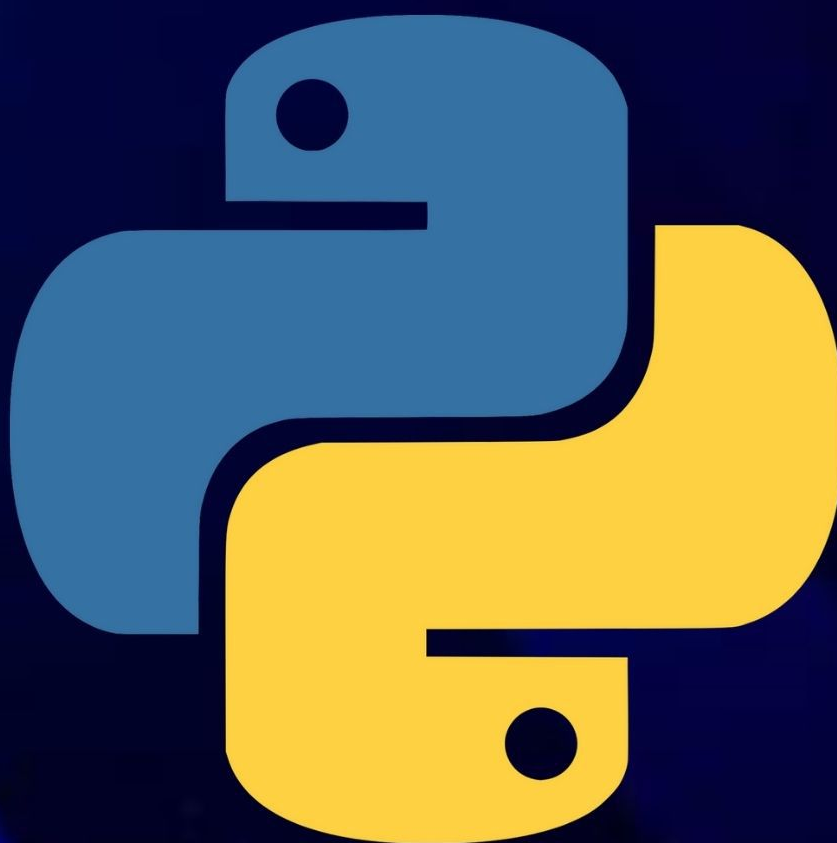


50 DAYS OF PYTHON

A Challenge a Day.



Benjamin Bennett Alexander

50 DAYS OF PYTHON

A challenge a Day

Benjamin Bennett Alexander

"Python" and the Python Logo are trademarks of the Python Software Foundation.

Copyright © 2022 by Benjamin Bennett Alexander

All rights are reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior permission of the publisher.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, we do not warrant or represent its completeness or accuracy.

Feedback and Reviews

I welcome and appreciate your feedback and reviews. Please consider writing a review on the platform you purchased this book from. Please send your queries to:

benjaminbennettalexander@gmail.com.

Table of Contents

Feedback and Reviews	3
Table of Contents.....	4
Introduction	8
Day 1: Division and Square-root	10
Extra Challenge: Longest Value	10
Day 2: Strings to Integers.....	11
Extra Challenge: Duplicate Names.....	11
Day 3: Register Check.....	12
Extra Challenge: Lowercase Names	12
Day 4: Only Floats	13
Extra Challenge: Index of the Longest Word.....	13
Day 5: My Discount.....	14
Extra Challenge: Tuple of Student Sex	14
Day 6: User Name Generator	15
Extra Challenge: Zero Both Ends	15
Day 7: A String Range	16
Extra Challenge: Dictionary of Names.....	16
Day 8: Odd and Even.....	17
Extra Challenge: List of Prime Numbers.....	17
Day 9: Biggest Odd Number.....	18
Extra Challenge: Zeros to the End.....	18
Day 10: Hide my Password	19

Extra Challenge: A Thousand Separator	19
Day 11: Are They Equal?	20
Extra Challenge: Swap Values	20
Day 12: Count the Dots	21
Extra Challenge: Your Age in Minutes	21
Day 13: Pay Your Tax	22
Extra Challenge: Pyramid of Snakes	22
Day 14: Flatten the List	23
Extra Challenge: Teacher's Salary	23
Day 15: Same in Reverse	24
Extra Challenge: What's My Age?	24
Day 16: Sum the List	25
Extra Challenge: Unpack A Nest	25
Day 17: User Name Generator	26
Extra Challenge: Sort by Length	26
Day 18: Any Number of Arguments	27
Extra Challenge: Add and Reverse	27
Day 19: Words and Elements	28
Day 20: Capitalize First Letter	29
Extra Challenge: Reversed List	29
Day 21: List of Tuples	30
Extra Challenge: Even Number or Average	30
Day 22: Add Under_Score	31
Day 23: Simple Calculator	32
Extra Challenge: Multiply Words	32

Day 24: Average Calories	33
Extra Challenge: Create a Nested List	33
Day 25: All the Same.....	34
Extra Challenge: Reverse a String	34
Day 26: Sort Words	35
Extra Challenge: Length of Words	35
Day 27: Unique Numbers	36
Extra Challenge: Difference of Two Lists	36
Day 28: Return Indexes	37
Extra Challenge: Largest Number	37
Day 29: Middle Figure	38
Day 30: Most Repeated Name	39
Extra Challenge: Sort by Last Name	39
Day 31: Longest Word.....	40
Extra Challenge: Create User	40
Day 32: Password Validator.....	41
Extra Challenge: Valid Email	41
Day 33: List Intersection	42
Extra Challenge: Set or List.....	42
Day 34: Just Digits	43
Day 35: Pangram.....	44
Day 36: Count String.....	45
Extra Challenge: Sum a Nested Dictionary	45
Day 37: Count the Vowels.....	46
Day 38: Guess a Number	47

Extra Challenge: Find Missing Numbers	47
Day 39: Password Generator	48
Day 40: Pig Latin	49
Day 41: Only Words with Vowels	50
Extra Challenge: Class of Cars	50
Day 42: Spelling Checker	51
Extra Challenge: Set Alarm	51
Day 43: Student Marks.....	52
Day 44: Save Emails	53
Day 45: Words & Special Characters.....	54
Day 46: Create a DataFrame	55
Extra Challenge: Website Data with Pandas.....	55
Day 47: Save a JSON.....	57
Day 48: Binary Search.....	58
Day 49: Create a Database	59
Day 50: Create a Flask App.....	60
Answers to the Challenges.....	62
Other Books by Author	145

Introduction

Welcome to the world of Python challenges! Just like in life, challenges are the building blocks of a strong character. They push us to grow, learn, and become better. This book is designed to provide you with an exciting opportunity to master the key concepts and fundamentals of Python through solving a series of captivating challenges. With over 80 challenges waiting for you, embark on this 50-day journey to become a Python expert.

Why Solve challenges?

As a programmer, your ultimate goal is to solve problems efficiently. By tackling challenges, you not only solidify your knowledge but also develop the invaluable skill of problem-solving. Each challenge will train your mind to think critically and find innovative solutions to real-life programming scenarios.

How to Use this Book?

To make the most of this book, embrace the challenge and resist the temptation to immediately seek the answers. Take your time and dedicate a day to solve each problem. Some challenges may seem easy, while others will push your limits. Remember, there can be multiple solutions to a problem, and your unique approach is just as valid as the one presented in the book.

Throughout this book, you will encounter a wide range of Python fundamentals. From creating password generators to working with modules, translating strings, manipulating dictionaries, handling CSV and JSON files, and even building a website - you'll be immersed in a diverse set of tasks. Get ready to dive in and embark on an exhilarating learning journey.

Conclusion

With the promise of over 80 engaging challenges covering various Python concepts, this book is your gateway to mastering the fundamentals. Take one step at a time, challenge by challenge, and watch your Python skills flourish. Remember, it's not about how quickly you finish, but the knowledge and experience you gain along the way. So, without further ado, let the adventure begin!

Day 1: Division and Square-root

Write a function called **`divide_or_square`** that takes one argument (a number) and returns the square root of the number if it is divisible by 5, or its remainder if it is not divisible by 5. For example, if you pass 10 as an argument, then your function should return 3.16 as the square root.

