000											
8	-20338.97	160,000											
9	-21610.16	170,000											
10	-22881.35	180,000											
11	-24152.53	190,000											
12	-25423.72	200,000											
13	-26694.90	210,000											
14	-27966.09	220,000											
15	-29237.27	230,000											
16	-30508.46	240,000											
17	-31779.65	250,000											
18													
19													
20													
21													

Using SEQUENCE to simulate loan installments: usual PMT returns only one value. Now you can "play" with 6 arguments and return many results. We dynamically change pv: from 100K (I2) to 250K (K2) in increments of 10k (J2). We also present two payment frequencies: monthly (E2=12) or quarterly (E2=4).

=SEQUENCE(((\$K\$2-\$I\$2)/\$J\$2+1,,\$I\$2,\$J\$2))

=PMT(\$G\$2/\$E\$2,\$H\$2/(12/\$E\$2),SEQUENCE(((\$K\$2-\$I\$2)/\$J\$2+1,,\$I\$2,\$J\$2))

Checking results with: [loancalculator.org](https://www.loancalculator.org)
https://www.loancalculator.org/

Simulation of a loan

An excellent method to see how varying loan amounts are reflected in the periodic payment

Figure 6.12: Multiple loan amounts (PV) – quarterly payment frequency

Checking the formula's accuracy for a \$100,000 loan (payment= quarterly, annual rate=1.5%, number of payments=24). The result in Figure 6.12 (cell A2) is identical to the result in the following screenshot (Figure 6.13). The picture is from the Loan Calculator site:

Loan Calculator

Amortized Loan: Paying Back a Fixed Amount Periodically

Use this calculator for basic calculations of common loan types such as [mortgages](#), [auto loans](#), [student loans](#), or [personal loans](#), or click the links for more detail on each.

Loan Amount	<input type="text" value="\$100000"/>
Loan Term	<input type="text" value="2"/> years <input type="text" value="0"/> months
Interest Rate	<input type="text" value="1.5"/> %
Compound	<input type="text" value="Quarterly"/>
Pay Back	<input type="text" value="Every Quarter"/>
Calculate	

Results:	
Payment Every Quarter	\$12,711.86
Total of 8 Payments	\$101,694.87
Total Interest	\$1,694.87
View Amortization Table	

Figure 6.13: Multiple loan amounts (PV) – quarterly payment for a 100,000\$ loan

Checking the formula's accuracy for a \$250,000 loan (payment= quarterly, annual rate=1.5%, number of payments=24). The result in Figure 6.12 (cell A17) is identical to the result in the following screenshot (Figure 6.14). The picture is from the Loan Calculator site:

Loan Calculator

Amortized Loan: Paying Back a Fixed Amount Periodically

Use this calculator for basic calculations of common loan types such as [mortgages](#), [auto loans](#), [student loans](#), or [personal loans](#), or click the links for more detail on each.

Loan Amount	<input type="text" value="\$250000"/>
Loan Term	<input type="text" value="2"/> years <input type="text" value="0"/> months
Interest Rate	<input type="text" value="1.5"/> %
Compound	<input type="text" value="Quarterly"/>
Pay Back	<input type="text" value="Every Quarter"/>
Calculate	

Results:	
Payment Every Quarter	\$31,779.65
Total of 8 Payments	\$254,237.17
Total Interest	\$4,237.17
View Amortization Table	

Figure 6.14: Multiple loan amounts (PV) – quarterly payment for a 250,000\$ loan

The following screenshot is the counterpart of Figure 6.12. It shows the same loan simulation, but here, the pay back is monthly, not quarterly:

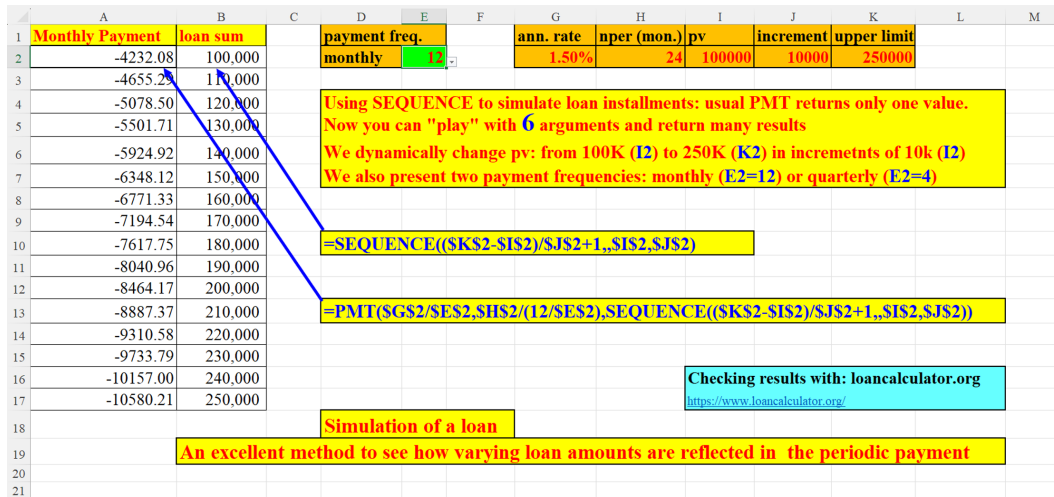


Figure 6.15: Multiple loan amounts (PV) – monthly payment frequency

Checking the formula's accuracy (Figure 6.15) for a \$100,000 loan (payment=monthly, annual rate=1.5%, number of payments=24). The result in Figure 6.15 (cell A2) is identical to the result in the following screenshot (Figure 6.16). The picture is from the Loan Calculator site:

Loan Calculator

Amortized Loan: Paying Back a Fixed Amount Periodically

Use this calculator for basic calculations of common loan types such as [mortgages](#), [auto loans](#), [student loans](#), or [personal loans](#), or click the links for more detail on each.

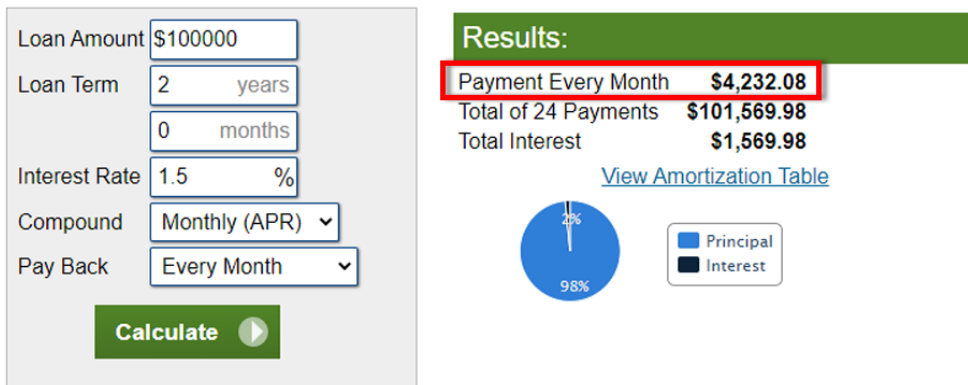


Figure 6.16: Multiple loan amounts (PV) – monthly payment for a 100,000\$ loan

Checking the formula's accuracy (Figure 6.15) for a \$100,000 loan (payment= monthly, annual rate=1.5%, number of payments=24). The result in Figure 6.15 (cell A17) is identical to the result in the following screenshot (Figure 6.17). The picture is from the Loan Calculator site :

Loan Calculator

Amortized Loan: Paying Back a Fixed Amount Periodically

Use this calculator for basic calculations of common loan types such as [mortgages](#), [auto loans](#), [student loans](#), or [personal loans](#), or click the links for more detail on each.

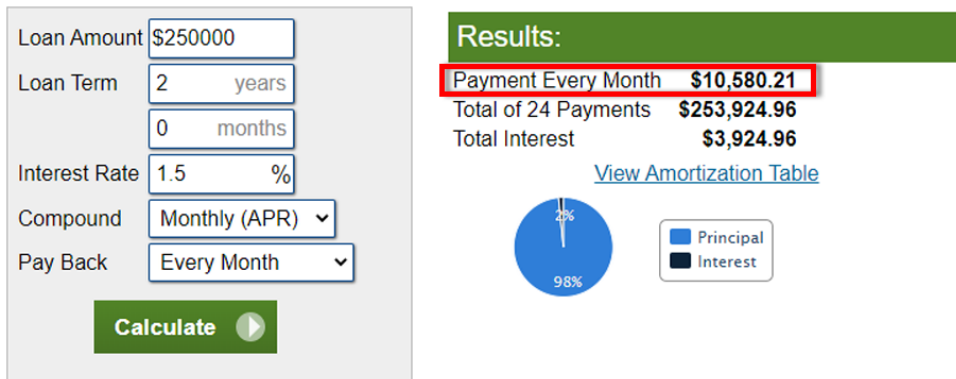


Figure 6.17: Multiple loan amounts (PV) – monthly payment for a 250,000\$ loan

NPV – No need for a data table

This section explains how the data table in Excel's Data Tab menu (**Data*Forecast*What-If Analysis*Data table**) becomes redundant if we use the **SEQUENCE** function.

The following is a sensitivity analysis of **Net Present Value (NPV)**:

- We have a two-dimensional array: the vertical (cells B2:B12) analyses the cash flow from: 10% to: +10% (in increments of 2%) [see cells N5:O7] whereas the horizontal array (cells C1:L1) examines the cost of capital from: 2% to: 20% (in increments of 2%) [see cells N1:O3]
- The investment (-500) can be seen in E17. We are analyzing the cash flow for years 1-6 (cells F17:K17).
- As can be seen in the formula written in cell C2, the NPV is calculated for the cash flow array (C2:C12, corresponding to B2:B2). The formula in cell C2 is then dragged to the right all the way to the last horizontal array item: C2:L2.

So here you have it, one formula instead of the *old-fashioned* data table:

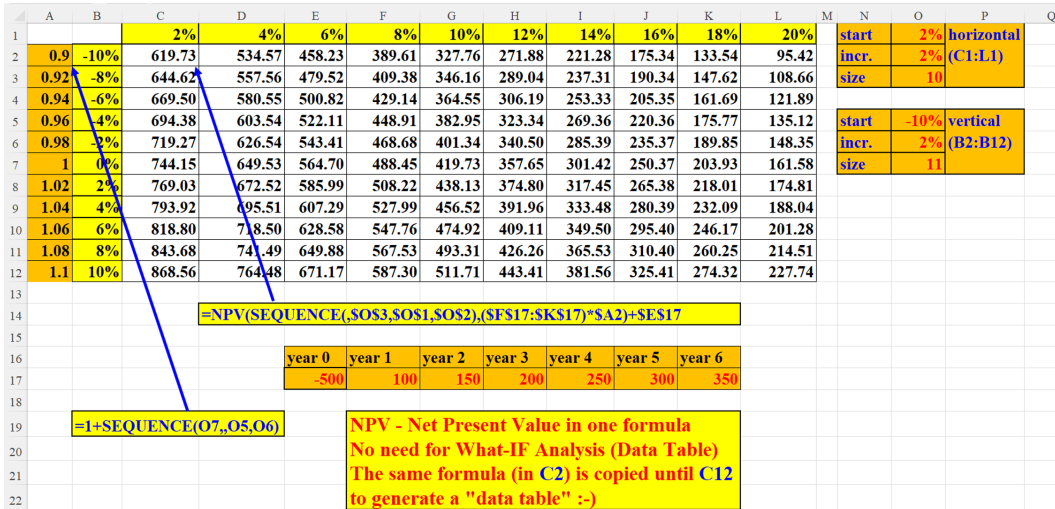


Figure 6.18: NPV – instead of a Data Table

PDURATION - Multiple results

The classic **PDURATION** function calculates the number of periods (in years or in months) until a certain investment amount (in our example: cell B1 = \$8000) reaches the desired **Future value** (FV), in our case: \$16,000 (in cell B2).

The classic solution is shown in cell D18. The **PDURATION** function takes three arguments: Monthly Rate (cell B20), PV and FV (B1 and B2, respectively, as explained previously).

However, replacing the first argument with an array (cells D5:D13), yields an array of results, as can be seen in cells A5:A13. The array size is set in cell A3. To make the solution even more flexible, we have added 2 spin buttons: one for the annual rate (in cell H1, linked to G1) and one for the increments (Δ in cell H2, linked to G2). Thus, we can control both the initial annual rate and the size of the increments, as shown in the following screenshot:

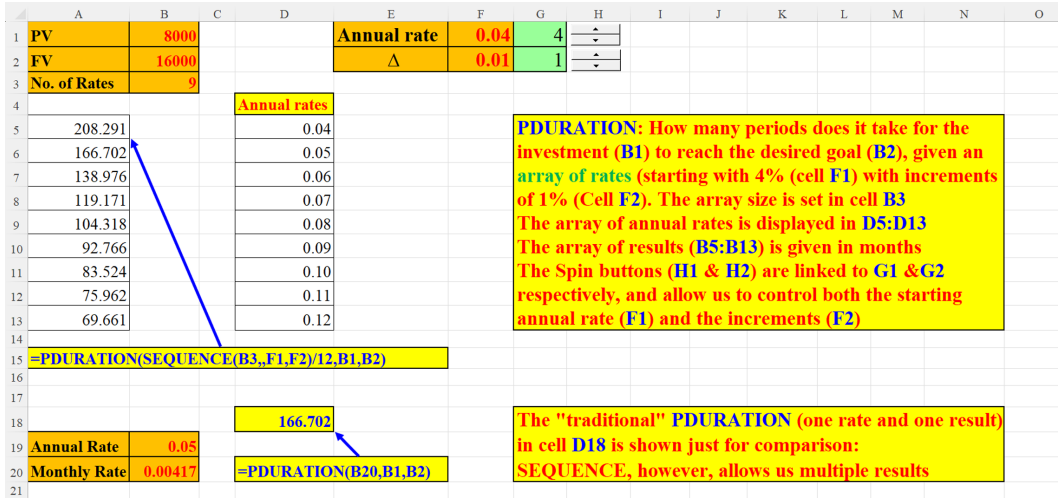


Figure 6.19: The PDURATION function – “classic” vs. “revolutionary”

The **Spin button** for the annual rate is defined in cell H1. It is linked to cell G1. Since the rates are expressed in percentage, the value in G1 is divided by 100 as shown in the following screenshot:

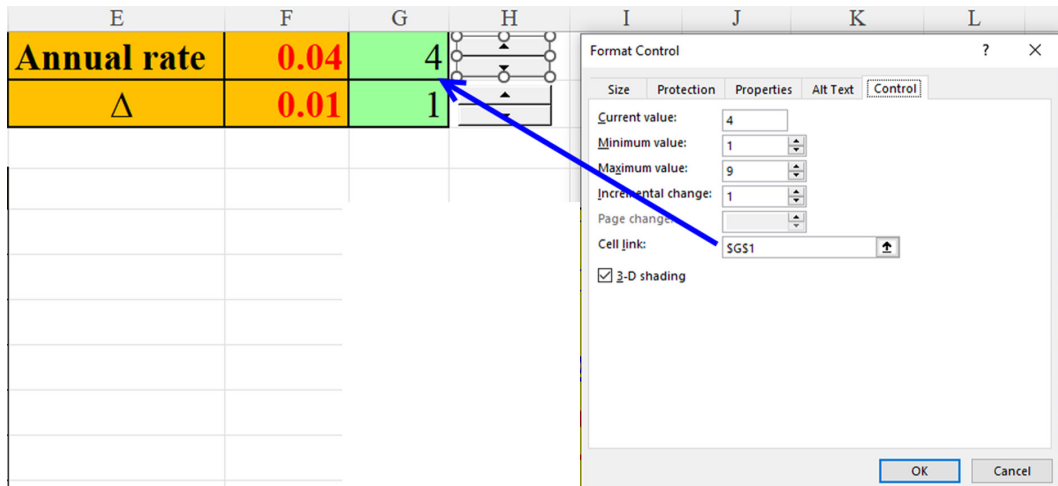


Figure 6.20: The PDURATION function – Spin Button – Annual Rate

The **Spin button** for the interest rate increments is defined in cell H2. It is linked to cell G2. Since the rates are expressed in percents, the value in G2 is divided by 100 as shown in the following screenshot:

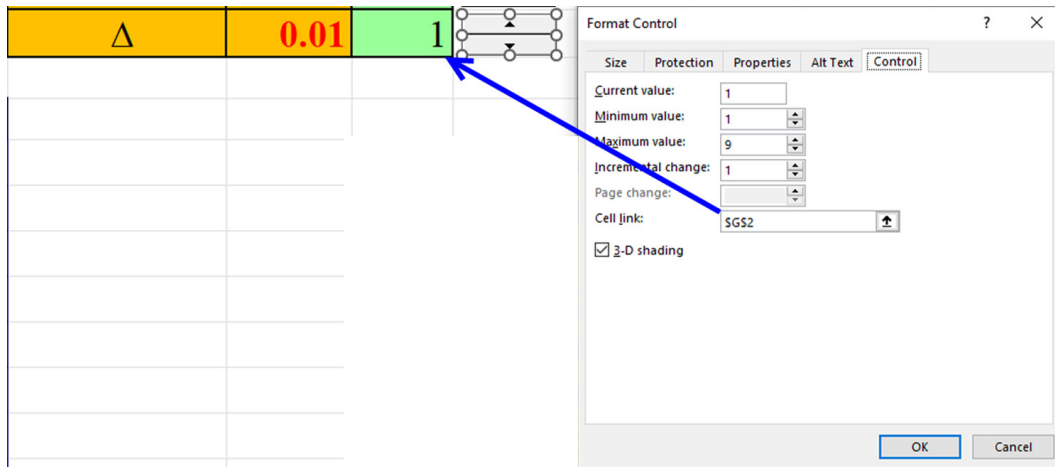


Figure 6.21: The PDURATION function – Spin Button – Increment (Δ)

The RATE function – multiple results

The **RATE** function usually returns only one value. It takes three arguments:

- Loan period (cell D1, in Figure 6.22)
- Monthly payment (cell D2, in Figure 6.22)
- Loan amount (cell D3, in Figure 6.22)

However, by replacing the last argument with an array of 11 members (parameter in cell G1), as shown in cells D7:D17, we can examine the impact of 11 different loan amounts on the monthly rate, (cells A7:A17) where both the loan period and the monthly payment are immutable.