Extra Challenge: Longest Value

Write a function called **`longest_value`** that takes a dictionary as an argument and returns the first longest value of the dictionary. For example, the following dictionary should return "apple" as the longest value.

```
fruits = {'fruit': 'apple', 'color': 'green'}
```

Day 2: Strings to Integers

Write a function called **`convert_add`** that takes a list of strings as an argument, converts them to integers, and sums the list. For example, `["1", "3", "5"]` should be converted to `[1, 3, 5]` and **summed** to 9.

Extra Challenge: Duplicate Names

Write a function called **`check_duplicates`** that takes a list of strings as an argument. The function should check if the list has any duplicates. If there are duplicates, the function should return a list of duplicates. If there are no duplicates, the function should return "no duplicates." For example, the list of **fruits** below should return `["apple", "banana"]`, and the list of **names** should return **"no duplicates."**

```
fruits = ['apple', 'orange', 'banana', 'apple', 'banana']  
names = ['Yoda', 'Moses', 'Joshua', 'Mark']
```

Day 3: Register Check

Write a function called **register_check** that checks how many students are in school. The function takes a dictionary as an argument. If the student is in school, the dictionary says "yes." If the student is not in school, the dictionary says "no." Your function should return the number of students in school. Using the dictionary below, your function should return 3.

```
In [*]: register = {'Michael': 'yes', 'John': 'no',  
                   'Peter': 'yes', 'Mary': 'yes'}
```

Extra Challenge: Lowercase Names

```
names = ["kerry", "dickson", "John", "Mary",  
        "carol", "Rose", "adam"]
```

You are given a list of names above. This list is made up of names with lowercase and uppercase letters. Your task is to write a code that will return a **tuple** of all the names in the list that **only** have **lowercase** letters. Your tuple should have names sorted alphabetically in descending order. Using the list above, your code should return:

('kerry', 'dickson', 'carol', 'adam')

Day 4: Only Floats

Write a function called `only_floats`, which has two parameters, **a** and **b**, and returns **2** if both arguments are floats, **1** if only one argument is a **float**, and **0** if neither argument is a float. If you pass (12.1, 23) as an argument, your function should return 1.

Extra Challenge: Index of the Longest Word

Write a function called `word_index` that takes one argument, a **list** of strings, and returns the **index** of the longest word in the list. If there is no longest word (if all the strings are of the same length), the function should return zero (0). For example, the first list below should return 2.

```
words1 = ["Hate", "remorse", "vengeance"]
```

And this list below, should return zero (0)

```
words2 = ["Love", "Hate"]
```

Day 5: My Discount

Create a function called `my_discount`. The function takes no arguments but asks the user to input the **price** and the **discount** (percentage) of the product. Once the user inputs the price and discount, it calculates the **price after the discount**. The function should return the price after the discount. For example, if the user enters **150** as the price and **15%** as the discount, your function should return **127.5**.

Extra Challenge: Tuple of Student Sex

You work for a school, and your boss wants to know how many female and male students are enrolled in the school. The school has saved the students on a list. Your task is to write a code that will count how many males and females are in the list. Here is a list below:

```
students = ['Male', 'Female', 'female', 'male', 'male', 'male',  
            'female', 'male', 'Female', 'Male', 'Female', 'Male',  
            'female']
```

Your code should return a list of tuples. The list above should return: `[("Male", 7), ("female", 6)]`

Day 6: User Name Generator

Write a function called `user_name` that generates a username from the user's email. The code should ask the user to input an email, and the code should return everything before the @ sign as their user name. For example, if someone enters **ben@gmail.com**, the code should return **ben** as their user name.

Extra Challenge: Zero Both Ends

Write a function called `zeroed` code that takes a list of numbers as an argument. The code should zero (0) the first and last number in the list. For example, if the input is **[2, 5, 7, 8, 9]**, your code should return **[0, 5, 7, 8, 0]**.

Day 7: A String Range

Write a function called `string_range` that takes a single number and returns a string of its range. The string characters should be **separated by dots(.)**. For example, if you pass **6** as an argument, your function should return **"0.1.2.3.4.5."**

Extra Challenge: Dictionary of Names

You are given a list of names, and you are asked to write a code that returns all the names that start with "S." Your code should return a dictionary of all the names that start with **S** and how many times they appear in the dictionary. Here is a list below:

```
names = ["Joseph", "Nathan", "Sasha", "Kelly",  
         "Muhammad", "Jabulani", "Sera", "Patel", "Sera"]
```

Your code should return: **{"Sasha": 1, "Sera": 2}**

Day 8: Odd and Even

Write a function called **odd_even** that has one parameter and takes a list of numbers as an argument. The function returns the difference between the largest **even** number in the list and the smallest **odd** number in the list. For example, if you pass **[1, 2, 4, 6]** as an argument, the function should return **6 - 1 = 5**.

Extra Challenge: List of Prime Numbers

Write a function called **prime_numbers**. This function asks the user to enter a number (an integer) as an argument and returns a list of all the prime numbers within its range. For example, if the user enters 10, your code should return **[2, 3, 5, 7]** as prime numbers.

Day 9: Biggest Odd Number

Create a function called **`biggest_odd`** that takes a string of numbers as an argument and returns the biggest odd number in the string. For example, if you pass **`'23569'`** as an argument, your function should return **`9`**. Use **list comprehension**.

Extra Challenge: Zeros to the End

Write a function called **`zeros_last`**. This function takes a list as an argument. If a list has zeros (0), it will move them to the end of the list and maintain the order of the other numbers in the list. If there are no zeros in the list, the function should return the original list, sorted in ascending order. For example, if you pass **`[1, 4, 6, 0, 7, 0, 9]`** as an argument, your code should return **`[1, 4, 6, 9, 0, 0]`**. If you pass **`[2, 1, 4, 7, 6]`** as your argument, your code should return **`[1, 2, 4, 6, 7]`**.

Day 10: Hide my Password

Write a function called **hide_password** that takes no parameters. The function takes an input (a password) from a user and returns a hidden password. For example, if the user enters "hello" as a password, the function should return "*****" as a password and tell the user that the password is **4 characters** long.

Extra Challenge: A Thousand Separator

Your new company has a list of figures saved in a database. The issue is that these numbers have no separator. The numbers are saved in a list in the following format:

[1000000, 2356989, 2354672, 9878098].

You have been asked to write a code that will convert each of the numbers in the list into a **string**. Your code should then add a comma to each number as a **thousand separator** for readability. When you run your code on the above list, your output should be:

['1,000,000', '2,356,989', '2,354,672', '9,878,098']

Write a function called **convert_numbers** that will take one argument, the list of numbers above.

Day 11: Are They Equal?

Write a function called **`equal_strings`**. The function takes two strings as arguments and compares them. If the strings are equal (if they have the same characters and have equal length), it should return **True**; if they are not, it should return **False**. For example, "love" and "evol" should return **True**.

Extra Challenge: Swap Values

Write a function called **`swap_values`**. This function takes a list of numbers and swaps the first element with the last element. For example, if you pass **[2, 4, 67, 7]** as a parameter, your function should return **[7, 4, 67, 2]**.

Day 12: Count the Dots

Write a function called `count_dots`. This function takes a string separated by dots as a parameter and counts how many dots are in the string. For example, "h.e.l.p." should return 4 dots, and "he.lp." should return 2 dots.

Extra Challenge: Your Age in Minutes

Write a function called `age_in_minutes` that tells a user **how old they are in minutes**. Your code should ask the user to enter their **year of birth**, and it should return their age in minutes (by subtracting their year of birth from the current year). Here are things to look out for:

- a. The user can only input a **4-digit** year of birth. For example, 1930 is a valid year. However, entering any number longer or shorter than 4 digits, should render the input invalid. Notify the user that they must enter a four-digit number.
- b. If a user enters a year **before** 1900, your code should tell the user that the input is invalid. If the user enters the year **after** the current year, the code should tell the user to input a valid year.

The code should **run until the user inputs a valid year**. Your function should return the user's age in minutes. For example, if someone enters 1930 as their year of birth, your function should return:

You are 48, 355, 200 minutes old.

Day 13: Pay Your Tax

Write a function called `your_vat`. The function has no parameters. The function asks the user to input the price of an item and VAT (VAT should be a percentage). The function should return the price of the item plus VAT. If the price is 220 and the VAT is 15%, your code should return a VAT-inclusive price of 253. Check to see if your code can handle **ValueError** and negative inputs from the user. Ensure the code runs until valid numbers are entered. (Hint: Your code should include a *while* loop.)

Extra Challenge: Pyramid of Snakes

Write a function called `python_snakes` that takes a number as an argument and creates the pyramid shape using the number's range. (Hint: Use the loops and emoji libraries for snake emojis.) If you pass 7 as an argument, your code should print the following:



Day 14: Flatten the List

Write a function called `flat_list` that takes one argument, a nested list. The function converts the nested list into a one-dimensional list. For example, `[[2, 4, 5, 6]]` should return `[2, 4, 5, 6]`.

Extra Challenge: Teacher's Salary

A school has asked you to write a program that will calculate teachers' salaries. The program should ask the user to enter the teacher's **name**, the **number of periods** taught in a month, and the **rate** per period. The monthly salary is calculated by multiplying the **number of periods** by the **monthly rate**. The current monthly rate per period is \$20. If a teacher has more than 100 periods in a month, everything above 100 is overtime. Overtime is \$25 per period. For example, if a teacher has taught **105** periods, their gross monthly salary should be **\$2,125**. Write a function called `"your_salary"` that calculates a teacher's gross salary. The function should return the **teacher's name**, **periods taught**, and **gross salary**. Here is how you should format your output:

Teacher: John Kelly,
Periods: 105
Gross salary:2,125

Day 15: Same in Reverse

Write a function called `same_in_reverse` that takes a string and checks if the string reads the same in reverse. If it is the same, the code should return **True** if not, it should return **False**. For example, **"dad"** should return **True**, because it reads the same in reverse.

Extra Challenge: What's My Age?

Write a function called `your_age`. This function asks a student to enter their name, and then it returns their age. For example, if a user enters **Peter** as their name, your function should return, **"Hi, Peter, you are 27 years old."** This function reads data from the database (dictionary below). If the name is not in the dictionary, your code should tell the user that their name is **not** in the dictionary and ask them for their age. Your code should then add the **name** and **age** to the dictionary named **"names_age"** below. Once added, your code should return to the user, **"Hi, (name), you are (age) years old."** Remember to convert the input from the user to lowercase letters.

```
names_age = {"jane": 23, "kerry": 45, "tim": 34, "peter": 27}
```

Day 16: Sum the List

Write a function called `sum_list` with one parameter that takes a **nested list** of integers as an argument and returns the sum of the integers. For example, if you pass `[[2, 4, 5, 6], [2, 3, 5, 6]]` as an argument, your function should return a sum of **33**.

Extra Challenge: Unpack A Nest

```
nested_list = [[12, 34, 56, 67], [34, 68, 78]]
```

Write a **code** that generates a list of the following numbers from the nested list above: 34, 67, 78. Your **output** should be another list: `[34, 67, 78]`. The list output should **not** have **duplicates**.

Day 17: User Name Generator

Write a function called `user_name`, that creates a username for the user. The function should ask a user to **input** their name. The function should then reverse the name and attach a randomly issued number between 0 and 9 at the end of the name. The function should return the **username**.

Extra Challenge: Sort by Length

```
names = [ "Peter", "Jon", "Andrew"]
```

Write a function called `sort_length` that takes a list of strings as an argument and sorts the strings in ascending order according to their length. For example, the list above should return:

```
['Jon', 'Peter', 'Andrew']
```

Do **not** use the built-in sort functions.

Day 18: Any Number of Arguments

Write a function called `any_number` that can receive any number of positional arguments (integers and floats) and return the average of those integers. If you pass **12, 90, 12** and **34** as arguments, your function should return **37.0** as the average. If you pass **12** and **90**, your function should return **51.0** as the average.

Extra Challenge: Add and Reverse

Write a function called `add_reverse`. This function takes **two** lists as arguments, adds each corresponding number, and **reverses** the outcome. For example, if you pass **[10, 12, 34]** and **[12, 56, 78]**, your code should return **[112, 22, 68]**. If the two lists are not of equal length, the code should return a message that "the lists are not of equal length."

Day 19: Words and Elements

Write two functions. The first function is called **`count_words`** which takes a string of words as argument and counts how many words are in the string.

The second function, called **`count_elements`** takes a string of words and counts how many elements are in the string. Do not count the whitespaces. The first function will return the number of **words** in a string, and the second one will return the number of **elements** (less whitespace). If you pass **"I love learning,"** the **`count_words`** function should return **3 words** and **`count_elements`** should return **13 elements**.

Day 20: Capitalize First Letter

Write a function called **capitalize**. This function takes a string as an argument and **capitalizes** the first letter of each word. For example, "i like learning" becomes "I Like Learning."

Extra Challenge: Reversed List

```
str1 = 'leArning is hard, bUt if You appLy youRself ' \
       'you can achieVe success'
```

You are given a string of words. Some of the words in the string contain uppercase letters. Write a code that will return all the words that have an uppercase letter. Your code should return a list of the words. Each word in the list should be reversed. Here is how your output should look:

['gninrAel', 'tUb', 'uoY', 'yLppa', 'flesRuoy', 'eVeihca']

Day 21: List of Tuples

Write a function called **`make_tuples`** that takes **two equal** lists and combines them into a list of tuples. Your code should check that the input lists are of the same length. For example, if list **a** is **[1, 2, 3, 4]** and list **b** is **[5, 6, 7, 8]**, your function should return **[(1, 5), (2, 6), (3, 7), (4, 8)]**. If the lists are not of equal length, your function should **raise** a `ValueError`.

Extra Challenge: Even Number or Average

Write a function called **`even_or_average`** that asks a user to input five numbers. Once the user is done, the code should return the largest even number in the inputted numbers. If there is no even number in the list, the code should return the average of all the five numbers.

Day 22: Add Under_Score

Create three functions. The first, called **`add_hash`** takes a string and adds a hash (#) between the words. The second function, called **`add_underscore`** removes the hash (#) and replaces it with an underscore ("_"). The third function, called **`remove_underscore`**, removes the underscore and replaces it with nothing. If you pass "Python" as an argument for the three functions, and you call them at the same time like:

```
print(remove_underscore(add_underscore(add_hash('Python'))))
```

it should return **"Python"**.

Day 23: Simple Calculator

Create a simple calculator. The calculator should be able to perform the following basic math operations: **add**, **subtract**, **divide**, and **multiply**. The calculator should take input from users. The calculator should be able to handle **ZeroDivisionError**, **NameError**, and **ValueError**.

Extra Challenge: Multiply Words

s = "love live and laugh"

Write a function called **multiply_words** that takes a string as an argument and multiplies the length of each word in the string by the length of other words in the string. For example, the string above should return **240: love (4), live (4), and (3), laugh (5)**. However, your function should only multiply words with all lowercase letters. If a word has one uppercase letter, it should be ignored. For example, the following string:

s = "Hate war love Peace" should return **12 – war (3) love (4)**. **Hate** and **Peace** will be ignored because they have at least one uppercase letter.

Day 24: Average Calories

Write a function called **average_calories** that calculates the average calorie intake of a user. The function should ask the user to input their calorie intake for **any** number of days, and once they hit "done," it should calculate and **return** the average intake.

Extra Challenge: Create a Nested List

Write a function called **nested_lists** that takes any number of lists and creates a 2-dimensional nested list of lists. Your code must check that the inputs are of the list data type. For example, if you pass the following lists as arguments: **[1, 2, 3, 5], [1, 2, 3, 4], [1, 3, 4, 5]**.

Your code should return: **[[1, 2, 3, 5], [1, 2, 3, 4], [1, 3, 4, 5]]**

If you pass: **(1, 2, 3, 5), [1, 2, 3, 4], [1, 3, 4, 5]** as arguments, your function should return **"Invalid arguments. Please check your arguments."**

Day 25: All the Same

Write a function called **`all_the_same`** that takes one argument, a **string**, a **list**, or a **tuple**, and checks if all the elements are the same. Use the **`instance`** function to check that the input data is either a **string**, a **list**, or a **tuple**, or else raise a **`TypeError`** that says that **"Input must be a string, a list, or a tuple."** If the elements are the same, the function should return **`True`**. If not, it should return **`False`**. For example, **`["Mary", "Mary", "Mary"]`** should return **`True`**.