The multiple results option in the **RATE** function is possible, of course, thanks to the **SEQUENCE** function as shown in the following screenshot:

	A	B	C	D	E	F	G	H	I	J	K	L
1			Loan period (months)	60	Number of Simulations		11					
2			Monthly payment	600								
3			Loan amount	\$ -25,000.00								
4			delta loan amount	\$ -500.00								
6	Monthly Rate			Loan Amount			Checking PMT for Monthly Rate calculated in A7:A17					
7	1.283%			\$ -25,000.00		\$ 600.00 <=	=PMT(\$A7,\$D\$1,D7)					
8	1.209%			\$ -25,500.00		\$ 600.00 <=	=PMT(\$A8,\$D\$1,D8)					
9	1.136%			\$ -26,000.00		\$ 600.00 <=	=PMT(\$A9,\$D\$1,D9)					
10	1.065%			\$ -26,500.00		\$ 600.00 <=	=PMT(\$A10,\$D\$1,D10)					
11	0.996%			\$ -27,000.00		\$ 600.00 <=	=PMT(\$A11,\$D\$1,D11)					
12	0.929%			\$ -27,500.00		\$ 600.00 <=	=PMT(\$A12,\$D\$1,D12)					
13	0.864%			\$ -28,000.00		\$ 600.00 <=	=PMT(\$A13,\$D\$1,D13)					
14	0.800%			\$ -28,500.00		\$ 600.00 <=	=PMT(\$A14,\$D\$1,D14)					
15	0.738%			\$ -29,000.00		\$ 600.00 <=	=PMT(\$A15,\$D\$1,D15)					
16	0.678%			\$ -29,500.00		\$ 600.00 <=	=PMT(\$A16,\$D\$1,D16)					
17	0.618%			\$ -30,000.00		\$ 600.00 <=	=PMT(\$A17,\$D\$1,D17)					
19			=RATE(D1,D2,SEQUENCE(G1,,D3,D4))			=SEQUENCE(G1,,D3,D4)						
21	Checking multiple rates by changing gradually the loan amount. The loan period (D1) and the monthly payment (D2) are fixed while we check the impact of varying loan amounts (shown in D7:D17) on the monthly rate (cells A7:A17). No. of Simulations is defined in G1											

Figure 6.22: The RATE function – multiple results

RRI - calculate the average annual interest rate of an investment

The RRI financial function calculates the average annual interest rate for a period.

It takes the following arguments:

The number of years invested (nper), the starting amount (PV), and the ending amount (FV).

This function usually returns only one value. However, by replacing the first argument with an array (created by the SEQUENCE function), we can simulate multiple interest rates for multiple periods. In our example, we simulate 10 different periods, from one year to 10 years. The interest rates, as can be seen in cells A1:A10 change accordingly.

The PDURATION function (explained previously, PDURATION – multiple results) implemented here in cells: H10:H19, is an excellent tool to verify the results of the RRI function as shown in the following screenshot. If we calculate the PDURATION, where its first argument is the rate (in cell A1=30%), the PV is 10000 (I2+J2) and FV is 13000 (I3+J3) – the result is: 1 (one year). This means that the RRI for one year, when the PV is: 10000 and FV is: 13000, will be 30%. The fact that these two

functions are *interchangeable* makes it possible to check the result of the **RRI** with the **PDURATION** function, and vice versa:

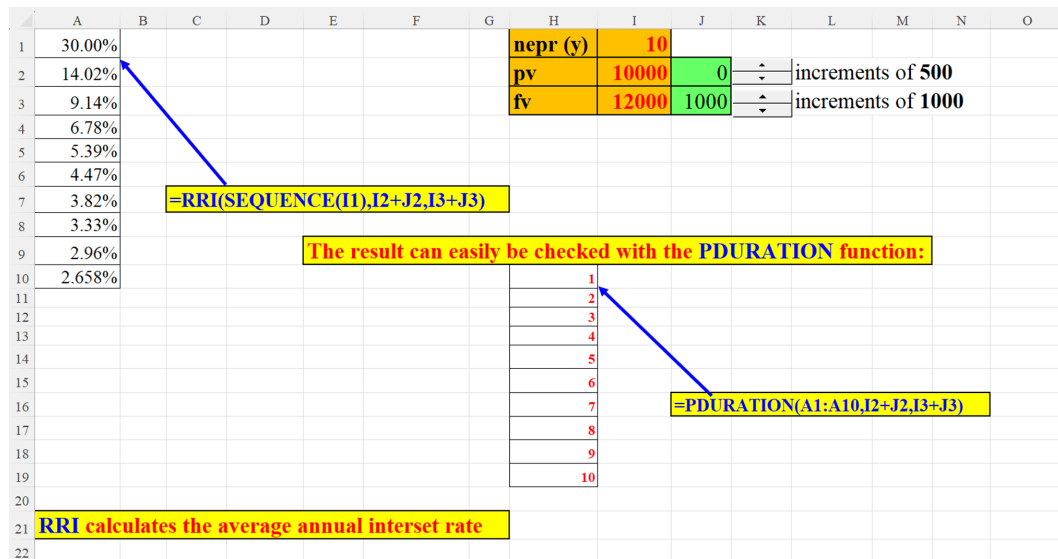


Figure 6.23: Adding multiple average annual interest rates instead of just one

The **Spin** button for incrementing the **Present Value (PV)** is defined in cell K2. It is linked to cell J2, and serves to enhance the PV in increments of 500: 500,1000,1500 etc. Its minimum value is: 0 and its maximum value is: 9500.

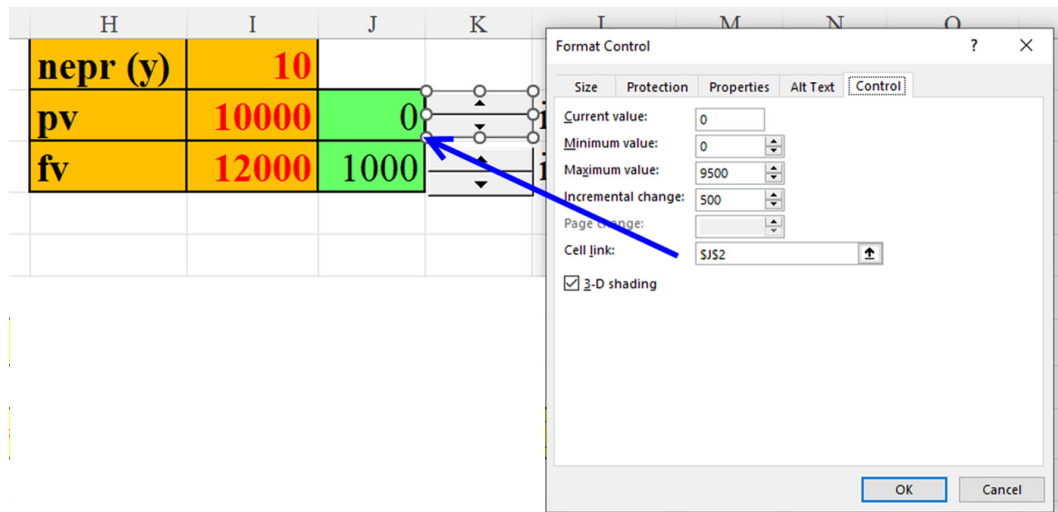


Figure 6.24: Adding multiple average annual interest rates spin button - PV

The **Spin button** for incrementing the **Future Value (FV)** is defined in cell K3. It is linked to cell J3, and serves to enhance the PV in increments of 1000: 1000,2000,3000 etc. Its minimum value is: 0 and its maximum value is: 30000:

H	I	J	K	L	M	N	O
nepr (y)	10						
pv	10000	0					
fv	12000	1000					

increments of 500

Format Control

Size Protection Properties Alt Text Control

Current value: 1000

Minimum value: 0

Maximum value: 30000

Incremental change: 1000

Page change:

Cell link: \$J\$3

3-D shading

OK Cancel

Figure 6.25: Adding multiple average annual interest rates spin button - FV

SEQUENCE and SUM

In this section, we will learn how to sum only transactions from a certain part of the year.

The following figure demonstrates a method by which you can sum transactions which belong to a certain part of the year, for example: April-September, June-October, February-August, and so on.

The trick is to use the months to sum as an array: the number of elements to sum is determined by the expression: $K2-K1+1$ and the starting month is set by the **From Month** parameter (cell K1):

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Date	Sum								From Month	4		
2	01/01/2022	8277.00			SUM only month >=4 & <=9					To Month	9		
3	01/02/2022	3693.00											
4	04/03/2022	825.00											
5	04/04/2022	5814.00											
6	05/05/2022	5377.00											
7	05/06/2022	8178.00											
8	06/07/2022	8401.00					44089.00						
9	06/08/2022	7172.00											
10	06/09/2022	9147.00											
11	07/10/2022	1072.00											
12	07/11/2022	3539.00											
13	08/12/2022	2651.00					=SUM((B2:B13)*(MONTH(A2:A13)=(SEQUENCE(,K2-K1+1,K1))))						
14													
15													
16													
17													
18													
19													
20													

Figure 6.26: Sum only transactions from a defined sequence of months

Conclusion

Most financial functions in Excel accept only single-value arguments and return single-value results. In this chapter, we have shown how to convert such traditional functions into multiple-result functions, thanks to the **SEQUENCE** function. Moreover, in cases where we could achieve an array of results (for example, in the **NPV** function, by using a Data table, a built-in function in Excel), we showed that this unwieldy method can be easily replaced by the **SEQUENCE** function.

In the next chapter, we will discuss some interesting implementations of **SEQUENCE** in mathematics: algebra, geometry, trigonometry and more.

Points to remember

- SEQUENCE is extremely practical when you weigh several options, in loans or investments, looking for the optimal interest rate, loan period, periodic payment amount and so on.
- Designing your arguments cleverly can be immensely helpful when setting the increments (Δ) for a certain argument. For example, if you want to see the impact of every fraction of an annual interest rate (0.2%, 0.25%, 0.3%) set the increments accordingly.
- Using **Spin** buttons makes the process much faster and easier. The **Spin** buttons add an additional level of flexibility.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



CHAPTER 7

SEQUENCE -

The Ancilla of

Math

Introduction

Every software is built upon binary logic, that is, numbers containing only 0s and 1s. In this sense, Excel is no exception. However, since Excel is basically a calculatory software, it also has many mathematical functions. The examples in this chapter are going to demonstrate its outstanding capabilities, both in calculations and in its graphical user interface: a visual display of the mathematical computation. This chapter mentions about two dozen examples showcasing the integration of **SEQUENCE** with mathematical functions. The illustrations cover some areas such as: algebra (fractions, sums, powers), probability, trigonometry, bit operations and more.

Structure

In this chapter, we will discuss the following topics:

- Examples of SEQUENCE in math operations
 - A number to the power of
 - Two methods to create a sequence of square roots
 - Two methods to generate a sequence of fractions
 - Creating a sequence of alternate 1's and 0's

- Dynamic quadratic equation
- SUM - a virtual Array
- How many candles
- Raising the number 2 to the power of 10 using bit operation
- Simplest OR
- Dynamic Sine with 2 Spin Buttons
- Exponential Growth example
- Dynamic multiplication table
- BIN2DEC – MMULT and SEQUENCE
- BIN2DEC - SUM (or SUMPRODUCT) with SEQUENCE
- Filling the missing values in a geometric series
- Trigonometry with SEQUENCE
- An array of duplicate numbers generated by bit operations
- Using MMULT and SEQUENCE to track wins, losses, and ties in each quarter
- Digital root
- First N odd numbers squared (A simple solution)
- First N odd numbers squared (A complex solution)
- Find first divisor of a number (divisor found)
- Find first divisor of a number (divisor not found)

Objectives

The objective of this chapter is to show you how advantageous the use of **SEQUENCE** is when dealing with mathematical operations. Since **SEQUENCE** is an array function, it can speed up your calculations by executing several solutions concurrently. Learning these techniques will undoubtedly save you a lot of time, effort, and frustration.

Examples of SEQUENCE in math operations

The ensuing examples demonstrate the implementation of **SEQUENCE** in math.

A number to the power of

The **POWER** function in Excel yields only one solution. However, by using the **SEQUENCE** function cleverly, we can achieve multiple solution with only one formula. In the example below we use two spin buttons to make the array solution even more flexible. We can control both the numbers to be raised to the desired exponent (in cell J1) and the exponent (in cell M1) as shown in the following screenshot:

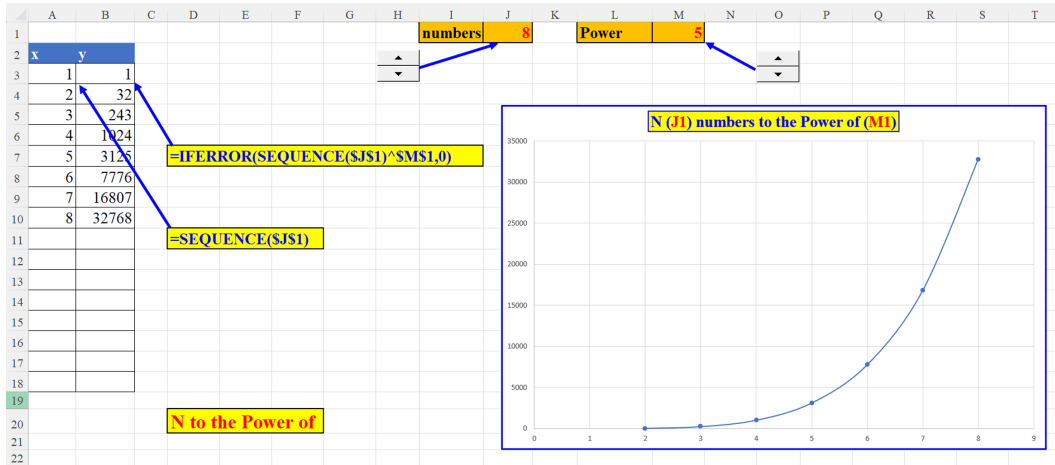


Figure 7.1: A number to the power of

The **Spin** button linked to cell J1 controls the numbers presented in column A as shown in the following screenshot:

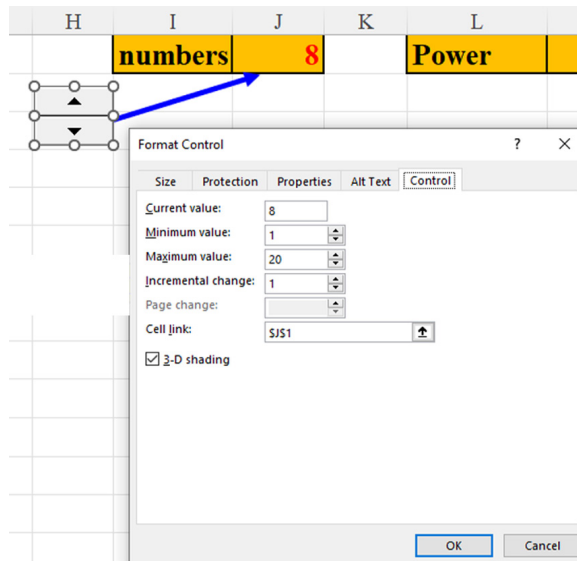


Figure 7.2: The Number Spin button

The **Spin** button linked to cell M1 controls the power by which the numbers in column A are raised.