Extra Challenge: Reverse a String

`str1 = "the love is real"`

Write a function called **`read_backwards`** that takes a string as an argument and reverses it. The string above, for example, should return: **`"real is love the."`**

Day 26: Sort Words

Write a function called **`sort_words`** that takes a string of words as an argument, removes the whitespaces, and returns a list of letters sorted in alphabetical order. Letters will be separated by commas. All letters should appear once in the list. This means that you sort and remove duplicates. For example, "**love life**" should return as **`['e', 'f', 'i', 'l', 'o', 'v']`**.

Extra Challenge: Length of Words

`s = 'Hi my name is Richard'`

Write a function called **`string_length`** that takes a string of words (words and spaces) as an argument. This function should return the length of all the words in the string. Return the results in the form of a dictionary. The string above should return:

`{'Hi': 2, 'my': 2, 'name': 4, 'is': 2, 'Richard': 7}`

Day 27: Unique Numbers

Write a function called **unique_numbers** that takes a list of numbers as an argument. Your function is going to find all the unique numbers in the list. It will then sum up the unique numbers. You will calculate the difference between the **sum of all the numbers in the original list** and the **sum of the unique numbers in the list**. If the difference is an **even** number, your function should return the **original list**. If the difference is an **odd** number, your function should return a **list with only unique numbers**. For example, `[1, 2, 4, 5, 6, 7, 8, 8]` should return `[1, 2, 4, 5, 6, 7, 8, 8]`.

Extra Challenge: Difference of Two Lists

Write a function called **difference** that takes **two** lists as arguments. This function should return all the elements that are in list **a** but not in list **b** and all the elements in list **b** but not in list **a**. **For example:**

list1 = `[1, 2, 4, 5, 6]`

list2 = `[1, 2, 5, 7, 9]`

should return:

`[4, 6, 7, 9]`

Use **list comprehension** in your function.

Day 28: Return Indexes

Write a function called **`index_position`**. This function takes a string as a parameter and returns the positions, or indexes, of all lowercase letters in the string. For example, "LovE" should return **[1, 2]**.

Extra Challenge: Largest Number

Write a function called **`largest_number`** that takes a list of integers as an argument and re-arranges the individual digits to create the largest number possible. For example, if you pass the following as an argument, **list1 = [3, 67, 87, 9, 2]**. Your code should return the number in this exact format:
9, 877, 632 as the largest number.

Day 29: Middle Figure

Write a function called ***middle_figure*** that has **two** parameters, **a** and **b**. The **two** parameters are strings. The function joins the **two** strings and finds the middle element. If the combined string has a middle element, the function should return the element; otherwise, it should return "no middle figure." Use **"make love"** as an argument for **a**, and **"not wars"** as an argument for **b**. Your function should return **"e"** as the middle element. Whitespaces should be removed.

Day 30: Most Repeated Name

Write a function called **repeated_name** that finds the most repeated name in the following list. Your function should return a tuple of the name and how many times it appears. The list below should return: ("Peter", 3).

```
name = ["John", "Peter", "John", "Peter", "Jones", "Peter"]
```

Extra Challenge: Sort by Last Name

You work for a local school in your area. The school has a list of names of students saved in a list format. The school has asked you to write a program that takes a list of names and sorts them alphabetically. The names should be sorted by last name. Here is a list of names:

["Beyoncé Knowles", "Alicia Keys", "Katie Perry", "Chris Brown", "Tom Cruise"]

Your code should not just sort them alphabetically; it should also switch the names (the last name must be the first). Here is how your code output should look:

['Brown Chris', 'Cruise Tom', 'Keys Alicia', 'Perry Katie', 'Knowles Beyoncé']

Write a function called **sorted_names**.

Day 31: Longest Word

Write a function that has one parameter and takes a list of words as an argument. The function returns the longest word from the list. Name the function **`longest_word`**. The function should return the longest word and the number of letters in that word. For example, if you pass `['Java', 'JavaScript', 'Python']`, your function should return

`[10, JavaScript]` as the longest word.

Extra Challenge: Create User

Write a function called **`create_user`**. This function asks the user to enter their name, age, and password. The function saves this information in a dictionary. For example, if the user enters Peter as his name, 18 as his age, and "love" as his password, the function should save the information as: **`{"name": "Peter", "age": "18", "password": "love"}`**.

Once the information is saved, the function should print to the user, **"User saved. Please login"**

The function should then ask the user to re-enter their name and password. If the name and password match what is saved in the dictionary, the function should return **"Logged in successfully."** If the name and password do not match what is saved in the dictionary, the function should print **"Wrong password or user name, please try again"**. The function should keep running until the user enters the correct logging details.

Day 32: Password Validator

Write a function called **`password_validator`**. The function asks the user to enter a password. A valid password should have at least **one uppercase letter, one lowercase letter, and one number**. It should not be less than **8 characters long**. When the user enters a password, the function should check if the password is valid. If the password is valid, the function should return the valid password. If the password is not valid, the function should inform the user of the errors in the password and prompt the user to enter another password. The code should only stop once the user enters a valid password.

Extra Challenge: Valid Email

```
emails = [ "ben@mail.com", "john@mail.cm", "kenny@gmail.com", "@list.com" ]
```

Write a function called **`email_validator`** that takes a list of emails and checks if the emails are valid. Only valid emails are returned by the function. A valid email address will have the following: the @ symbol (if the @ sign is at the beginning of the name, the email is invalid). If there is more than one @ sign, the email is invalid. All valid emails must end with a **dot com (.com)**; otherwise, the email is invalid. For example, the list of emails above should output the following as valid emails:

```
['ben@mail.com', 'kenny@gmail.com']
```

If no emails in the list are valid, the function should return "All emails are invalid."

Day 33: List Intersection

Write a function called `inter_section` that takes **two** lists and finds the intersection (the elements that are present in both lists). The function should return a tuple of intersections with the number at **index 0** switched with the number at **index -1**. Use list comprehension in your solution. Use the lists below. Your function should return **(80, 65, 30)**.

```
list1 = [20, 30, 60, 65, 75, 80, 85]
list2 = [42, 30, 80, 65, 68, 88, 95]
```

Extra Challenge: Set or List

You want to implement code that will search for a number in a range. You have a decision to make as to whether to store the number in a **set** or a **list**. Your decision will be based on time. You have to pick a data type that **executes faster**.

You have a range, and you can either store it in a **set** or a **list**, depending on which one executes faster when you are searching for a number in the range. See below:

```
a = range(100000000)
x = set(a)
y = list(a)
```

Let's say you are looking for the number 9999999 in the range above. Search for this number in the **list "x"** and the **set "y."** Your challenge is to find which code executes faster. You will select the one that executes the fastest between **lists** and **sets**. Run the two searches and time them.

Day 34: Just Digits

In this challenge, copy the **text below** and save it as a CSV file. Save it in the same folder as your Python file. Save it as **python.csv**.

Write a function called **`just_digits`** that reads the text from the CSV file and returns only digit elements from the file. Your function should return **1991, 2, 200, 3, 2008** as a list of strings.

“Python was released in 1991 for the first time. Python 2 was released in 2000 and introduced new features, such as list comprehensions and a cycle-detecting garbage collection system (in addition to reference counting). Python 3 was released in 2008 and was a major revision of the language that is not completely backward-compatible.”

Source: Wikipedia

Day 35: Pangram

Write a function called `check_pangram` that takes a string and checks if it is a pangram. A **pangram is a sentence that contains all the letters of the alphabet**. If it is a pangram, the function should return **True**, otherwise, it should return **False**. The following sentence is a pangram, so it should return **True**:

'the quick brown fox jumps over a lazy dog'

Extra Challenge: Find my Position

Write a function called `find_index` that takes two arguments; a list of integers, and an integer. The function should return the indexes of the integers in the list. If the integer is **not** in the list, the function should convert all the numbers in the list into the given integer.

For example, if the list is **[1, 2, 4, 6, 7, 7]** and the integer is **7**, your code should return **[4, 5]** as the indexes of **7** in the list. If the list is **[1, 2, 4, 6, 7, 7]** and the integer is **8**, your code should return **[8, 8, 8, 8, 8, 8]** because **8** is not in the list.

Day 36: Count String

Write a function called **count** that takes one argument, a **string**, and returns a **dictionary** of how many times each element appears in the string. For example, **'hello'** should return: **{'h':1,'e': 1,'l':2, 'o':1}**.