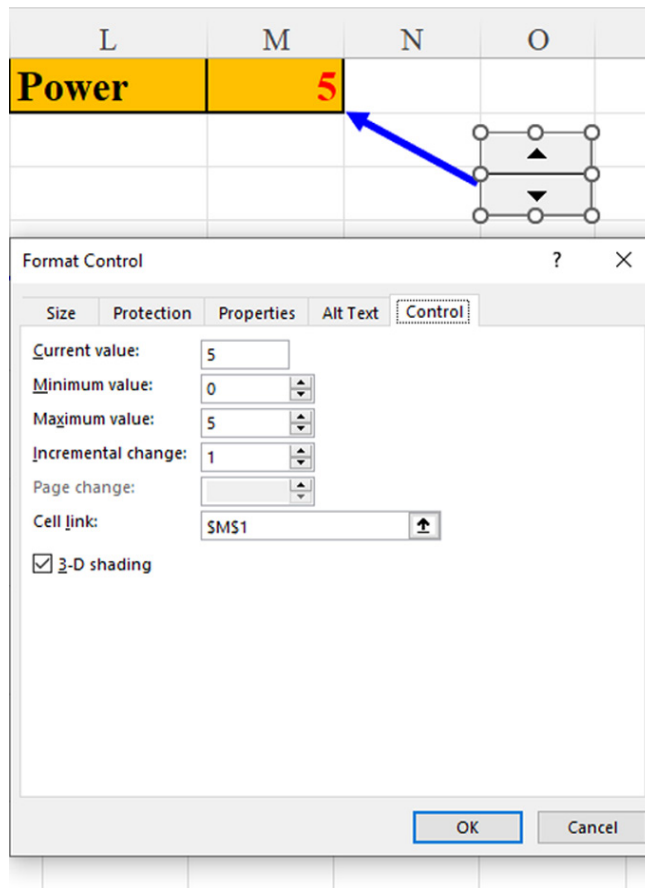


Figure 7.3: The Power Spin button

Two methods to create a sequence of square roots

The method on the left-hand side of *Figure 7.4* uses array constants. This is a static array, that is, whenever you want to increase or decrease the array's size, you need to tamper with the formula. On the right-hand side of the following figure, we have a dynamic array created, with the **SEQUENCE** function. If we want to change the array's size, all we have to do is change the parameter (in L1). We do not touch the formula as shown in the following figure:

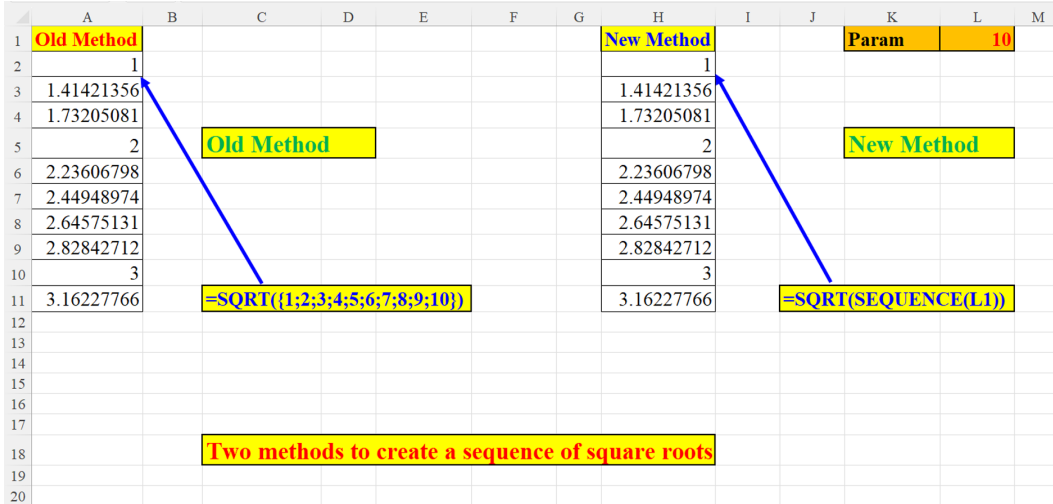


Figure 7.4: Two methods to create a sequence of square roots

Two methods to generate a sequence of fractions

To create an array of n fractions we need to know its size (n is 9, defined in cell E1), and the fraction itself (0.1, defined in cell E2). The first method (on the left-hand side) saves us the bother of defining the first member of the array as shown in the following screenshot:

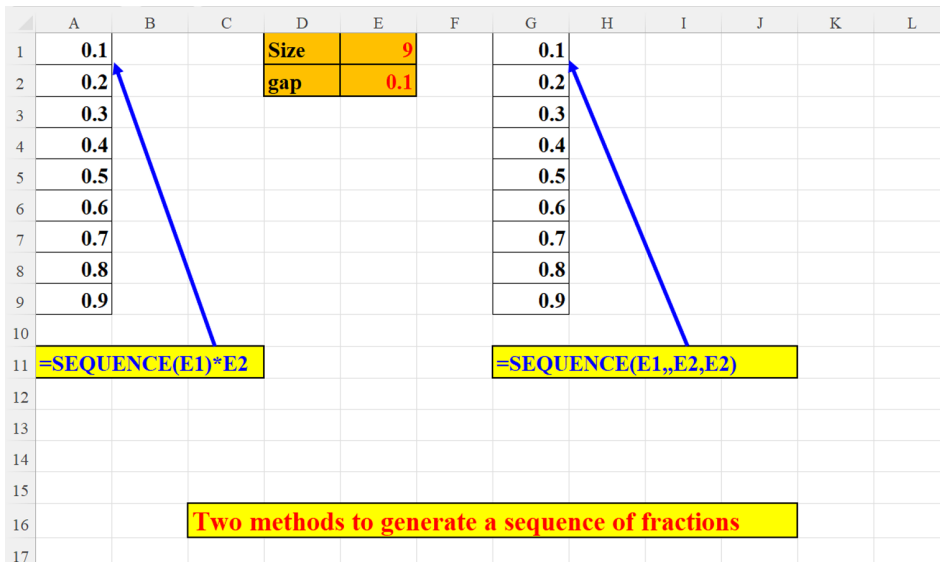


Figure 7.5: Two methods to create a sequence of fractions

Creating a sequence of alternate 1's and 0's

In this section, we will use the **BITAND** function to generate a series of alternate 1's and 0's. To spawn the opposite series, we use the **NOT** operator as shown in the following screenshot:

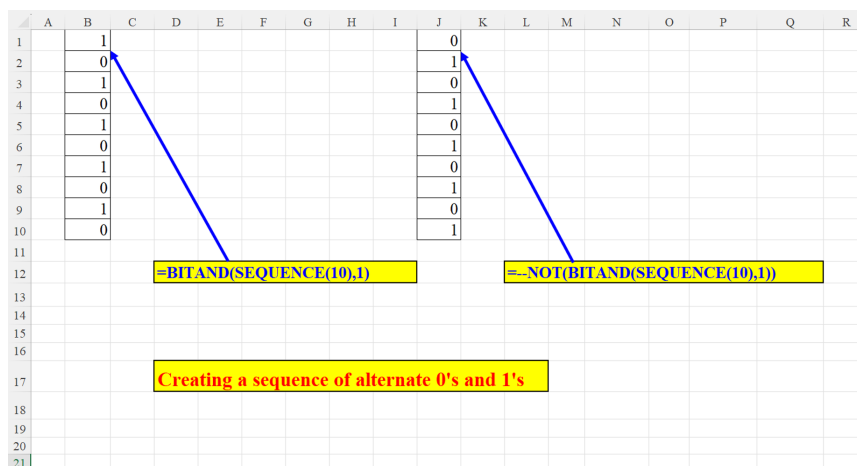


Figure 7.6: Creating a sequence of alternate 1's and 0's

Dynamic quadratic equation

Here is a demonstration of the $Y=X^2$ equation, a quadratic equation. Each value in column A (x) is raised to the power of 2 in column B (y). The size of the array is determined by a **Spin** button attached to cell P1 (in our case: 5, so the x axis goes from -5 to 5), as can be seen in the **SEQUENCE** function. We use a helper cell (J1) because a spin button cannot handle negative numbers as shown in the following screenshot:

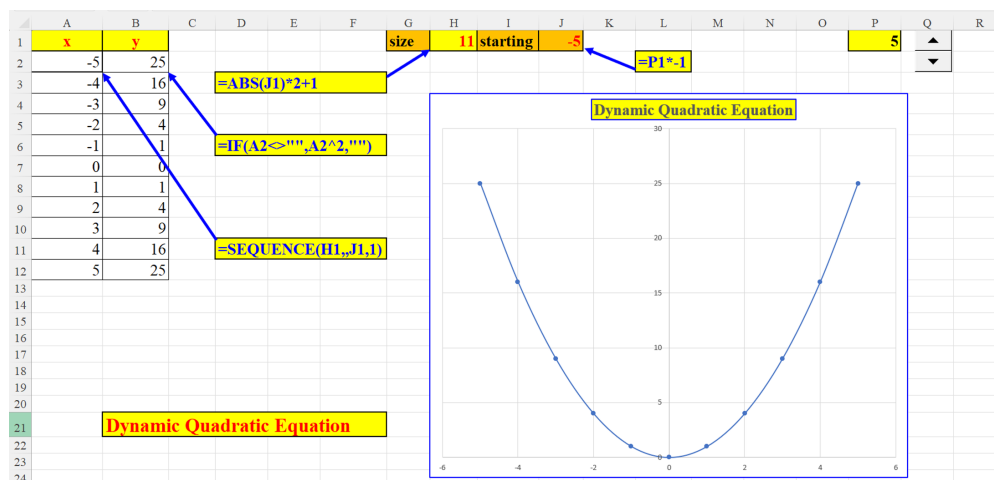


Figure 7.7: Dynamic quadratic equation

The **Spin** button linked to cell P1 determines the two extremes of the array in column A. For example, if P1 is 5 (as in our example), then the array's range is from -5 to +5:

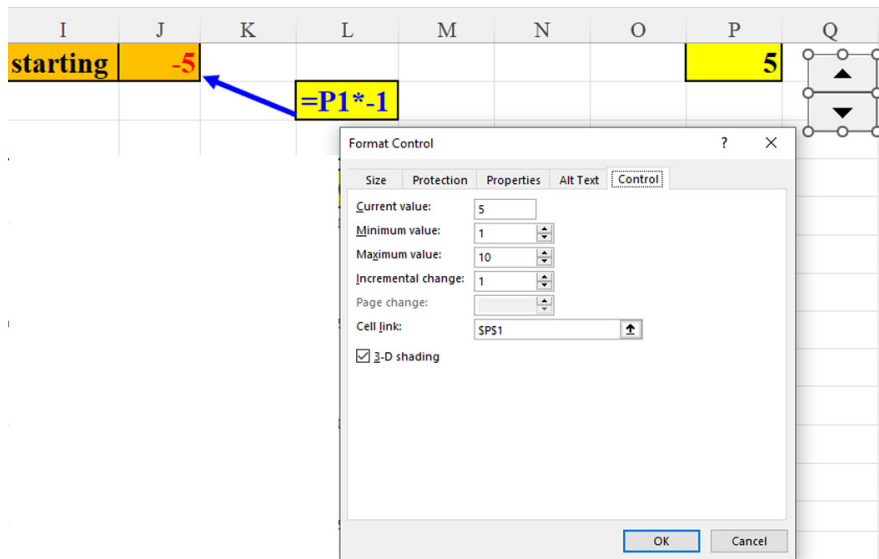


Figure 7.8: Dynamic quadratic equation – Spin button

SUM - a virtual Array

The upper-part function is a demonstration of a bi-dimensional array: 4 rows by 4 columns, which starts with one and is augmented by 3. The sum function on the right then sums this array. The notation C2# is a spill range reference that refers to the entire array and not to cell C2 alone. The formula in the lower part attains the same result with a *virtual* array, that is, the array created (which is exactly the same as the previous one) exists in Excel's memory, but cannot be seen on the spreadsheet itself as shown in the following screenshot:

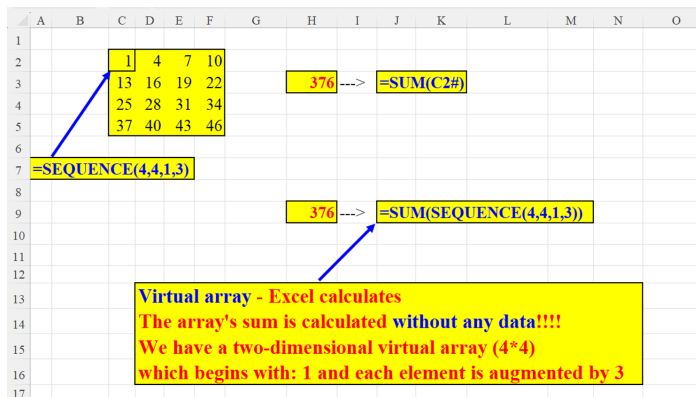


Figure 7.9: SUM a virtual array

How many candles

In the Jewish Feast of lights (which lasts eight days), we light a Hanukkiah (a nine-branched candelabrum). We start by lighting two candles on the first day and ending with nine candles in the eighth and last day of the holiday. But what if we want to add a candelabrum on each day till the last, that is, the eighth day? We will have 8 candelabras. How can we calculate the total candles needed for all the candelabras during the whole feast?