Extra Challenge: Sum a Nested Dictionary

You have a nested dictionary below. Write a function called **sum_nested_dict**, that takes one argument, a dictionary, and returns the sum of all the values in the dictionary. The nested dictionary below, should return 15.

```
nested_dict = {'a': 1, 'b': {'c': 2, 'd': {'e': 3, 'f': 4}}, 'g': 5}
```

Day 37: Count the Vowels

Create a function called `count_the_vowels`. The function takes one argument, a string, and returns the number of vowels in the string. If the data type of the input is not of the string type, the function should return "invalid input." If the argument has no vowels, your function should return **"The string has no vowels."** If a vowel appears in a string more than once, it should be counted as **one**. For example, **"hello"** should return **two (2)** vowels; **"saas"** should return **one (1)** vowel; and **"sly"** should return: **"The string has no vowels."** Your code should count lowercase and uppercase vowels.

Day 38: Guess a Number

Write a function called **`guess_a_number`**. The function should ask a user to guess a randomly generated number. If the user guesses a higher number, the code should tell them that the guess is too high; if the user guesses a low number, the code should tell them that their guess is too low. The user should get a maximum of **three (3)** guesses. When the user guesses right, the code should declare them a winner. After three wrong guesses, the code should declare them losers.

Extra Challenge: Find Missing Numbers

```
list1 = [1, 2, 3, 5, 6, 7, 9, 11, 12, 23, 14, 15, 17,  
         18, 19, 20, 21, 22, 24, 25, 26, 27, 28, 31]
```

Write a function called **`missing_numbers`** that takes a list of sequence of numbers as argument, and finds the missing numbers in the sequence. The list above should return:

```
[4, 8, 10, 13, 16, 29, 30]
```


Day 39: Password Generator

Create a function called **`generate_password`** that generates any length of password for the user. The password should have a random mix of uppercase letters, lowercase letters, numbers, and punctuation symbols. The function should ask the user how strong they want the password to be. The user should pick from **weak**, **strong**, or **very strong**. If the user picks "weak," the function should generate a **5-character** long password. If the user picks "strong," generate an **8-character** password, and if they pick "very strong," generate a **12-character** password.

Day 40: Pig Latin

Write a function called ***translate*** that takes the following words and translates them into Pig Latin.

```
my_string = 'i love python'
```

Here are the rules:

1. If a word starts with a vowel (a, e, i, o, u) add "yay" at the end. For example, "eat" will become "eatyay"
2. If a word starts with anything other than a vowel, move the first letter to the end and add 'ay' to the end. For example, "day" will become "ayday."

Your code should return:

iyay ovelay ythonpay

Day 41: Only Words with Vowels

Create a function called **`words_with_vowels`**. This function takes a string of words and returns a list of only the words that have vowels in them. "You have no rhythm," for example, should return ["you," "have," "no"].

Extra Challenge: Class of Cars

Create three classes for three car brands: **Ford**, **BMW**, and **Tesla**. The instance attributes of the car's objects will be **model**, **color**, **year**, **transmission**, and whether the car is electric or not (a Boolean value). Consider using inheritance in your answer.

You will create one object for each car brand:

bmw1 : **model**: x6, **Color**: silver, **Year**: 2018, **Transmission**: Auto, **Electric**: False
tesla1: **model**: S, **Color**: beige, **Year**: 2017, **Transmission**: Auto, **Electric**: True
ford1 : **model**: focus, **Color**: white, **Year**: 2020, **Transmission**: Auto, **Electric**: False

You will create an **instance method** called **`print_cars`** that will be able to print out objects of the class. For example, if you call the method on the **ford1** object of the Ford class, your function should be able to print out **car info** in this exact format:

car_model = focus
Color = White
Year = 2020
Transmission = Auto
Electric = False

Day 42: Spelling Checker

Write a function called **`spelling_checker`**. This code asks the user to input a word, and if the user inputs the wrong spelling, it should suggest the **correct spelling** by asking the user if they meant to type that word. If the user says no, it should ask the user to enter the word again. If the user says yes, it should return the correct word. If the word entered by the user is correctly spelled, the function should return the correct word. Use the **module `textblob`**.

Extra Challenge: Set Alarm

Write a function called **`set_alarm`** that sets an alarm clock for the user. The function should ask the user to enter the time they want the alarm to go off. The user should enter the time in hours and minutes. The function should ensure that the user enters valid inputs for hour and minutes. The function should run until the user enters valid input. Once the user enters valid inputs, the function should then print out the time when the alarm will go off. When it's alarm time, the code should let off a sound. Use the **`winsound`** module for sound.

Day 43: Student Marks

Write a function called **`student_marks`** that records the marks achieved by students in a test. The function should ask the user to input the **name** of the student and then ask the user to input the **marks** achieved by the student. The information should be stored in a dictionary. The **name** is the **key**, and the **mark** is a **value**. When the user enters "done," the function should return a dictionary of the names and values entered or an empty dictionary if no values are entered. Your code should ensure that:

- ◁ If a **name** contains punctuation characters (!"#\$%&'()*+,-./:;<=>?@[\\]^_`{|}~) or **numbers(digits)** your code should raise a **NameError** and ask the user to enter a valid name.
- ◁ If the a user enters invalid values for **value**, your code should handle the **ValueError**.
- ◁ Your code should run until a valid values are inputted.

Day 44: Save Emails

Create a function called ***save_emails***. This function takes no arguments but asks the user to input an email, and it saves the emails in a CSV file. The user can input as many emails as they want. Once they hit "done," the function saves the emails and closes the file. Create another function called ***open_emails***. This function opens and reads the content of the CSV file. Each email requires its own line. Here is an example of how the emails must be saved:

ji@gmail.com

kate@yahoo.com

and **not** like this:

ji@gmail.comkate@yahoo.com

Day 45: Words & Special Characters

Write a function called **`analyse_string`** that takes a string as an argument and returns the number of **special characters** (`#$%&'()*+,-./:;<=>?@[\]^_`{|}~`), **numeric characters**, and **total characters** (the length of the string minus the whitespaces) in the string. Return everything in a dictionary format:

```
{"special_char": "number," "numeric_char": "number,"  
"total characters": "number"}
```

Use the string below as an argument:

"Python has a string format operator %. This functions analogously to printf format strings in C, e.g. "spam=%s eggs=%d" % ("blah", 2) evaluates to "spam=blah eggs=2".

Source Wikipedia.

Day 46: Create a DataFrame

Create a **DataFrame** using pandas. You are going to create code to put the following into a **DataFrame**. You will use the information in the table below. Basically, you are going to recreate this table using pandas. Use the information in the table to recreate the table.

year	Title	genre
2009	Brothers	Drama
2002	Spider-Man	Sci-fi
2009	WatchMen	Drama
2010	Inception	Sci-fi
2009	Avatar	Fantasy

Extra Challenge: Website Data with Pandas

Create a code that extracts data from a website. You will extract a table from the website. And from that table, you will extract columns about the data types in Python and their mutability. You will extract information from the following website:

[https://en.wikipedia.org/wiki/Python_\(programming_language\)](https://en.wikipedia.org/wiki/Python_(programming_language))

The following table (next page) is what you will extract from the website.

Summary of Python 3's built-in types			
Type	Mutability	Description	Syntax examples
bool	immutable	Boolean value	True False
bytearray	mutable	Sequence of bytes	bytearray(b'Some ASCII') bytearray(b'Some ASCII') bytearray([119, 105, 107, 105])
bytes	immutable	Sequence of bytes	b'Some ASCII' b'Some ASCII' bytes([119, 105, 107, 105])
complex	immutable	Complex number with real and imaginary parts	3+2.7j 3 + 2.7j
dict	mutable	Associative array (or dictionary) of key and value pairs; can contain mixed types (keys and values), keys must be a hashable type	{'key1': 1.0, 3: False} {}
types.EllipsisType	immutable	An ellipsis placeholder to be used as an index in NumPy arrays	... Ellipsis
float	immutable	Double-precision floating-point number. The precision is machine-dependent but in practice is generally implemented as a 64-bit IEEE 754 number with 53 bits of precision. ^[105]	1.33333
frozenset	immutable	Unordered set, contains no duplicates; can contain mixed types, if hashable	frozenset([4.0, 'string', True])
int	immutable	Integer of unlimited magnitude ^[106]	42
list	mutable	List, can contain mixed types	[4.0, 'string', True] []
types.NoneType	immutable	An object representing the absence of a value, often called null in other languages	None
types.NotImplementedType	immutable	A placeholder that can be returned from overloaded operators to indicate unsupported operand types.	NotImplemented
range	immutable	A Sequence of numbers commonly used for looping specific number of times in for loops ^[107]	range(-1, 10) range(10, -5, -2)
set	mutable	Unordered set, contains no duplicates; can contain mixed types, if hashable	{4.0, 'string', True} set()
			'Wikipedia'

Once you extract this table, you will write a code that will extract the data types and their mutability (Two columns). Here is how your output should look:

```

|:

```

	Type	Mutability
0	bool	immutable
1	bytearray	mutable
2	bytes	immutable
3	complex	immutable
4	dict	mutable
5	types.EllipsisType	immutable
6	float	immutable
7	frozenset	immutable
8	int	immutable
9	list	mutable
10	types.NoneType	immutable
11	types.NotImplementedType	immutable
12	range	immutable
13	set	mutable
14	str	immutable
15	tuple	immutable

Day 47: Save a JSON

Write a function called `save_json`. This function takes a dictionary as an argument and saves it on a file in JSON format.

Write another function called `read_json` that **opens** the file that you just saved and reads its content. Use the function below as an argument.

```
names = {'name': 'Carol', 'sex': 'female', 'age': 55}
```

Day 48: Binary Search

Write a function called **`search_binary`** that searches for the number **22** in the following list and returns its index. The function should take two parameters: the list and the item that is being searched for. Use binary search (iterative method).

```
list1 = [12, 34, 56, 78, 98, 22, 45, 13]
```

Day 49: Create a Database

For this challenge, you are going to create a **database** using Python's SQLite. You will **import SQLite** into your script. Create a database called **movies.db**. In that database, you are going to create a table called "**movies**." In that table, you are going to save the following movies:

year	title	genre
2009	Brothers	Drama
2002	Spider Man	Sci-fi
2009	WatchMen	Drama
2010	Inception	Sci-fi
2009	Avatar	Fantasy

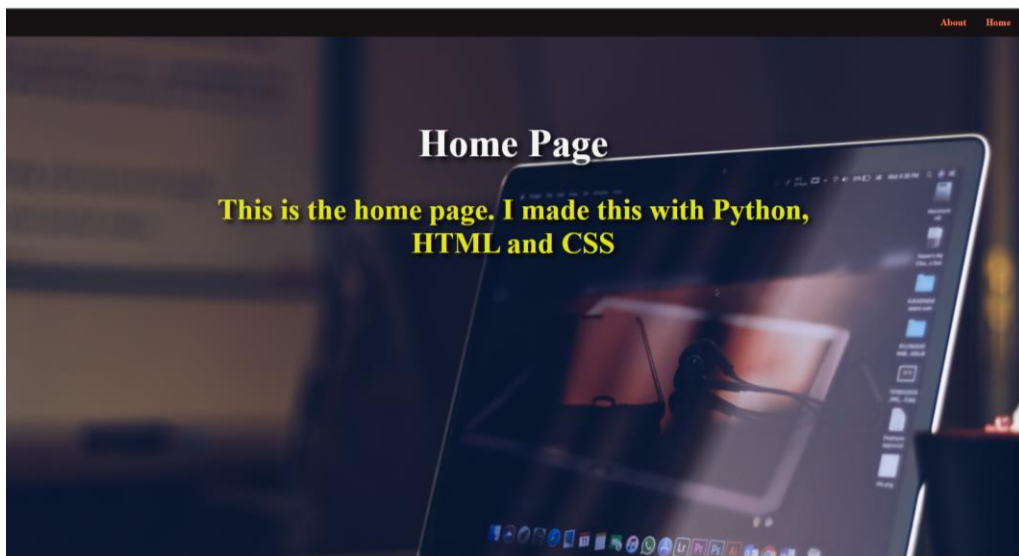
- Once you create a table, run a SQL query to see all the movies in your table.
- Run another SQL query to select only the movie Brothers from the list.
- Run another SQL query to select all movies that were released in 2009 from your table.
- Run another query to select movies in the fantasy and drama genres.
- Run a query to delete all the contents of your table.

Day 50: Create a Flask App

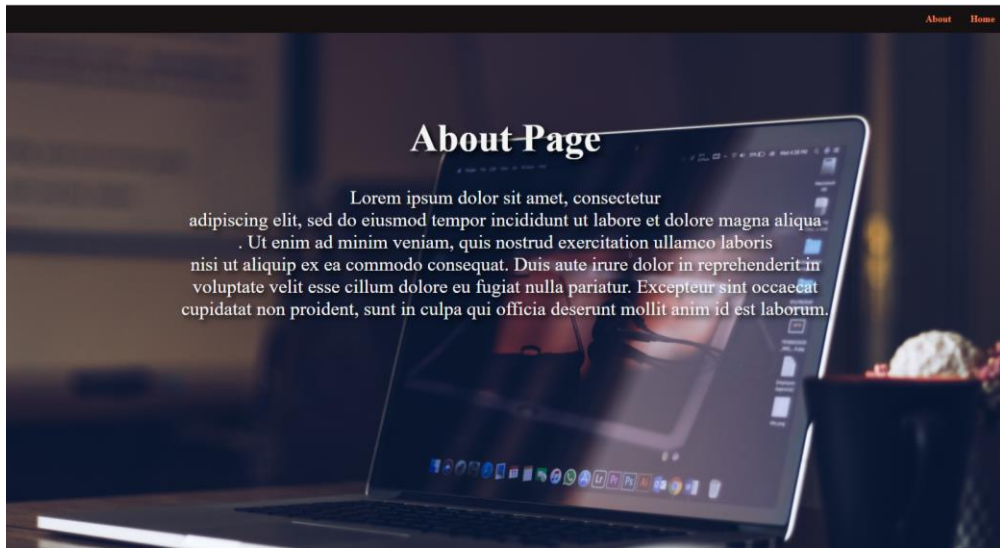
In this challenge, you will get to create an app using **Flask**. Flask is a Python web framework. Learning to build websites with Flask is a very important skill to have if you want to get into web development. You will build an app using **Python**, **HTML**, and **CSS**. I recommend using **Visual Studio code** for this challenge. You can still use whatever editor or IDE you are comfortable with. You are going to create an app with two pages, the **home page** and the **about page**. Your website will have a **navigation bar**, so you can move between the home page and the about page. It will have a background image. The image in the project is by [Hasan Albari](#) from [Pexels](#).

Below is what you should make:

Home Page



About Page



Flask does not come with Python, so you will have to install it on your machine and import it into your script. If you have never made a website before, this challenge will require some research. I suggest using a virtual environment for your project. Good luck. I believe in you.

Answers

Answers to the Challenges

Day 1

In this challenge, we use **conditional statements** to create two conditions. If any one of those conditions is true, that block of code will be executed. The **if statement** checks if the number is divisible by 5, and if it is, we calculate the square root of the number. The **else statement** will execute only if the number is not divisible by 5. In that case, the **else block** will return the remainder of the division.

```
In [1]: def divide_or_square(number):
        if number % 5 == 0:
            sq_root = number ** 0.5
            return f'The square-root of the number is {sq_root:.2f}'
        else:
            remainder = number % 5
            return f'The remainder of the division is {remainder:.2f}'

        print(divide_or_square(10))
```

The square-root of the number is 3.16

Extra Challenge: Longest Value

We use the **max()** function to return the maximum value of the dictionary values. The max function has a **key** parameter. Since we are looking for the first longest value, we pass **len** to the key parameter. If we do not pass **len** to the key parameter, it will return **green** as the longest value, which is lexicographically larger than **apple**.

```
In [2]: def longest_value(d: dict):
        # Using max and key len to get the longest value
        longest = max(d.values(), key=len)
        return longest

        fruits = {'fruit': 'apple', 'color': 'green'}
        print(longest_value(fruits))
```

apple

Day 2

In this challenge, we use **typecasting** to convert the strings in the list into integers using the `int()` function. Once they are converted, we use the built-in **sum** function to sum the integers in list b.

```
In [3]: def convert_add(list1):  
        b = []  
        for i in list1:  
            b.append(int(i))  
        return f'The sum is {sum(b)}'  
  
print(convert_add(['1','3','5']))  
  
The sum is 9
```

If you do not want to use the built-in **sum()** function, you can create a second **for loop** in the function that will iterate through **list2** of integers. We add all the elements of list2 to the variable **count**.

```
In [4]: def convert_add(list1:list):  
        list2 = []  
        count = 0  
        for i in list1:  
            list2.append(int(i))  
        for j in list2:  
            count += j  
        return count  
  
print(convert_add(['1','3','5']))  
  
9
```