	A	B	C	D	E	F	G	H	I
	Day	Candles needed for 1 candelabra	Candles needed for N candelabras						
1									
2	1	2	2						
3	2	3	6						
4	3	4	12		240				
5	4	5	20						
6	5	6	30						
7	6	7	42						
8	7	8	56						
9	8	9	72						
10			240						
11									
12									
13	The Jewish Feast of Lights (Hanukkah) lasts 8 days. On the 1st day you light 2 candles, on the 2nd - three, on the 3rd - four etc. The question: how many candles do we need, if: On the first day you light only one Hanukkiah (9-candle candelabra), on the 2nd - 2 candelabras, on the 3rd - 3 candelabras...?								
14									
15									
16									
17									
18									
19									
20									
21									

Figure 7.10: How many candles

Raising the number 2 to the power of 10 using bit operation

The **BITLSHIFT** function is a bit operation which shifts to the left bit at a time. Each such shift multiplies the result by 2. Using the **SEQUENCE** function, we can dynamically create a geometric series whose size is set by the parameter in cell H1 as shown in the following screenshot:

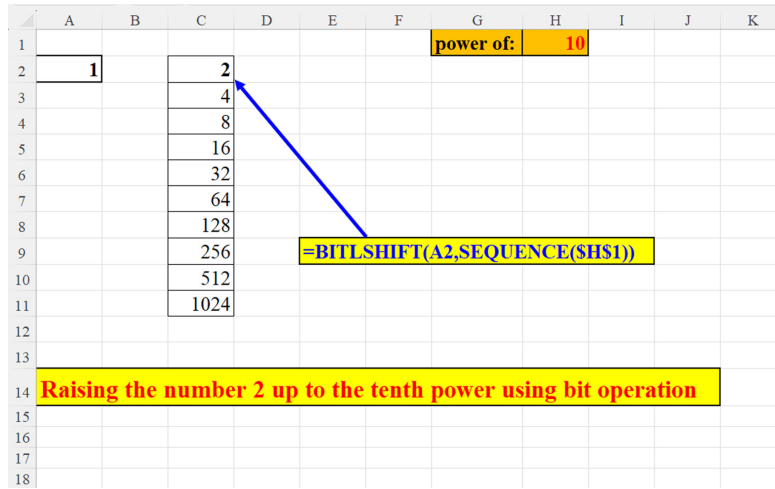


Figure 7.11: Raising the number 2 to the tenth power using bit operation

Simplest OR

The fastest, simplest method to check whether the value defined in cell A1 is an integer between 1 and 10 as shown in the following screenshot:

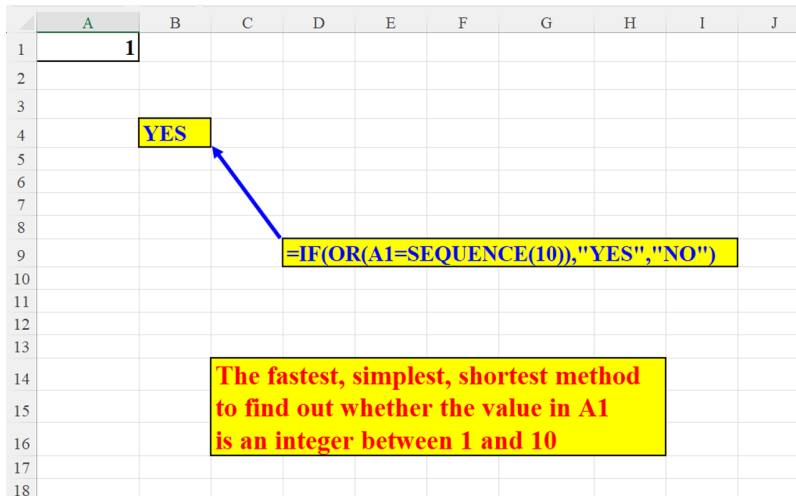


Figure 7.12: Simplest OR

Dynamic Sine with two Spin buttons

The following figure displays the use of **SEQUENCE** to create a dynamic sine chart. On the left-hand side we have a table of x - y values, created dynamically with two

spin buttons. The x-axis values are expressed in fractions of Radians ($=\text{PI}() = 180$), generated by the first spin button in cell I2 and set by cell G2 (the amplitude). The distance of the chart is indicated by the second Spin button in cell N2:

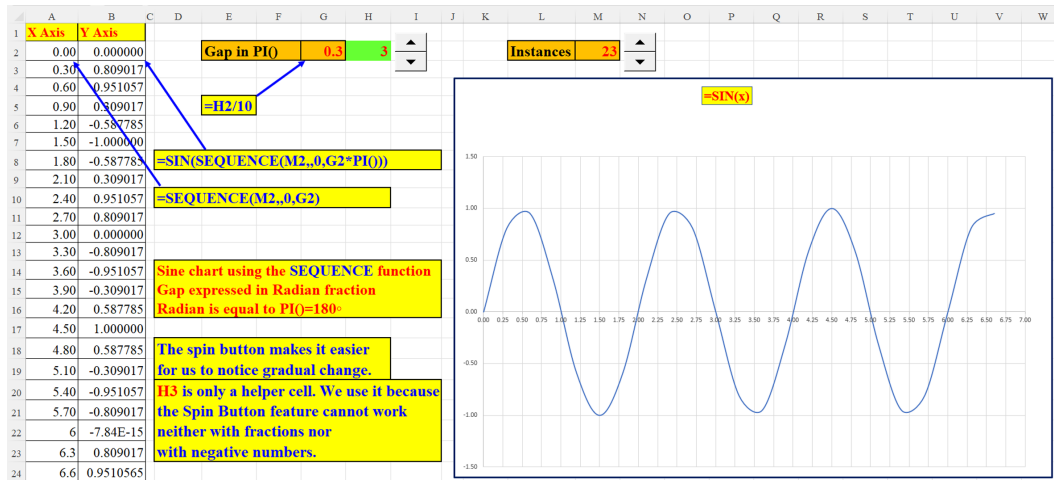


Figure 7.13: Dynamic SINE with two Spin buttons

The Spin button in cell I2 sets the amplitude of the chart. Since Spin buttons cannot take fractions, we divide the value in H2 (the cell linked to the Spin button) by 10, to produce a fraction (in G2) as shown in the following screenshot:

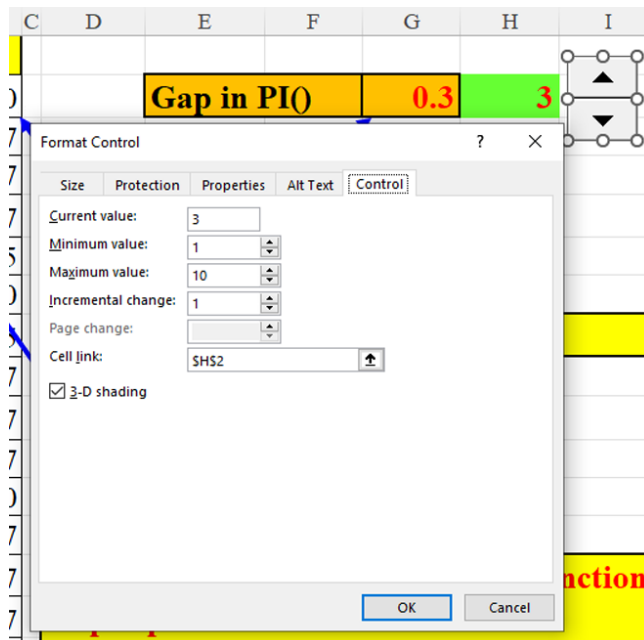


Figure 7.14: Dynamic SINE – Spin Button 1 – Gap in PI()

The **Spin** button in I2 determines the number of the chart's instances (x -axis values) as shown in the following screenshot:

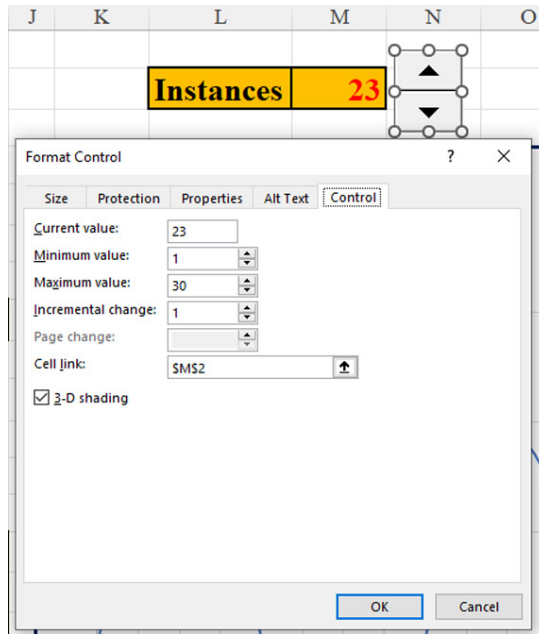


Figure 7.15: Dynamic SINE – Spin Button 2 – number of instances

Exponential Growth example

Figure 7.16 displays an array of numbers raised to the power of 2 over a period of 12 months; each value in the array is twice as large as its predecessor. The initial number is a parameter (in cell G1) as shown in the following screenshot:

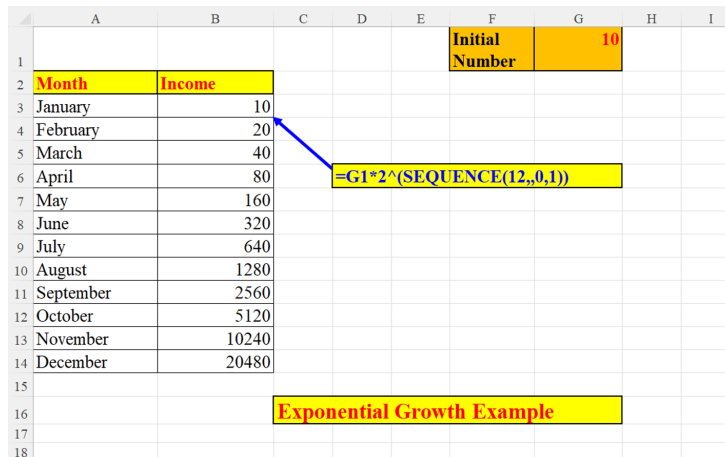


Figure 7.16: Exponential growth example

Dynamic multiplication table

The dynamic multiplication table shown below is controlled by two parameters which decide its horizontal (cell Q1) and vertical size (cell Q2):

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1		1	2	3	4	5	6	7	8	9	10					Horiz.	10			
2	1	1	2	3	4	5	6	7	8	9	10					Vert.	5			
3	2	2	4	6	8	10	12	14	16	18	20									
4	3	3	6	9	12	15	18	21	24	27	30									
5	4	4	8	12	16	20	24	28	32	36	40									
6	5	5	10	15	20	25	30	35	40	45	50									
7																				
8																				
9																				
10																				
11																				
12																				
13																				
14																				
15																				
16																				
17																				
18																				

Figure 7.17: Dynamic multiplication table

BIN2DEC – MMULT and SEQUENCE

The following figure demonstrates a method of converting a binary number (in A1) into a decimal number. The formula makes use of the **MMULT** function. This function returns the matrix product of two arrays: the array of the input's digits and the array of the descending numbers: 7,6,...,1, raised to the power of 2 as shown in the following screenshot:

The first array (of the input digits) returned by: `MID(A1,SEQUENCE(LEN(A1)),1)*1` is: {1,1,1,1,1,1,1}

The second one is the array of two raised to the power of the following numbers: 7,6,5,4,3,2,1,0. These digits represent the location of the digits within the original string, from left to right. We start from 7 (not from 8) because the value of number in the n-th place, is: $2^{(n-1)}$.

So, the array generated by `(2^(LEN(A1)-SEQUENCE(LEN(A1))))` is: {128;64;32;16;8;4;2;1}

The multiplication of the horizontal array by the vertical array yields the desired result 255 as shown in the following screenshot:

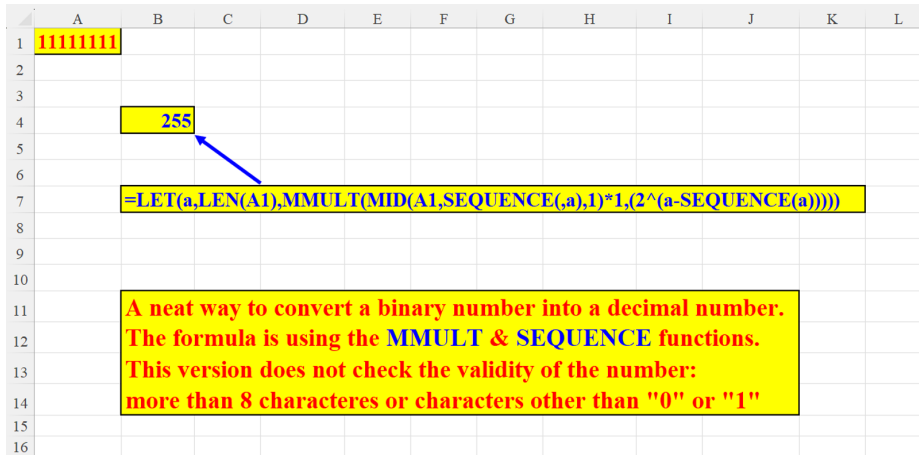


Figure 7.18: BIN2DEC – MMULT and SEQUENCE

BIN2DEC - SUM (or SUMPRODUCT) with SEQUENCE

This solution is similar to the previous one, but here we will take advantage of the **SUM** or **SUMPRODUCT** function instead of the **MMULT** function as shown in the following screenshot.

This solution is identical to the previous one (Figure 7.18). The only difference is that here we use **SUMPRODUCT** instead of **MMULT**:

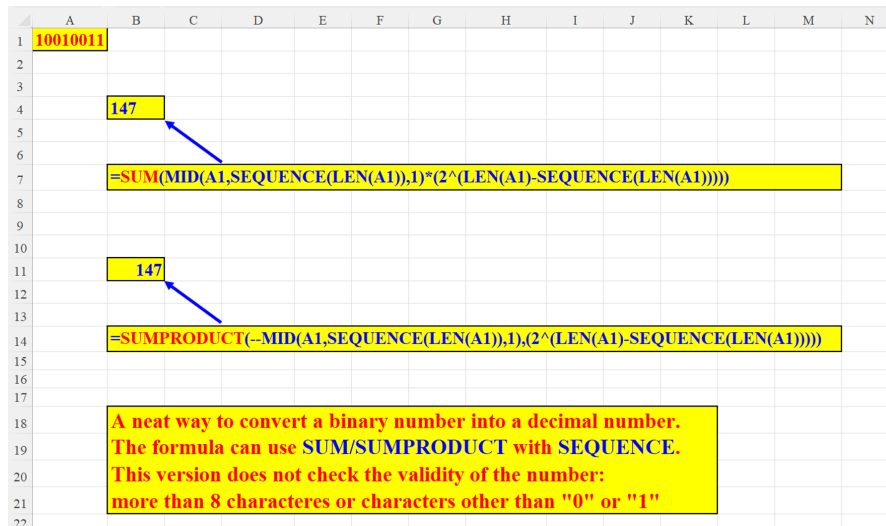


Figure 7.19: BIN2DEC – SUM (or SUMPRODUCT) and SEQUENCE

Filling the missing values in a geometric series

Suppose you have amounts of money spread over twelve months. You need to know the exact amount for each month, but all you know are the extremes in column B: the amounts of January (smallest) and December (largest). You know that the sum grows exponentially (thus creating a geometric series). The formula in B3 calculates the ratio of the series (the series grows by the power of 2) and correctly loads the amounts of the missing months:

	A	B	C	D	E	F	G	H	I	J	K
1	Month	Income									
2	January	450									
3	February	900									
4	March	1800									
5	April	3600									
6	May	7200									
7	June	14400									
8	July	28800									
9	August	57600									
10	September	115200									
11	October	230400									
12	November	460800									
13	December	921600									
14											
15											
16											
17											
18											

$=B2*((B13/B2)^(1/(12-1)))^SEQUENCE(10)$

Filling the missing values in a geometric series (exponential growth), by finding the ratio (n).
 No. of terms: 12, 1st term (a₁=450),
 last term (a₁₂=921600)
 $921600 = 450 * n^{(12-1)}$
 $n^{11} = 921600/450 = 2048$
 $n = 2048^{(1/11)} = 2$

Figure 7.20: Filling the missing values in a geometric series

Trigonometry with SEQUENCE

Figure 7.21 presents two trigonometric functions in Excel: SIN() and COS() through a full cycle (360°).

On the left-hand side the table of numeric values for each function is displayed. It is presented in degrees, according to the parameter specified in cell M1.

On the right-hand side a graphical exposition of this cycle can be seen as shown in the following screenshot.

Please pay attention to the notation of the formulae in cells: B2 and C2.

The # in the A2# denotes the Spilled Range Operator. A2# is the array created by the formula in cell A2:

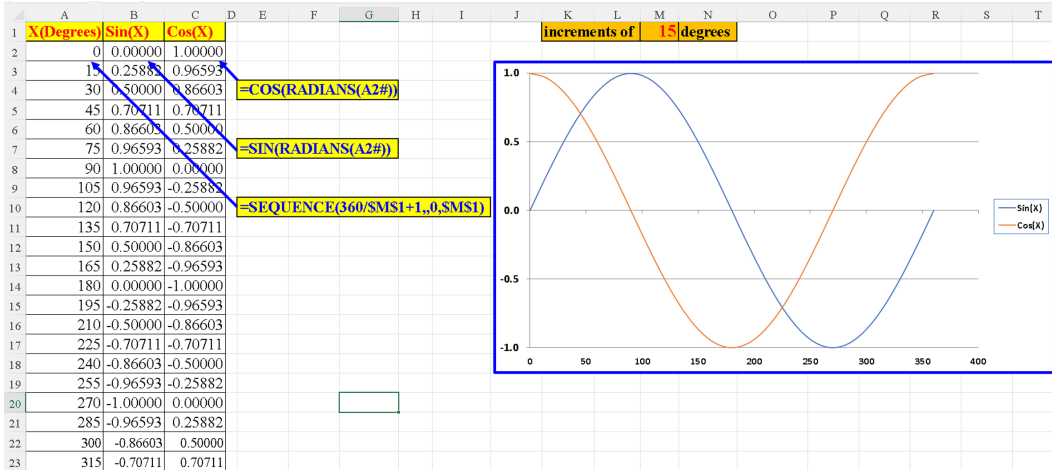


Figure 7.21: Trigonometry with SEQUENCE

An array of duplicate numbers generated by bit operations

Suppose you have a vertical array of consecutive numbers, but you want to duplicate each member of the array. This can easily be done with BIT operations. The **SEQUENCE** function creates a 14-member array starting with 2 and ending with 15. The **BITXOR** function converts every odd number to even, and vice versa. Finally, the **BITRSHIFT** divides each number by 2, returning the correct answer as shown in the following screenshot:

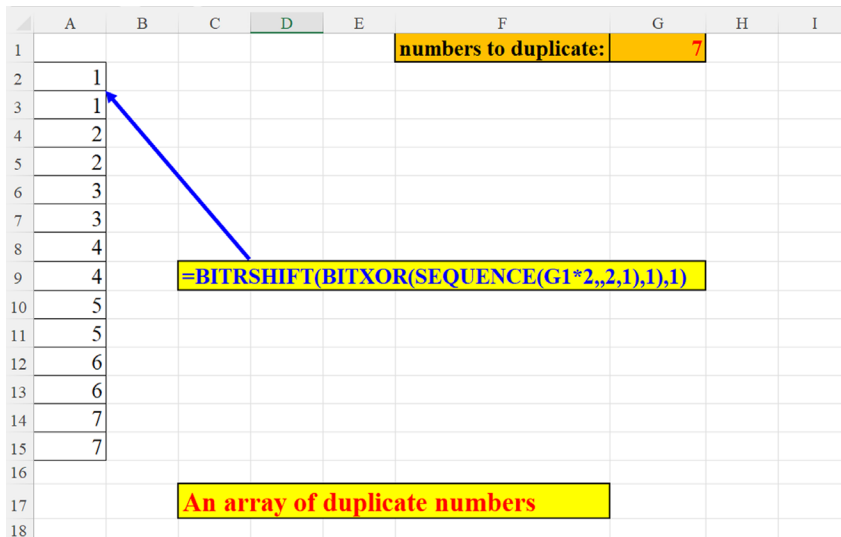


Figure 7.22: An array of duplicate numbers generated by bit operations

Using MMULT and SEQUENCE to track wins, losses, and ties in each quarter

This formula was inspired by the legendary Frankens Team.

The challenge is this: we have a data set of our football team, which competed in many games over a period of several months. Each entry specifies the number of wins, losses, and ties for that month. But sometimes, we have more than one entry per month. We want to analyze the data by the quarter: how many wins, losses and ties do we have in each quarter. The formula uses the **MMULT** function: the dates in column A are converted to the year's quarters and are compared to the relevant quarter. This creates an array of **TRUES** and **FALSEs** which is then multiplied by the games results (cells B2:D15), that yields the correct outcome as shown in the following screenshot:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	
1	Month	Wins	Losses	Ties														
2	01/2021	9	9	2														
3	08/2021	8	9	3														
4	09/2021	11	2	7														
5	10/2021	7	8	5														
6	02/2021	5	9	6														
7	03/2021	4	8	8														
8	04/2021	8	9	3														
9	04/2021	5	6	9														
10	05/2021	12	3	5														
11	06/2021	12	1	7														
12	07/2021	5	10	5														
13	10/2021	9	6	5														
14	11/2021	8	10	2														
15	12/2021	3	8	9														
16		106	98	76														
17																		
18	Quarter 1	18	26	16														
19	Quarter 2	37	19	24														
20	Quarter 3	24	21	15														
21	Quarter 4	27	32	21														
22																		
23		=MMULT(-TRANSPOSE(CEILING(MONTH(\$A\$2:\$A\$15)/3,1)=SEQUENCE(4)),SBS2:SDS15)																
24																		

Figure 7.23: Using MMULT and SEQUENCE to track wins, losses, and ties in each quarter

Digital root

The digital root is the process by which you take a number and add all its digits (iteratively) until you are left with only one digit. There are, of course, several algorithms to solve this challenge. This solution was devised before Microsoft introduced **LAMBDA** and its auxiliary functions.

The algorithm implemented here is that the whole process does not require more than two steps, as illustrated in the examples: The process can end in one step (for example: 440 -> 4 + 4 + 0 -> 8) or two. If it ends in two steps, then in case the result at the end of step two is two digits (for example: 2345678 -> 2 + 3 + 4 + 5 + 4 + 6 + 7 + 8 -> step 1: 39 -> 3 + 9 -> step 3: 12), we subtract 9 from the result, so the solution has only one digit (12 - 9 = 3) as shown in the following screenshot:

	A	B	C	D	E	F	G	H	I	J	K	L
		Number		Sum of the Number's digits		"breaking" the number's digits	step 1	step 2	desidered result			
1												
2		90456		6		9+0+4+5+6	-> 24	-> 2+4	-> 6			
3		21		3		2+1	-> 3		-> 3			
4		89940438048		3		8+9+9+4+0+4+3+8+0+4+8	-> 57	-> (5+7)-9	-> 3			
5		1899		9		1+8+9+9	-> 27	-> 2+7	-> 9			
6		23454678		3		2+3+4+5+4+ 6+7+8	-> 39	-> (3+9)-9	-> 3			
7		7865432989		7		7+8+6+5+4+3+2+9+8+9	-> 61	-> 6+1	-> 7			
8		9999999999		9		9+9+9+9+9+9+9+9+9	-> 90	-> 9+0	-> 9			
9		9999999999999999		9		9+9+9+9+9+9+9+9+9+9+9+9+9+9+9+9	-> 135	-> 1+3+5	-> 9			
10		999999999999998		4		9+9+9+9+9+9+9+9+9+9+5+9+9+9+8	-> 130	-> 1+3+0	-> 4			
11												
12		=LET(x,SUM(IFERROR(-MID(B2,SEQUENCE(LEN(B2),1),0)),y),SUM(-MID(x,SEQUENCE(LEN(x),1),IF(LEN(x)=1,x,IF(y>9,y-9,y))))										
13		Sum all of a number's digits to one digit only. This process can end up in one step, for example: 21 -> 2+1 -> 3 (a one-digit result). But sometimes you need 2 steps, for example: 1899 -> 1+8+9+9 -> 27 -> 2+7 -> 9 (a 2-digit result). Now you sum up the digits again: if the sum exceeds 9, you subtract 9 from the result.										
14												
15												
16												
17												
18												
19												
20												
21												

Figure 7.24: Digital root

First n odd numbers squared (A simple solution)

We can easily list the squares of the first n odd natural numbers (n is a parameter in cell G1) with this simple formula. In our example, we display the squares of the first 10 odd positive integers as shown in the following screenshot:

	A	B	C	D	E	F	G	H	I	J	
1		1				n	10				
2		9									
3		25									
4		49									
5		81									
6		121									
7		169									
8		225									
9		289									
10		361									
11											
12											
13		=SEQUENCE(G1,,1,2)^2									
14											
15											
16											
17		Generate the first n odd numbers squared									
18											
19											
20											

Figure 7.25: First n odd numbers squared (a simple solution)

First N odd numbers squared (A complex solution)

A more complex solution to the challenge presented in the previous section (see Figure 7.25) is manifested in the following figure. The **SEQUENCE(2*g1,,0)** creates a sequence of 20 numbers, starting from 0 up to 19. The multiplication of **BITOR** of the array with **BITXOR** of the array engenders an array of alternate odd and even numbers. What is interesting about this array is that the numbers in the odd locations of the array are the numbers we are looking for, that is, an array of squares of the first n odd natural numbers. All that is left for us to do is to get rid of the even numbers of the array by the **FILTER** function which includes only the odd numbers (numbers that leave a remainder of 1 when divided by 2) as shown in the following screenshot:

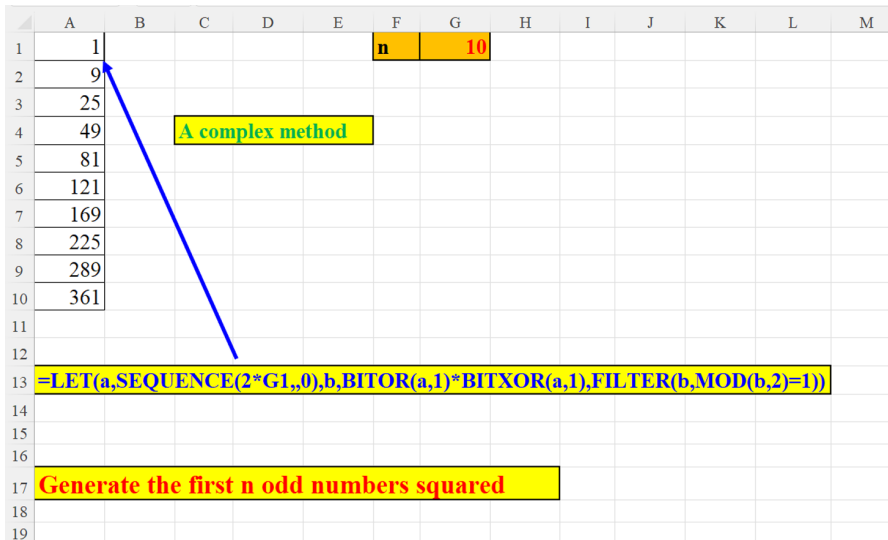


Figure 7.26: First n odd numbers squared (a complex solution)

Find first divisor of a number (divisor found)

In the following section we are going to demonstrate a method to find the first divisor of a number given in a parameter (cell G1). The divisor must be an integer between 2 and 10. We skip the divisor 1, since every number is divisible by 1. One can, of course, extend the range of divisors by changing the first argument of the **SEQUENCE** function (10) to any desired number.

We first find all the remainders of the corresponding divisors (in cells A2:A10), and then find that the first 0 (which means that the dividend (in G1) is divisible by that number without any remainder).

In our example, the first divisor of 150 is 2. The **MATCH** function finds the first 0 and then we add 1 to the result, since our first divisor was 2 and not 1 as shown in *Figure 7.27*.

The **SEQUENCE(9,,2)** returns the array: {2,3,4,5,6,7,8,9,10}

The **MOD(G1,SEQUENCE(9,,2))** returns the array of numbers in the range: A2:A10.

Each 0 in that range signifies the fact that the number (in cell G1) is divisible (with no remainder) by that number. Since our **SEQUENCE** array starts with 2 (because, as stated above, every number is divisible by 1), the first divisor of the number in cell G1 is 2. Therefore, we add 1 to the result of the **MATCH** function, which finds the first 0 in the array A2:A10:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1						dividend	150							
2		0	2											
3		0												
4		2												
5		0												
6		0												
7		3												
8		6												
9		6												
10		0												
11														
12														
13														
14														
15														
16														
17														
18														
19														

=IFERROR(MATCH(0,MOD(G1,SEQUENCE(9,,2)),0)+1,"not found")

We want to check whether the dividend (in G1) is divisible by 2,3,4,5,...10.
The result (C2) tells us the first divisor (remainder=0)
The first divisor in the SEQUENCE function is 2 because a number is always divisible by 1

Figure 7.27: Find first divisor of a number (example 1)

Find first divisor of a number (divisor not found)

This example is similar to the one shown in the preceding section. However, here we could not find a divisor in the defined range (2-10). In such cases, we need to cater to errors. Therefore, the result here is **not found**, that is, the dividend in cell G1 (151) is

not divisible by any number in the range defined within the **SEQUENCE** function: the 9 integers from 2 to 10:

	A	B	C	D	E	F	G	H	I	J	K	L	M
1						dividend	151						
2		1	not found										
3		1											
4		3											
5		1											
6		1				=IFERROR(MATCH(0,MOD(G1,SEQUENCE(9,,2)),0)+1,"not found")							
7		4											
8		7											
9		7											
10		1											
11													
12													
13													
14													
15													
16													

Figure 7.28: Find first divisor of a number (example 2)

Conclusion

In this chapter, we have described and presented many formulae that integrate the **SEQUENCE** function with the domain of mathematics. To name a few mathematical fields employed in the examples:

- Square roots
- Fractions
- Exponentiation
- Bit operations
- Trigonometry
- Geometric series
- Multiplication table
- Conversion of binary into decimal
- Matrix product of arrays
- Digital root

In the next chapter, we shall discuss the implementation of **SEQUENCE** in more complex challenges. For example, how to implement **SEQUENCE** in data validation of text (certain sequence of characters in an alphabet), how to use **SEQUENCE** in a more flexible method than the traditional nested IF, how to calculate the check digit of a

number using the Luhn algorithm and more. We will also demonstrate how to carry out Excel challenges with the **LAMBDA** function.

Points to remember

- **SEQUENCE** is essential when working with series, either arithmetic or geometric.
- It is very convenient to use it in trigonometric functions, especially if we want to unfold the visual manifestation of these functions.
- Multiplication of arrays (or matrix products of arrays) can be achieved by combining **SEQUENCE** with the **MMULT** function.
- Bit operation functions in tandem with **SEQUENCE** can produce results that might be difficult to achieve in alternative attitudes.
- **SEQUENCE** is also very handy in dealing with other mathematical *entities*: fractions, power operations, divisibility, and so on.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



CHAPTER 8

SEQUENCE and Other Animals

Introduction

This is the book's last chapter. It consists of some more complex examples of using **SEQUENCE** under various challenges in Excel. Some of them will explain in detail the use of the newest and most challenging functions in Excel 365: the **LAMBDA** function and some of its helper functions.

Structure

This chapter will demonstrate the application of new concepts with the assistance of **SEQUENCE**:

- Examples of **SEQUENCE** with other animals
 - Better than nested **IF**
 - Fetch the first and last digits from a string
 - Data validation – only Hebrew letters
 - Data validation – only uppercase English letters
 - Splitting cell by chunk size and separator (two examples)
 - Remove all digits from string

- Remove names and split numbers to separate cells
- Removing “A”, “B” and “C” from string – Two methods
- **INDEX-SQRT** instead of **FILTER**
- Remove all uppercase or lowercase letters from string
- Verifying the validity of a check digit with the Luhn algorithm

Objectives

The objective of this chapter is to show you the advantage of using **SEQUENCE** when dealing with complex operations. As you already know, since **SEQUENCE** is an array function, it can speed up your calculations by executing several solutions concurrently. Learning these techniques will undoubtedly save you a lot of time, and effort.

Examples of SEQUENCE with other animals

The following examples manifest the versatility of **SEQUENCE** in conjunction with other functions to effectuate dynamic solutions in complicated problems.

Better than nested IF

This section will demonstrate the difference between the old, *traditional* nested IF function and the more flexible and sophisticated combination of **XLOOKUP** and **SEQUENCE**. The example chosen is conversion of grade points (0-100) to marks (1-5).