Extra Challenge: Duplicate Names

Below, we use a **for loop** and conditional statements to iterate over a list and check if there are items in the list that appear more than once. We count how many times each item appears in the list using the **count()** method. Once we find an item that appears more than once, we first check if the item is in the **duplicates** list; if not, we append the item to the **duplicates** list. We write another **if-statement** to check if the duplicate list is empty or not. If it is not empty, we return the duplicates list; if it is empty, then it means that we do not have duplicates in the list, so we return "No duplicates."

```
In [5]: def check_duplicates(arr: list):
        # Create empty list
        duplicates = []
        for item in arr:
            if arr.count(item)>1:
                if item not in duplicates:
                    duplicates.append(item)
        if duplicates:
            return duplicates
        else:
            return "No duplicates"

fruits = ['apple', 'orange', 'banana', 'apple', 'banana']
names = ['Yoda', 'Moses', 'Joshua', 'Mark']

print(check_duplicates(fruits))
print(check_duplicates(names))

['apple', 'banana']
No duplicates
```

Day 3

First, we create a **count** variable and assign it a value of zero (0). To count the "yes" values in the dictionary, we need to access the dictionary values (reg.values) using a **for loop**. The **for loop** loops through the dictionary looking for "yes" values. Every value is added to the **count** variable.

```
In [6]: register = {'Michael': 'yes', 'John': 'no',
                  'Peter': 'yes', 'Mary': 'yes'}

def register_check(reg: dict):
    # Create a count variable
    count = 0
    for value in reg.values():
        if value == 'yes':
            count += 1
    return 'Number of students in school is', count

print(register_check(register))

('Number of students in school is', 3)
```

Another way to go about it

In this method below, we use the dictionary **items()** method to access the values.

```
In [7]: def register_check(reg: dict):
        count = 0
        for key, value in reg.items():
            if value == 'yes':
                count += 1
        return 'Number of students in school is', count

print(register_check(register))

('Number of students in school is', 3)
```

Extra Challenge: Lowercase names

For this challenge, we are going to use two built-in functions: the **sorted** function and the **tuple** function. The **sorted()** function will sort the list in descending order. To sort a list in descending order, set reverse to True. The **tuple()** function will convert the sorted list into a tuple.

```
In [8]: names = ["kerry", "dickson", "John", "Mary",  
                "carol", "Rose", "adam"]  
  
# creating an empty list  
d = []  
# Using sorted function to sort list in descending order  
for name in sorted(names, reverse=True):  
    if name.islower():  
        d.append(name)  
        tuple_names = tuple(d)  
  
print(tuple_names)  
  
( 'kerry', 'dickson', 'carol', 'adam')
```

Day 4

In this challenge, we use the **type()** function and conditional statements. The **type()** function is used to check the data type of the arguments passed to the function. We also use the comparative operators "**and**", and "**or**" to check whether both arguments are floats, or if only one of them is a float.

```
In [9]: def only_float(a,b):  
        if type(a) == float and type(b) == float:  
            return 2  
        elif type(a) == float or type(b) == float:  
            return 1  
        else:  
            return 0  
  
        print(only_float(12.1, 23))
```

1

Extra Challenge: Index of the Longest Word

The first thing we do, is check if all the strings in the list are of the same length. If so, we **return 0**. If they are not, we search for the longest word using the **len()** and **max()** functions. Once we find it, we search for its index using the **index()** method.

```
In [10]: def word_index(arr: list):  
        for word in arr:  
            # Check if length of words is the same  
            if all(len(word) == len(arr[0]) for word in arr):  
                return 0  
        else:  
            word = len(max(arr))  
            return f'The longest word is at index: {arr.index(word)}'  
  
        words1 = ["Hate", "remorse", "vengeance"]  
        words2 = ["Love", "Hate"]  
        print(word_index(words1))  
        print(word_index(words2))
```

```
The longest word is at index: 2  
0
```

Day 5

In this code, our function takes no arguments. We ask the user for input using the **input()** function. Since the input is in string format, we must first convert it to integer data type using the **int()** function before we can calculate the discount. The output in this code assumes an input price of **150** and a discount of **15%**.

```
In [11]: def my_discount():
         price = int(input('Enter the price: '))
         discount = int(input('Enter the discount: '))
         aft_dsc = price * (100-discount)/100
         return 'Price after discount is ', aft_dsc

         print(my_discount())
```

```
Enter the price: 150
Enter the discount: 15
('Price after discount is ', 127.5)
```

Extra Challenge: Tuple of Student Sex

The first thing to do for this challenge is to ensure that we convert all the strings into lowercase letters, using the **lower()** method. So, we create a new list with names converted into lowercase. Then we search for **males** and **females** in the list. We use the **count** method to count how many times each item appears in the **new_list**. We then combine the **sex** and the number of times it appears in the list into a tuple (**sex, new_list.count(sex)**). We then append the tuples to the student count list. Notice that we use the **break statement** after append, to ensure that we only iterate through the list once.

```

In [12]: students = ['Male', 'Female', 'female', 'male', 'male', 'male',
                    'female', 'male', 'Female', 'Male', 'Female', 'Male', 'female']

def count_students(arr: list) -> list:
    # create an empty list to append lowercase strings
    new_list = []
    student_count = []
    # converting names to lowercase
    # and appending to new_list
    for names in students:
        new_list.append(names.lower())
    # Finding and counting all males in the
    # list and putting it in a tuple
    for sex in new_list:
        if sex == 'male':
            student_count.append((sex, new_list.count(sex)))
            break
    # Finding and counting all females in the
    # list and putting it in a tuple
    for j in new_list:
        if j == 'female':
            student_count.append((j, new_list.count(j)))
            break
    # returning a tuple of students
    return student_count

print(count_students(students))

[('male', 7), ('female', 6)]

```

Day 6

The **split()** function splits the email at the "@" element. Once the email is split, we can use indexing to access the elements of the email. The part that we want to use for the user name is sitting at index 0, which is why we use **[0]** in our code to access it.

```
In [13]: def user_name ():  
        your_email = input("Enter your email: ")  
        user_name = your_email.split('@')[0]  
        return f'Your user name is {user_name}'  
  
        print(user_name())
```

Extra Challenge: Zero Both Ends

This challenge requires a little slicing. Just access the first and last elements in the list and replace them with zeros.

```
In [14]: def zeroed(arr: list) -> list:  
        # Access and modify the first element  
        arr[0] = 0  
        # Access and modify the last element  
        arr[-1] = 0  
        return arr  
  
        list1 = [2, 5, 7, 8, 9]  
  
        print(zeroed(list1))  
  
        [0, 5, 7, 8, 0]
```


Day 7

In this code, using **list comprehension**, we create a list from the range. However, to use the **join()** method on the list, we must ensure that our list elements are in string format. We use the **str()** function to convert x into a string before calling the **join()** function to join the string with dots.

```
In [15]: def string_range(num:int):  
         x = [str(i) for i in range(num)]  
         #Using join method to add dots  
         x = ".".join(x)  
         return x  
  
         print(string_range(6))  
  
0.1.2.3.4.5
```

Extra Challenge: Dictionary of Names

In this challenge, we are asked to create a dictionary of names that start with "S." So, first, we need to find all the names in the list that start with "S," and then we need to return these results in a dictionary. The easiest way to find the names is to use the **startswith()** method. This method will return all names that start with "S." In the code below, we create an empty dictionary, then search for all the elements that start with "S." Using the dictionary method (update), we update the empty dictionary with the results of our search.

```
In [16]: names = ["Joseph", "Nathan", "Sasha", "Kelly",  
                 "Muhammad", "Jabulani", "Sera", "Patel", "Sera"]  
  
d = {} # Creating an empty dictionary  
for name in names:  
    if name.startswith('S'):  
        # Using the dictionary update method  
        d.update({name: names.count(name)})  
print(d)  
  
{'Sasha': 1, 'Sera': 2}
```

Second Method

Another method would be to use the Counter class from the collections module. The Counter class keeps track of elements and their counts and stores the results in a dictionary. It returns the name of the element as a key and its count as a value. See the code below:

```
In [17]: from collections import Counter

count = [] # Creating an empty list
for name in names:
    if name.startswith("S"):
        # Appending names that start with S to the list
        count.append(name)
        # Using the counter method to return a dictionary
names = Counter(count)
print(names)

Counter({'Sera': 2, 'Sasha': 1})
```