Traditional solution – nested IF

We are all familiar with the IF function and its younger sister, IFS (introduced in Excel 2019). This function is perhaps the most used function in Excel, neck to neck with **SUM** and **VLOOKUP**. Here is an example of utilizing nested IF in translating grade points (numbers) to marks (scale of numbers from 1 to 5). The table on the left-hand side depicts the conversion rules to be implemented in the formula. One can clearly see that the nested **if** is both lengthy and inflexible, which makes it hard to debug as shown in the following screenshot:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	
1				Score	56													
2	Rules																	
3	Grade	Mark																
4	>80	5																
5	>60	4																
6	>40	3																
7	>20	2																
8	>0	1																
9																		
10				=IF(SES1>80,5,IF(SES1>60,4,IF(SES1>40,3,IF(SES1>20,2,IF(SES1>0,1,0))))))														
11																		
12																		
13																		
14																		
15				Calculate marks from grade points														
16				Nested IF Solution														
17																		
18																		
19																		

Figure 8.1: Converting grade points to marks (nested if)

A different attitude is presented in the next three examples. The implementation of the conversion rules is dynamic since both the gaps between the grades and the marks scale – are dynamic (in parameters): you can change the number of marks (see example 2) and see the impact. Of course, you can change the highest score and the result will adapt itself to the new *upper limit* of the grades (as can be seen in the third example).

XLOOKUP and SEQUENCE instead of nested IF (example 1)

First, we check to see that only valid entries were keyed in the score field (cell E1): a number not greater than the highest score (in this example it is 100, but we shall see in the next example a different *highest score* and how the flexibility of the formula caters for such cases).

XLOOKUP's first argument is, of course, the score to be *translated* (cell E1).

The second argument to the **XLOOKUP** function:

SEQUENCE(,K1,N1-(N1/K1)+1,-(N1/K1)) yields the array {1,21,41,61,81}

The expression: $(N1/K1)$ determines the size of each *scale*: we divide the highest grade (100, in cell N1) by the number of marks (5, in cell K1): $100/5 = 20$. So, each mark is equivalent to 20 grade points.

The third argument to the **XLOOKUP** function:

SEQUENCE(,K1,K1,-1) produces the array {5,4,3,2,1} according to the number of marks (cell K1).

XLOOKUP's fourth argument tells the formula what to do in cases of errors, like, if the number entered in E1 is negative.

The fifth and last argument in our formula (since we skip the sixth one) defines the match mode. Its value (-1) tells Excel to find the exact match or the next smallest item. In our case, since there is no exact match, Excel returns the last smallest match: 56 between the third and fourth members of the array that we created in the second argument of **XLOOKUP** ({1,21,41,61,81}): $41 < 56 < 61$. So, **XLOOKUP**'s result is: 3. (The table on the left-hand side is displayed only for clarification):

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1				Score	56					marks	5		highest	100				
2		Rules																
3		Grade	Mark															
4		>80	5															
5		>60	4															
6		>40	3															
7		>20	2															
8		>0	1															
9																		
10																		
11		=IF(AND(ISNUMBER(E1),E1<=N1),XLOOKUP(E1,SEQUENCE(,K1,N1-(N1/K1)+1,-(N1/K1)),SEQUENCE(,K1,K1,-1),0,-1),0)																
12																		
13																		
14																		
15																		
16																		
17																		
18																		
19																		

Figure 8.2: Converting grade points to marks (a more flexible attitude – example 1)

XLOOKUP and SEQUENCE instead of nested IF (example 2)

The same formula used in the previous example is implemented here. The only difference made is that we now want a scale of only four marks (1:4). The formula: `=IF(AND(ISNUMBER(E1), E1<=N1), XLOOKUP(E1, SEQUENCE(, K1, N1 - (N1/K1) + 1, -(N1/K1)), SEQUENCE(, K1, K1, -1), 0, -1,), 0)` applies this change effortlessly.

Here, the second argument to the **XLOOKUP** function is:

`SEQUENCE(, K1, N1 - (N1/K1) + 1, -(N1/K1))` which yields the array {1,26,51,76}

and the third argument to the **XLOOKUP** function is:

`SEQUENCE(, K1, K1, -1)` which produces the array {4, 3, 2, 1}

Since the score (56) falls between 51 and 76, **XLOOKUP** returns the corresponding mark: 3 (in the third location). (The table on the left-hand side is displayed for clarification):

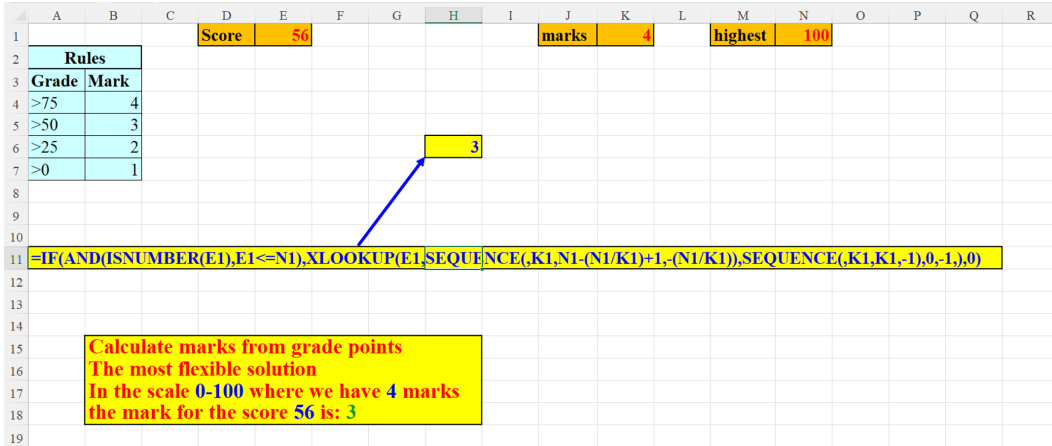


Figure 8.3: Converting grade points to marks (a more flexible attitude – example 2)

XLOOKUP and SEQUENCE instead of nested IF (example 3)

In our third and last example, we will change the highest possible score to 70. The formula adapts itself flawlessly to the new range. (The table on the left-hand side is displayed for clarification):

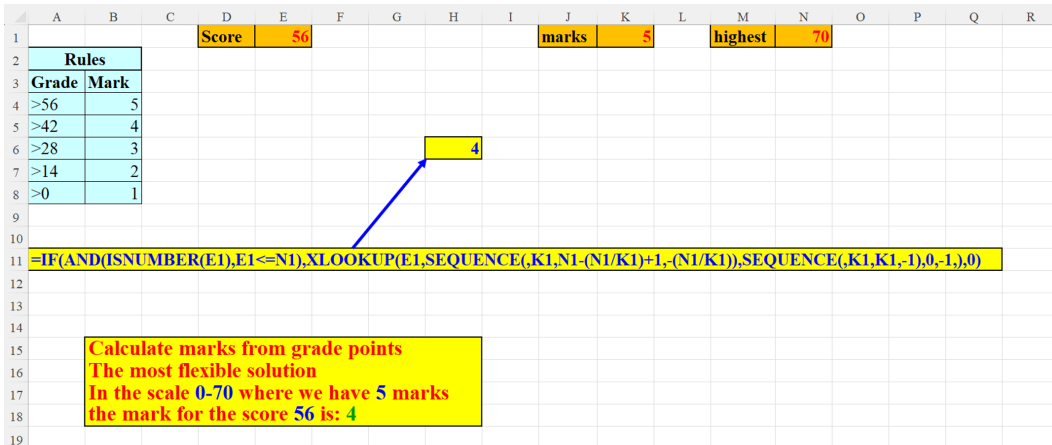


Figure 8.4: Converting grade points to marks (a more flexible attitude – example 3)

Fetch the first and last digits from a string

The formula depicted in cell B3 Figure 8.5 extracts the first and the last digits from the string in A3. If there are at least two digits in A3, the first and the last digits are extracted into cells B3 and C3, accordingly. If only one digit is found (for example,

in cell A7), cell C7 is left empty. If no digits were found (cell A5) , both A5 and C5 are empty.

Follow these steps for fetching the first and last digits from a string:

1. Remove all non-digits from the string. Only digits are extracted and concatenated by the **TEXTJOIN** function. (result: "0456")
2. A " " is suffixed to the first extracted digit (which is 0 and the result is "0 ")
3. If the length of the string created in *Step 1* is greater than 1 (it is), then the last digit of the string in *Step 1* is extracted (the digit: 6)
4. The result of *Step 2* is concatenated to the result of *Step 3* (result: "0 6")
5. **TEXTSPLIT** splits the string created in *Step 4* ("0 6") by the delimiter (" "), thus returning the final result in cells B3 and C3:

String	First Digit	Last Digit
abc04d#e56er	0	6
q%w67ly	6	7
Excel		
nineteen84	9	4
Eigh8*	8	

=LET(a,TEXTJOIN("","IFERROR(-MID(A3,SEQUENCE(LEN(A3),1),1),"")),TEXTSPLIT(MID(a,1,1)&" "&IF(LEN(a)>1,MID(a,LEN(a),1),"")," "))		
Extracting only first and last digit within a string		

Figure 8.5: Fetch the first and the last digits from a mixed string

Data validation – only Hebrew letters

The technique used here allows only Hebrew letters in the data validation cell (B1). More than that, we can limit the range of allowed letters. If you enter in B1, any letter greater than the upper limit (parameter in cell G1) or smaller than the lower limit (parameter in F1), the result is **FALSE** as shown in the following screenshot:

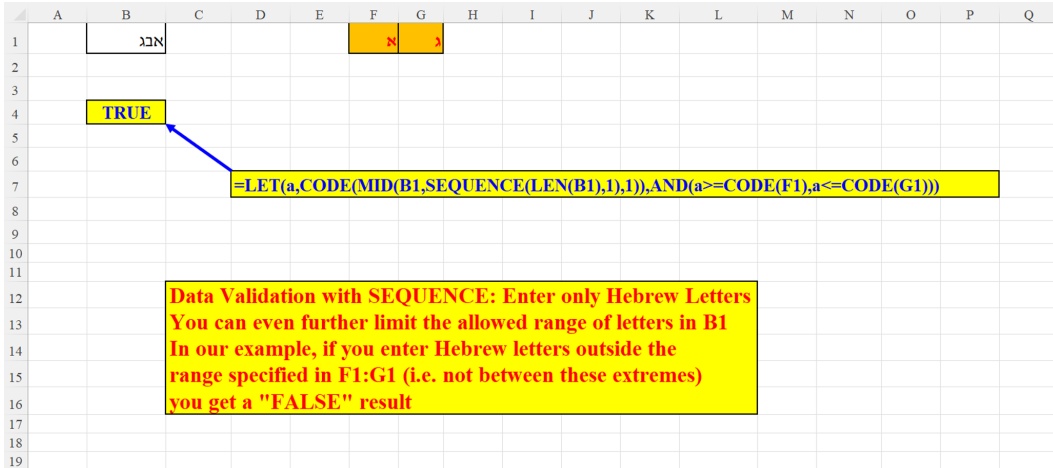


Figure 8.6: Data validation – only Hebrew letters

The rule applied in the data validation can be seen in the following figure. The **CODE** function converts the ASCII character into its numeric value, so that every character is checked to make sure it is valid. The full formula can be seen in *Figure 8.6*:

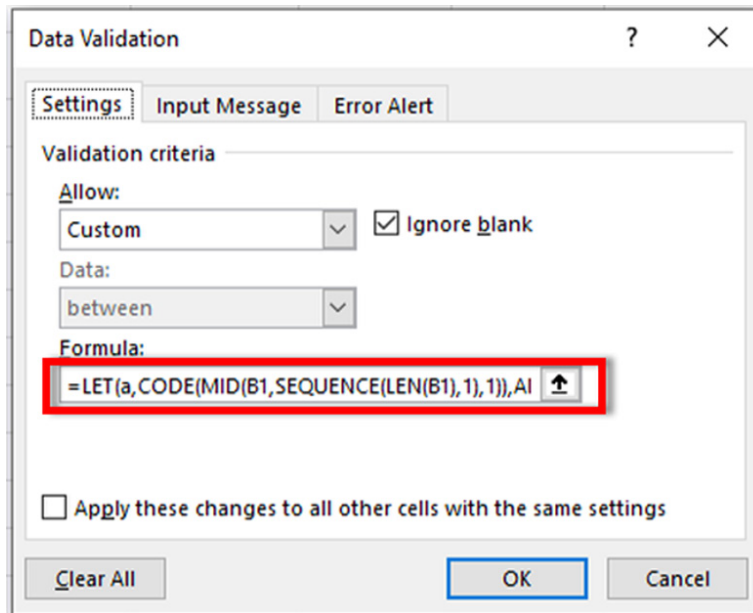


Figure 8.7: Data validation – only Hebrew letters (the rule)

Data validation – only uppercase English letters

This solution is similar to the one applied in the previous example. Here, we will check that only uppercase English letters are entered in cell B1. Again, we employ the same rule: the letters entered should not be below “A” (parameter in F1) or above “Z” (parameter in G1). If you enter letters outside this range, the formula returns: **FALSE**.

As in the previous example, we display the rule defined in the data validation pane. The full formula can be seen in the following figure (D7):

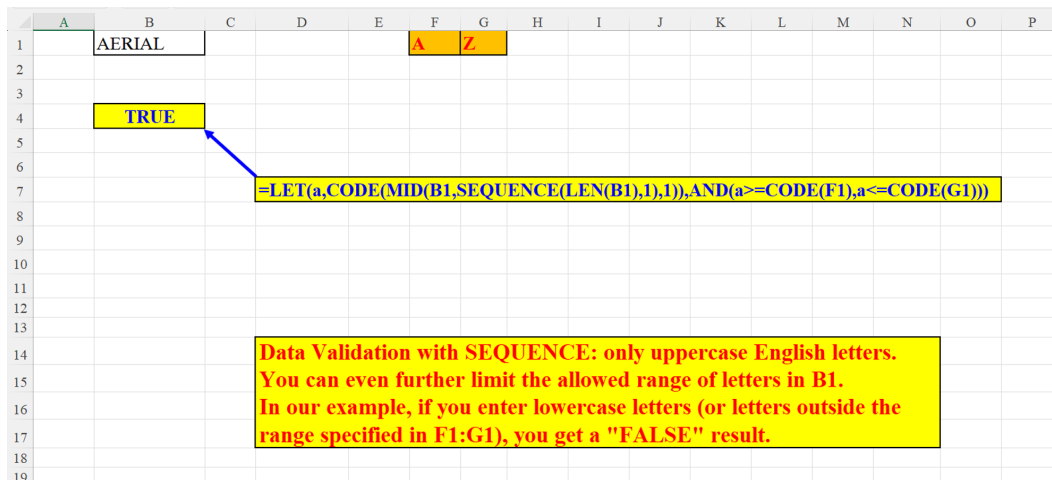


Figure 8.8: Data validation – only uppercase English letters

The full formula can be seen in Figure 8.8 (cell D7). The following screenshot depicts the Data Validation rule applied:

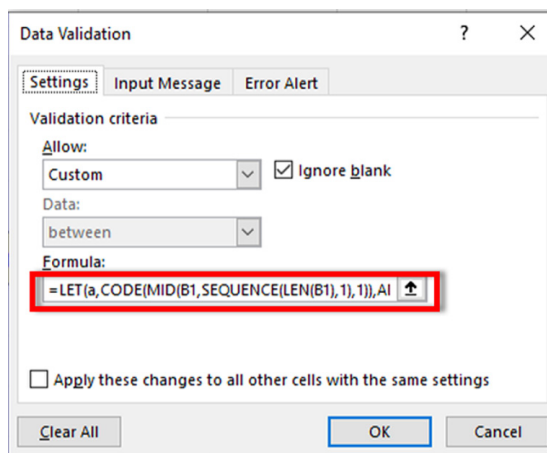


Figure 8.9: Data validation – only uppercase English letters (the rule)

Splitting cell by chunk size and separator (two examples)

Suppose we have a string in a cell, and we want to split it by a separator into equal “chunks”. The chunk size is defined in cell G1, and the separator is defined in cell I1. The technique is demonstrated in the next figure.

Explanation of the formula can be implemented through the following steps:

1. The **TEXTJOIN** function, which *wraps* the **MID** function creates the following string: "AB/CD/EF/GH/I/////".
2. The **FIND** function searches for the location of A3's last character ("I") within the result in *Step 1* and returns: 13. This is the length of the string we are looking for: the string created by the **TEXTJOIN** without the additional separator characters at the end.
3. The **LEFT** function returns only the first 13 characters of cell A3. In cell C4 we can see a practical utilization of this technique: the number in cell A4 is converted to: 12/02/20 which is a valid date. If we multiply C4 by 1 (to convert the string to a number), the result (in E4) will be a valid Excel date:

	A	B	C	D	E	F	G	H	I	J	K
1						Chunk Size	2	Separator	/		
2	Before		After								
3	ABIDEFGHI		AB/ID/EF/GH/I								
4	120220		12/02/20		43873						
5	TicTacToer		Ti/cT/ac/To/er								
6	abcdefgh		ab/cd/ef/gh								
7	abcdefghijk		ab/cd/ef/gh/ij/k								
8											
9											
10	=LET(x,TEXTJOIN("",".",MID(A3,SEQUENCE(LEN(A3))*SGS1-(SGS1-1),SGS1)&\$IS1),LEFT(x,FIND(RIGHT(A3),x,LEN(A3))))										
11											
12											
13											
14											
15	Split cell by chunk size and separator										
16											
17	Chunk size:	After how many characters should the separator be implemented									
18	Separator:	One character that separates between chunks									
19											
20	This formula doesn't add a separator as a last character										
21											

Figure 8.10: Split cell by chunk size and separator (example 1)

Another example demonstrates the flexibility of the solution. Here, the separator is composed of two characters, not just one:

	A	B	C	D	E	F	G	H	I	J	K
1						Chunk Size	3	Separator	//		
2	Before		After								
3	ABIDEFGHI		ABI//DEF//GHI								
4	120220		120//220								
5	TicTacToeer		Tic//Tac//Toe//r								
6	abcdefgh		abc//def//gh								
7	abcdefghijk		abc//def//ghi//jk								
8											
9											
10	=LET(x,TEXTJOIN("","",MID(A3,SEQUENCE(LEN(A3))*SG\$1-(SG\$1-1),SG\$1)&\$I\$1),LEFT(x,FIND(RIGHT(A3),x,LEN(A3))))										
11											
12											
13											
14											
15	Split cell by chunk size and separator										
16											
17	Chunk size:	After how many characters should the separator be implemented									
18	Separator:	Two characters that separate between chunks									
19											
20	This formula doesn't add a separator as a last character										
21											

Figure 8.11: Split cell by chunk size and separator (example 2)

Remove all digits from string

This solution combines two new functions: **LAMBDA** and a helper function, **REDUCE**.

The **REDUCE** is an *ancillary* function, which means that it cannot operate independently. It can be implemented only in conjunction with **LAMBDA**.

The **SEQUENC(10,,0)** creates an array of the digits: 0-9. The **LAMBDA** function then removes these digits from the string, replacing them with null by using the **SUBSTITUTE** function:

	A	B	C	D	E	F	G	H	I	J
1	Before		After							
2	123abc123		abc							
3	v1c2d3r5y78u		vcdryu							
4										
5										
6										
7										
8	=REDUCE(A2,SEQUENCE(10,,0),LAMBDA(x,y,SUBSTITUTE(x,y,"")))									
9										
10										
11										
12										
13	Remove all digits from string									
14										
15										
16										
17										

Figure 8.12: Remove all digits from string

Remove names and split numbers to separate cells

The main two functions (**LAMBDA** and its **helper** function: **REDUCE**) create a *stack* of all ASCII non-numeric characters (1:47, 58:255):

The **CHAR(SEQUENCE(47,,1))** creates an array of all the ASCII characters preceding the first digit (**CHAR(48)** which is 0) and all the ASCII characters following the last digit (**CHAR(57)** which is 9): **CHAR(SEQUENCE(198,,58))**.

Then, all these characters are removed from the original string and are replaced by " ".

We are left with only numbers, separated by spaces. The **TEXTSPLIT** then splits these numbers by the separator (" "), which leaves us with the three numbers as shown in the following screenshot:

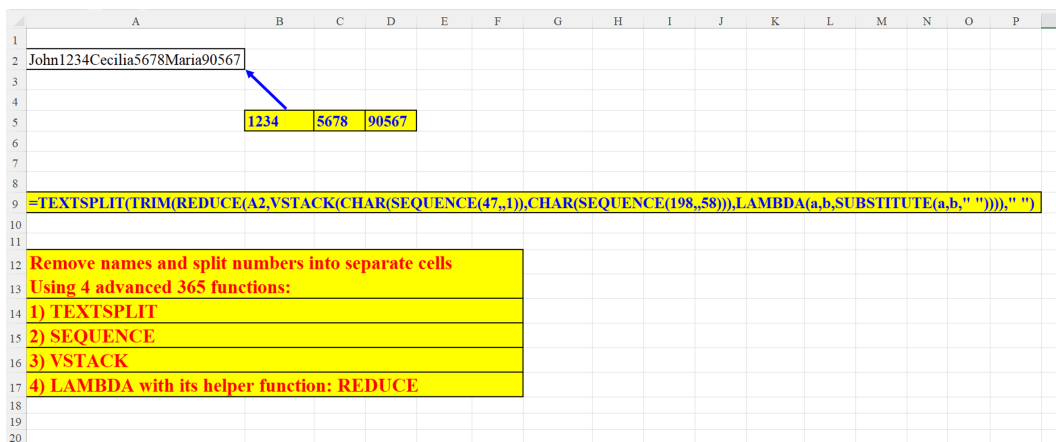


Figure 8.13: Remove names and split numbers into separate consecutive cells

Removing “A”, “B” and “C” from string – two methods

The idea here is to remove the first three uppercase English letters from a string.

This solution combines two new functions: **LAMBDA** and a helper function: **REDUCE**.

REDUCE is an *ancillary* function, which means that it cannot operate independently. It can be implemented only in conjunction with **LAMBDA**.

Method 1

In this solution, the values to be removed are hard coded within the formula, which is not an ideal solution as shown in the following screenshot:

	A	B	C	D	E	F
1	Before	After				
2	ABC123	123				
3	AB123CC	123				
4	124ABCCA	124				
5	98A47B509C	9847509				
6	888A	888				
7						
8						
9						
10		=REDUCE(A2,{"A","B","C"},LAMBDA(x,y,SUBSTITUTE(x,y,"")))				
11						
12						
13						
14	Remove all "A","B","C" from string					
15	"hard-coded" within the formula					
16						
17						
18						
19						

Figure 8.14: Removing "A", "B", "C" from string – method 1

Method 2

This is a better implementation of the problem posed in the previous figure. Instead of hard-coded values, we use two parameters: the first one specifies the first letter to remove, and the second defines the number of alphabetic characters to remove. In our case we have three, which means that the formula is supposed to eliminate the first three lowercase alphabet letters.

This solution also uses **LAMBDA** and **REDUCE**.

It accepts two arguments:

1. The first one is the initial value (in our case, it is the entire string in A2) and
2. The second is the expression: **CHAR(SEQUENCE(\$J\$1,,CODE(\$E\$1),1)** which yields the array of three characters: "a";"b";"c"

These are *fed* into the **LAMBDA** function which removes the occurrence of any character of the second argument from the first argument.

The result: A string *stripped off* from the letters specified in the parameters.

This solution is dynamic, because:

- You can start with any letter you want
- You may specify any number of consecutive letters, starting with the letter defined in the first argument.

Since we start with *a*, the formula will remove the first three lowercase letters, as shown in the following screenshot:

	A	B	C	D	E	F	G	H	I	J	K
1	Before	After		starting from	a		number of characters to remove	3			
2	abd123	d123									
3	ab123cc	123									
4	cd124abcca	d124									
5	98aA47bB509cC	98A47B509C									
6	888A	888A									
7											
8											
9											
10	=REDUCE(A2,CHAR(SEQUENCE(\$J\$1,,CODE(\$E\$1),1)),LAMBDA(x,y,SUBSTITUTE(x,y,"")))										
11											
12											
13											
14	Remove n first lowercase/uppercase letters using parameters for a flexible solution.										
15	You can select both the first letter to eliminate and also the number of consecutive letters.										
16											
17											
18											
19											
20											

Figure 8.15: Removing three first lowercase letters from string – method 2

INDEX-SQRT instead of FILTER

In cells A3:C10 we have a data set of songs and composers. Each combination of song and composer has one unique SKU. In cells G3:G10, we need to find the SKU that matches the song (in E3:E10) and the composer (F3:F10).

Of course, the simplest method would be using the **FILTER** function:

	A	B	C	D	E	F	G	H	I	J	K
1											
2	Song	Composer	SKU		Song	Composer	SKU				
3	Peace	John Lennon	10123		Garden	John Denver	not found				
4	Peace	John Denver	10124		Longing	Louis Armstrong	20343				
5	Love	John Lennon	10125		Peace	John Denver	10124				
6	Love	Shania Twain	10126		Love	John Lennon	10125				
7	Garden	Shania Twain	10127		Love	Shania Twain	10126				
8	Garden	Keith Jarret	10128		Garden	Shania Twain	10127				
9	Longing	Louis Armstrong	20343		Longing	John Denver	32038				
10	Longing	John Denver	32038		Peace	John Lennon	10123				
11											
12											
13											
14	=IFERROR(FILTER(SCS3:SCS10,(E3=\$A\$3:\$A\$10)*(F3=\$B\$3:\$B\$10),"not found"))										
15											
16											
17	using FILTER to get the SKU (in column C) for the song and composer in (columns E, F respectively)										
18											
19											

Figure 8.16: Using **FILTER** to get the SKU for the song and composer (columns E,F)

But, we have devised an alternative solution which uses the **SEQUENCE** function in tandem with **INDEX** and **SQRT**. You might wonder that **SQRT** is a mathematical function that has nothing to do with search functions. How is that possible?

Let us check it first on cell **G3** and then on cell **G4**.

In cell **G3**, the product: **(\$A\$3:\$A\$10=E3)*(\$B\$3:\$B\$10=F3)** yields 0. This happened because in the original data set, there is no SKU for the song *Garden* whose composer is *John Denver*. So, the formula returns: *not found*.

In cell **G4**, the product: **(\$A\$3:\$A\$10=E4)*(\$B\$3:\$B\$10=F4)** yields 1, because the combination of the song *Longing* composed by *Louis Armstrong* was found in column **C**. We need to find on which row (within the range: **C3:C10**) this combination exists. The expression:

SQRT(SUM(a^2*s)) is the crux of the formula.

a^2 is the expression. **SEQUENCE(8)** raised to the second power, produces the following array: {1;4;9;16;25;36;49;64}. This element is multiplied by **(\$A\$3:\$A\$10=E4)*(\$B\$3:\$B\$10=F4)** whose result is: {0;0;0;0;0;1;0} because this combination was found on the seventh row. The sum of these two arrays is: 49. Now, what is left for us to do is find the **SQRT** of the number: 7. So, the **INDEX** function fetches the 7th element in **C3:C10**, which is: 20343. The **SQRT** serves here as a replacement for **MATCH** as shown in the following screenshot:

Song	Composer	SKU	Song	Composer	SKU
Peace	John Lennon	10123	Garden	John Denver	not found
Peace	John Denver	10124	Longing	Louis Armstrong	20343
Love	John Lennon	10125	Peace	John Denver	10124
Love	Shania Twain	10126	Love	John Lennon	10125
Garden	Shania Twain	10127	Love	Shania Twain	10126
Garden	Keith Jarret	10128	Garden	Shania Twain	10127
Longing	Louis Armstrong	20343	Longing	John Denver	32038
Longing	John Denver	32038	Peace	John Lennon	10123

Formula in cell G4: **=LET(a,SEQUENCE(COUNTA(SAS3:SAS10)),s,(\$AS3:\$AS10=E3)*(\$BS3:\$BS10=F3),IF(SUM(--s),INDEX(\$CS3:\$CS10,SQRT(SUM(a^2*s)),),"not found"))**

INDEX-SQRT instead of FILTER

Figure 8.17: INDEX-SQRT instead of FILTER

Remove all uppercase or lowercase letters from string

This solution combines the **LAMBDA** and **REDUCE** functions in a similar manner to the examples in *Figure 8.11* to *Figure 8.14*.

The **REDUCE** function accepts two arguments:

- The first one is the initial value (in our case, it is the entire string in A2).
- The second argument is the expression:

CHAR(SEQUENCE(26,\$E\$2,1) which generates all the uppercase English letters (if cell E1 is "A") or all the lowercase letters (if cell E1 contains "a").

The **LAMBDA** function then removes every occurrence of the 26 uppercase letters starting from "A", the first argument in cell E1. The result string in cell A2 is "stripped off" from its uppercase letters.

This solution is dynamic, because you can choose whether to get rid of only the uppercase letters or the lowercase letters too as shown in the following screenshot:

	A	B	C	D	E	F	G	H	I
1	Before	After		Letter	A				
2	ABC1D2E3F	123		ASCII Code	65				
3	AB12J3CMNC	123							
4	124ABCCA	124							
5	98A47B50K9TC	9847509							
6	8ZZZ88A	888							
7									
8									
9									
10									
11									
12									
13									
14									
15									
16									
17									
18									
19									
20									

=REDUCE(A2,CHAR(SEQUENCE(26,SE\$2,1)),LAMBDA(x,y,SUBSTITUTE(x,y,"")))
Remove all uppercase English letters from string Using LAMBDA
If you want to remove only lowercase English letters change cell E1 from: A to: a

Figure 8.18: Remove all uppercase or lowercase letters from string

Verifying the validity of a check digit with the Luhn algorithm

The chapter's last example will demonstrate the implementation of the Luhn algorithm to verify the check digit of a 9-digit number, whose last digit is the check digit.

The algorithm is computed as follows:

Each even-place digit is multiplied by 2. If the result is greater than 9, we add the two digits that make up the result. For example, if the original digit was 6 and we multiplied it by 2, the result is 12. Since this number is greater than 9, we add the two digits: 1+2=3.

No calculation is executed on the odd-place digits. At the end of this computation, we sum up all the results. If the result is divisible by 10 without any remainder, then the check digit (9th digit) is correct as shown in the following screenshot:

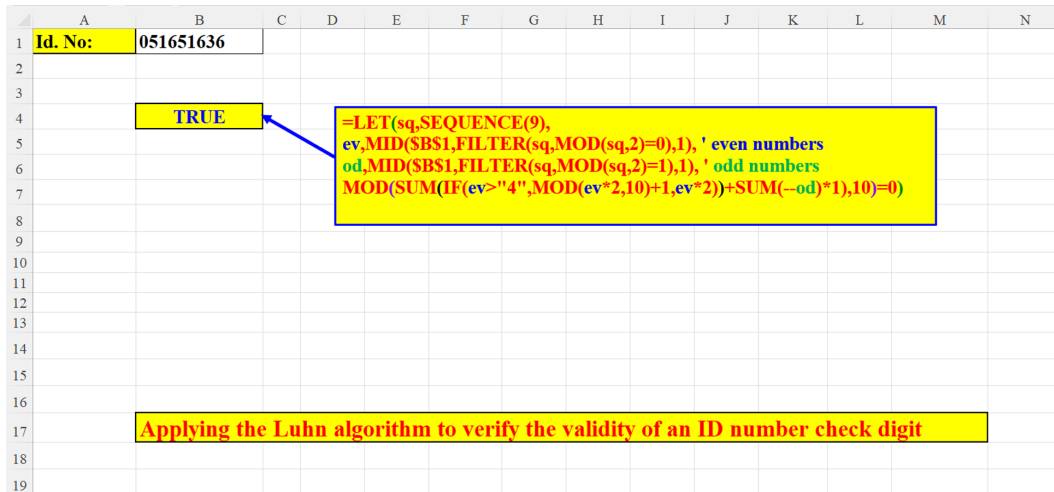


Figure 8.19: The Luhn algorithm applied to verify the check digit of the ID no. in cell B1

The next figure will explain the three steps of the process:

1. Extracting the even-place digits (2nd, 4th, 6th and 8th digits) (in C4#) and the odd-place digits (1st, 3rd, 5th, 7th, 9th digits) (in C12#)
2. Each even-place digit is multiplied by 2 (see explanation above). If any digit in the array of the even-place digits is greater than 4, then the product will be 10 or more. In such cases, we add 1 to the remainder of division of this number to 10 (which is the same as adding these two digits).

For example, if the original digit was 6 and we multiplied it by 2, the result is: 12. Since this number is greater than 9, we add the two digits: $1+2=3$.

For example: 5, the first even-place digit, is greater than 4. so, we multiply it by 2 ($5*2=10$), find the remainder of division by 10 ($=0$, no remainder) and then add 1. ($0+1=1$).

Another example: 6, the second even-place digit, is also greater than 4. So, we multiply it by 2 ($6*2=10$), find the remainder of division by 10 ($=2$) and then add 1. ($2+1=3$). Then, we sum the 4 results of all 4 even-place digits ($1+3+2+6=12$).

The odd-place digits are *summed* without calculation: $(0+1+5+6+6)=18$.

3. We now *combine* the 2 sums: $12+18=30$ and divide the total sum (30) by 10. If there is no remainder, then the number is a valid ID number according to the Luhn algorithm:

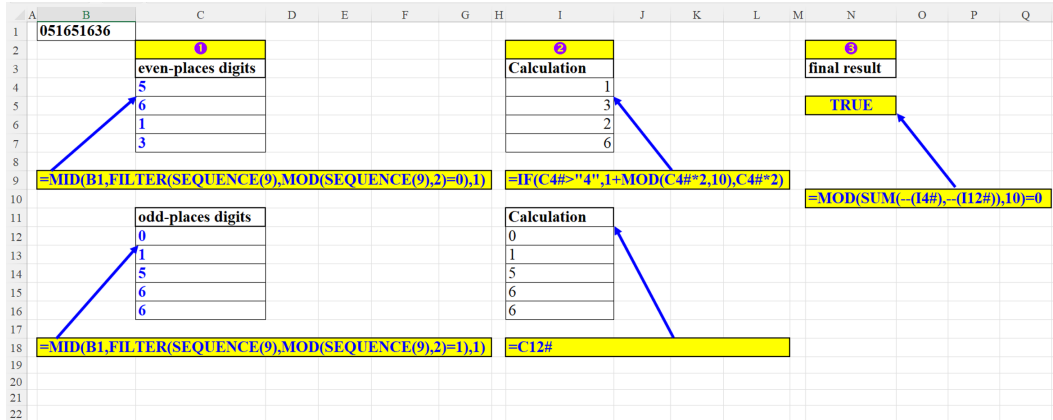


Figure 8.20: The three steps of the algorithm implemented in the formula

Conclusion

In this chapter you learnt some advanced uses of **SEQUENCE** with other Excel 365 functions (for example **LAMBDA** with its helper functions: **SCAN** and **REDUCE**). Besides that, some sophisticated solutions with **SEQUENCE** (for example, the use of **INDEX-SQRT** instead of **FILTER**, or the implementation of the Luhn algorithm to verify an ID no, check digit's validity – in one formula) were demonstrated.

In this book, we covered a huge scope of the **SEQUENCE** function in various Excel categories: text, numbers, arrays, date and time, finance, and mathematics.

You will best benefit from this book if you read it thoroughly and practice the examples that might be of interest to you.

You must travel over a rough road to reach the stars.

Points to remember

- **SEQUENCE** works seamlessly with the new **LAMBDA** function.
- It is easily applied in data validation formulas, where you need to verify a sequence of characters within a range.
- It is crucial when separating numbers from text or when splitting strings by predefined rules.
- Combining **SEQUENCE** with other Excel functions enables us to create more flexible solutions.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



Index

A

ARRAYTOTEXT function 11

C

CHOOSECOLS function 13

CHOOSEROWS function 14

classic monthly calendar

about 137, 138

versus non-classic monthly
calendar 137

COLUMNS function 90

conditional formatting 153

D

digits

extracting, from string 34, 35

doctor's schedule

versions 136, 137

DROP function 18

duplication factor

method 1 31

method 2 32

used, for duplicating cell 31

Dynamic Array Functions (DAF)

about 3

ARRAYTOTEXT function 11

CHOOSECOLS function 13

CHOOSEROWS function 14

DROP function 18

examples 3

EXPAND function 11

FILTER function 4

HSTACK function 10

RANDARRAY function 6

SEQUENCE function 7

SORTBY function 6

SORT function 5
TAKE function 17
TEXTAFTER function 12, 13
TEXTBEFORE function 12
TEXTSPLIT function 7
TOCOL function 8
TOROW function 9
UNIQUE function 4
VALUETOTEXT function 18
VSTACK function 9
VSTACK function,
 versus TOCOL function 10
WRAPCOLS function 15
WRAPROWS function 14
XLOOKUP function 15, 16
XMATCH function 17

E

EXPAND function 11

F

FILTER function 4
Flip vertical array 102

H

horizontal flip columns
 about 100
 methods 100, 101
HSTACK function 10

L

list of month's days
 bad attitude 142
 creating 142
 good attitude 142, 143

M

MonCal
 defining 134
MonCal Named Range 132
monthly calendar language
 about 139
 formulas 140
 list of languages and formats 140
 list of month numbers 141
 parameters 139

N

N consecutive dates
 methods 118, 119
nested IF function
 example 201-203
 traditional nested IF function 200, 201
Net Present Value (NPV) 167, 168
non-classic monthly calendar
 about 139
 versus classic monthly calendar 137
number of lower-case letters
 methods 44

R

RANDARRAY function 6
RATE function 170
reverse string
 methods 40, 41
RRI function 171, 172

S

SEQUENCE
 descending 68, 69
SEQUENCE function 7, 90

- SEQUENCE in math operations
- array of duplicate numbers, generating
 - by bit operations 191
 - bit operation, using 184
 - complex solution 194
 - digital root 192
 - divisor found 194, 195
 - divisor not found 195, 196
 - dynamic multiplication table 188
 - dynamic quadratic equation 182, 183
 - dynamic sine, with spin
 - buttons 185-187
 - examples 178
 - exponential growth example 187
 - missing values, filling
 - in geometric series 190
 - MMULT function 188
 - MMULT function, using 192
 - number of candles 184
 - OR method 185
 - POWER function 179, 180
 - SEQUENCE function 188
 - SEQUENCE function, using 192
 - sequence of alternate
 - 1's and 0's, creating 182
 - sequence of fractions, generating 181
 - sequence of square roots, creating 180
 - simple solution 193
 - SUMPRODUCT function,
 - with SEQUENCE 189
 - trigonometric function,
 - with SEQUENCE 190
 - virtual array 183
- SEQUENCE (ROW())
- about 78
 - n integers, creating 78
 - sum of virtual array, creating 79
- SEQUENCE with arrays
- active months 99
 - array of ascending
 - numbers, creating 97, 98
 - array of identical numbers, creating 97
 - columns by parameters, selecting 110
 - data, converting 107, 108
 - dynamic array, building 100
 - examples 96
 - first of each month 115
 - flexible LARGE 104
 - Flip vertical array 102
 - Flip vertical array, with parameter 102
 - horizontal flip columns 100
 - last day of each month 116, 117
 - MMULT function,
 - with dynamic arrays 104
 - MMULT function,
 - with static ranges 104
 - multiple results, fetching
 - for search value 111
 - one cell, to vertical array 98
 - two-dimensional array,
 - creating with parameters 103
 - two-dimensional arrays, generating
 - by two parameters 109
 - vertical array, converting into
 - horizontal array without TRANSPOSE 112
 - vertical array, transposing
 - without size 110
 - VLOOKUP tricks 105
- SEQUENCE with date and time
- by month 145

by week 145, 146
class by month 125
classic monthly calendar,
 versus non-classic calendar 137
conditional formatting 144, 153
dates of Wednesdays, displaying 128
doctor's schedule 136
dynamic yearly calendar formula 144
eligibility days 135
examples 115
flexible method 157-159
horizontal SEQUENCE of descending
 dates 130
last date for each month 129, 130
list of month's days, creating 142
minutes to time, adding 121
monthly calendar language 139
month names, displaying
 without specific date 119, 120
N consecutive dates 118
NETWORKDAYS.INTL function
 for any period 133
NETWORKDAYS.INTL function
 for month 131, 132
NETWORKDAYS.INTL function
 without weekends 133, 134
NETWORKDAYS.INTL function
 with weekends 133, 134
new method 124
number of Monday 125, 126
number of Saturday 126, 127
number of Wednesdays,
 displaying 128
number of working days 134, 135
seconds to time, adding 121
SEQUENCE of dates 122

sequence of days 122
traditional methods 123, 124
weekday names, displaying 120
yearly calendar 143
yearly calendar, with conditional
 formatting 152
yearly horizontal calendar 146
yearly vertical calendar,
 with highlighted weekday 149
SEQUENCE with financial functions
 depreciation function in Excel 162
 example 156
 flexible method 160-162
 loan installment, impacting 163-167
 loan return by payments
 per period 156
 Net Present Value (NPV) 167
 PDURATION function 168-170
 RATE function 170
 RRI function 171-173
 SEQUENCE 173, 174
 SUM 173, 174
 sum of money, dividing equally 163
SEQUENCE with numbers
 array of 1's and 0's 80
 chessboard, building 91, 92
 Dynamic frequency based,
 on dynamic bins 89
 Dynamic SEQUENCE 81
 even numbers 75
 examples 63
 highest score subject 87, 88
 horizontal ascending array,
 reversing 74
 horizontal cell, duplicating 70

- horizontal descending array,
 - reversing 74
- horizontal numbers,
 - reversing by parameter 86
- methods, extracting
 - from string end 81, 82
- methods, extracting
 - from string start 82, 83
- missing numbers, finding in list 73
- N-digit number, creating 93
- negative integer methods,
 - generating 66-68
- N largest numbers (Ascending),
 - finding 84
- N largest numbers (Descending),
 - finding 84
- number of columns 85
- number of digits 85, 86
- number, reversing 73
- numbers, duplicating 71
- odd numbers 75
- order of SEQUENCE of numbers,
 - reversing 87
- positive integer
 - methods, generating 63-65
- SEQUENCE based, on number
 - of unique values 88
- SEQUENCE column 90
- SEQUENCE(ROW()) 78
- sum of digits 76
- sum of digits and text 76
- sum of largest N numbers 77
- sum of Nth row 77
- sum of smallest N numbers 78
- SUM SEQUENCE 79, 80
- vertical cell, duplicating 70
- vertical SEQUENCE numbers,
 - creating 71, 72
- SEQUENCE with other animals
- cell, splitting by chunk size
 - and separator 207, 208
- check digit validity, verifying with
 - Luhn algorithm 213, 214
- digits, removing from string 208
- examples 200
- FILTER function, using 211, 212
- first and last digits, fetching
 - from string 203, 204
- Hebrew letters, in data
 - validation 204, 205
- lowercase letters,
 - removing from string 212, 213
- names and split numbers,
 - removing to separate cells 209
- nested IF function 200
- uppercase English letters,
 - in data validation 206
- uppercase English letters,
 - removing from string 209-211
- uppercase letters,
 - removing from string 212, 213
- SEQUENCE with text
- 10 highest-paid employees, finding 24
- cell, duplicating by duplication
 - factor 31
- character, removing from string 43
- country names, extracting 51
- Diacritics, removing
 - from Hebrew words 52
- digits, extracting 43
- digits, extracting from string 34, 35
- digits, removing from string 39

- English uppercase, in cell 30
- English uppercase, in column 29
- English uppercase letters, creating
 - without number of letters 32
- examples 24
- first name, moving from end of cell 40
- Gematria in English 56
- Greek letters, generating in formula 45
- Hebrew Gematria (Formula) 50
- Hebrew Gematria (translation table 50
- Hebrew letters, converting
 - into English letters 53, 54
- horizontal characters, extracting 28
- increasing, from first character 49
- last word, finding in cell 45, 46
- letters, extracting from formula 55
- letters, extracting
 - from validation list 55
- non-digits, extracting from string 58
- Nth item description, fetching 54
- number of characters,
 - without separator 47
- number of lower-case letters 44
- number of non-empty cells,
 - in column 47
- number of words, in cell (version 1) 25
- number of words, in cell (version 2) 25
- number of words, in cell
 - without SEQUENCE 42
- number of words, in cell
 - without TEXTSPLIT 42
- numbers, splitting from text 36
- palindrome 38
- palindrome (Arabic) 53
- parameter string, defining 52
- range of words 57

- reverse string 40
- separator, adding 43
- sequence of characters, duplicating 30
- sorting, in alphabetical order 41
- string, defining 37
- string, in cell method 1 26
- string, in cell method 2 26
- string, in cell method 3 27
- string, in cell method 4 27
- string, in cell method 5 28
- strip leading 48
- text, increasing from last character 49
- trailing digits 48
- Unicode value, finding in string 58
- unique Alphabetic characters,
 - extracting from string 35
- unwanted characters, removing
 - from formula 37
- unwanted characters, removing
 - from named ranges
 - as parameters 36
- vendor, adding to formula 39
- vendor, adding to table 38
- vertical characters, extracting 29
- vertical list, converting into horizontal
 - list without TRANSPOSE
 - function 33
- weekday names, extracting 33

SORTBY function 6

SORT function 5

T

TAKE function 17

TEXTAFTER function 12, 13

TEXTBEFORE function 12

TEXTSPLIT function 7

TOCOL function 8

versus VSTACK function 10

TOROW function 9

U

UNIQUE function 4

V

VALUETOTEXT function 18

VLOOKUP tricks

about 105

columns, fetching for searched
item 105

data per lookup key, fetching
in reverse order 105, 106

first and third data items
per lookup key, fetching 107

last two columns, fetching
for search key 106

VSTACK function

about 9

versus Tocol function 10

W

WRAPCOLS function 15

WRAPROWS function 14

X

XLOOKUP function 15, 16

XMATCH function 17

Y

yearly horizontal calendar

Conditional Formatting calendar 148

Conditional Formatting weekday
names 148, 149

example 147

with highlighted weekday 146

yearly vertical calendar

Conditional Formatting calendar 151

Conditional Formatting weekday
names 151, 152

example 150

with highlighted weekday 149

Mastering SEQUENCE

DESCRIPTION

The SEQUENCE function in Excel 365 allows you to generate sequences of numbers or values based on specific criteria. By utilizing this function, you can effectively manage a wide range of numerical and data manipulation tasks. If you're looking to leverage its dynamic capabilities to enhance your productivity, this book is an ideal resource for you.

This book provides the most comprehensive coverage of the SEQUENCE function, which is widely considered the most versatile function in Excel 365. It serves as a detailed introduction to the new Dynamic Array Functions, offering examples for a better understanding of this new revolutionary concept. Additionally, the book delves into the extensive applications of SEQUENCE in various areas, including text functions, number manipulation, arrays, date and time operations, financial calculations, math, and complex formulae involving SEQUENCE, with a special focus on the super-function: LAMBDA. With over 200 examples, this book allows you to actively engage and explore the multifaceted dynamism of the SEQUENCE function.

By the end of the book, you will be able to confidently apply the SEQUENCE function in your own Excel workflows, enhancing your productivity and efficiency.

KEY FEATURES

- Unleash the power of SEQUENCE to simplify complex array calculations and automate repetitive tasks.
- Discover techniques to efficiently perform calculations, text manipulation, financial and numerical analysis using SEQUENCE.
- Learn how to integrate SEQUENCE with other Excel functions and tools.

WHAT YOU WILL LEARN

- Explore advanced techniques to enhance text-based analysis using SEQUENCE.
- Understand how SEQUENCE can generate dynamic arrays with custom patterns, sizes, and dimensions.
- Learn how to use the LAMBDA function to solve complex calculations
- Gain insights into using SEQUENCE to streamline financial modeling and forecasting
- Get tips and tricks to optimize date and time-related calculations using SEQUENCE.

WHO THIS BOOK IS FOR

This book is for everyone who is eager to explore innovative approaches in Excel, expand their knowledge, and improve their problem-solving skills. It serves as a valuable resource, offering a wide range of techniques that you can apply to enhance their Excel proficiency.



BPB PUBLICATIONS

www.bpbonline.com

ISBN 978-93-5551-854-5



9 789355 1518545