Day 8

In this challenge, we use the modulus operator to find even and odd numbers in the list. Then we use the **max()** and **min()** functions to return the **maximum** number and **minimum** number, respectively.

```
In [18]: def odd_even(a):
        even = []
        odd = []
        for i in a:
            if i % 2 == 0:
                even.append(i)
            if i % 2 != 0:
                odd.append(i)
        return max(even) - min(odd)

print(odd_even([1,2,4,6]))
```

5

Extra Challenge: List of Prime Numbers

A prime number is a number that has two factors, 1 and itself. Prime numbers start from 2. For this challenge we use the **range()** function to find range of the number.

```
In [19]: def prime_numbers() -> list:
        prime_num = []
        n = int(input('Please enter a number (integer): '))
        # Creating a range of the input number
        # remember to add a 1 and the end ensure
        # all numbers in the range are covered
        for i in range(0, n + 1):
            # only numbers above 1 are needed
            if i > 1:
                for j in range(2, i):
                    # if a number in the range is divisible by j
                    # Then number is not prime
                    if i % j == 0:
                        break
                else:
                    prime_num.append(i)
        return prime_num

print(prime_numbers())
```

Day 9

Using **list comprehension**, we use a **for loop** and an **if statement** to return a list of odd numbers. Here, you get to use the modulus operator (%). The modulus operator returns the remainder of a division. A number is odd if it returns a remainder when divided by 2. In this code, we use the if statement to return only numbers that are not divisible by 2. The **max()** function will return the biggest odd number in the list.

```
In [20]: def biggest_odd(string1: str):  
         odd_nums = [i for i in string1 if int(i) % 2 != 0]  
         return f' The biggest old number is {max(odd_nums)}'  
  
print(biggest_odd('23569'))
```

The biggest old number is 9

Extra Challenge: Zeros to the End

This challenge is about understanding indexing. The first step is to identify non-zero numbers in the list. Once we identify them, we move them to the back of the list while incrementing the index by 1. We then fill the remaining spots with the zeros on the list.

```

In [21]: def zeros_last(arr: list) -> list:
        i = 0 # setting index 0
        arr1 = arr
        for num in arr:
            # Checking for non-zero numbers
            if num != 0:
                # Moving non-zero numbers to the front of the list
                arr[i] = num
                i += 1

        # Moving zero elements to the end of the list.
        while i < len(arr):
            arr[i] = 0
            i += 1
        return arr

    else:
        # if no zeros return original list in ascending order
        return sorted(arr1)

list1 = [1, 4, 6, 0, 7, 0]
list2 = [2, 1, 4, 7, 6]

print(zeros_last(list1))
print(zeros_last(list2))

[1, 4, 6, 7, 0, 0]
[1, 2, 4, 6, 7]

```

Day 10

In this challenge, we get to use the **input()** function to get information from a user. We then use the **len()** function to get the length of the password. We use the multiplication operator to multiply each element of the string with “*”.

```
In [22]: def your_password():
          password1 = input('Enter password: ')
          password = len(password1) * '*'
          return f'You password is {password} ' \
                 f'and its {len(password)} characters long'

          print(your_password())
```

Extra Challenge: A thousand Separator

The easiest way to do this is to use the format method.

```
In [23]: def convert_numbers(n):
          new_list = []
          for num in n:
              new_list.append("{:,}".format(num))
          return new_list

          print(convert_numbers([1000000, 2356989, 2354672, 9878098]))

          ['1,000,000', '2,356,989', '2,354,672', '9,878,098']
```

Day 11

To compare the two strings, we first need to sort them. We use the **sorted()** function to sort the two strings. Then we use the "==" operator to compare the two strings.

```
In [24]: def equal_strings(st1, st2):
          str1 = sorted(st1)
          str2 = sorted(st2)
          if str1 == str2:
              return True
          else:
              return False

          print(equal_strings('love', 'evol'))
```

True

Extra Challenge: Swap Values

The easiest way to solve this challenge would be to create separate variables for the numbers that you want to swap. Use list slicing to get the values. So we assign the first number in the list to the **"first_number"** variable and the last number in the list to the **"last_number"** variable. We then assign the last number to index 0 and the first number to index 1.

```
In [25]: def swap_values(arr: list):
          # Create a variable for first number
          first_number = arr[0]
          # Create a second variable for the last number
          last_number = arr[-1]
          # assign last number to index 0
          arr[0] = last_number
          # assign first number to index -1
          arr[-1] = first_number
          return arr

          list1 = [2, 4, 67, 7]
          print(swap_values(list1))
```

[7, 4, 67, 2]

Day 12

We use the **split()** method to split the word at the dots. Once the word is split, we use the **len()** function to count the dots.

```
In [26]: def count_dots(word):  
         m = word.split(".")  
         return f'The string has {len(m)-1} dots'  
  
print(count_dots("h.e.l.p."))
```

The string has 4 dots

Extra Challenge: Your Age in Minutes

Remember to **import** the datetime module for this challenge. To keep the code running until a valid year is entered, use the **while loop**. We will use the **int()** function to convert the input from user to integer.

```
In [28]: from datetime import datetime  
  
def your_age():  
    while True:  
        birth_year = input('Enter your year of birth: ')  
        if len(birth_year) != 4:  
            print('Please enter a four digits year')  
        elif int(birth_year) < 1900 or int(birth_year) > 2022:  
            print('Please enter a valid year')  
        else:  
            # This returns the current year  
            now1 = int(datetime.now().strftime("%Y"))  
            age = (now1 - int(birth_year)) * 525600  
            return f'You are {age:,} minutes old.'  
  
print(your_age())
```


Day 13

In this challenge, you get to use typecasting, the **input()** function, and the **try** and **except** blocks to handle errors. We use the **if statement** to check that the inputs from the user are non-negative. If the inputs are non-negative number, we calculate the **total price** and return it. The input from the user is always in string format, so we use the **int()** function to convert it into an integer. The **except** block ensures that we handle the **ValueError**. We have added the **while loop** to ensure that the code runs until valid numbers are entered.

```
In [28]: def your_vat():
        while True:
            try:
                price = int(input("Enter the price of item: "))
                vat = int(input('Enter vat: '))

                if price > 0 and vat > 0:
                    total_price = price + \
                        (price * vat / 100 + 1) - 1
                    return 'The price VAT inclusive is', total_price
                else:
                    print("Please enter non-negative values")
            except ValueError:
                print("Please enter a valid numbers")

        print(your_vat())
```

Extra Challenge: Pyramid of Snakes

With this challenge, we use loops to create a pyramid of snakes. We imported the emoji library to get the snake emoji. For each iteration, we calculate the number of spaces to print before the snake emojis using the formula **range(n , i, -1)**. We add the emojis using **range(0, i + 1)** to create the pyramid shape. See the code below:

```
In [29]: from emoji import emojiize

def python_snakes(n: int):
    # the loop to determine the number of rows of the pyramid
    for i in range(0, n):
        # The loop that determines number of columns
        for j in range(n, i, -1):
            # print space between the snake signs
            print(end=" ")
        for k in range(0, i+1):
            # printing the snake emoji
            print(emojiize(':snake:'), end=" ")
        print("\n")

python_snakes(7)
```

```

  🐍
 🐍 🐍
🐍 🐍 🐍
🐍 🐍 🐍 🐍
🐍 🐍 🐍 🐍 🐍
🐍 🐍 🐍 🐍 🐍 🐍
🐍 🐍 🐍 🐍 🐍 🐍 🐍
```

Day 14

Notice how we use nested **for loops** in the code below? The first **for loop** accesses the outer list, and the inner **for loop** accesses the inner list. Once we access the elements of the inner list, we append them to list 1. With this small code, we have flattened the list.

```
In [30]: def convert_list(lst1: list):
          list1 = []
          for items in lst1:
              for i in items:
                  list1.append(i)
          return list1

          print(convert_list([[2, 4, 5, 6]]))

          [2, 4, 5, 6]
```

Extra Challenge: Teacher's Salary

The output below assumes that the user inputs the following: **name:** John Kelly, **periods:** 105, and **rate:** \$20.

```
In [31]: def your_salary():
          name = input('Enter the name of the teacher: ')
          rate = int(input('Enter rate per period: '))
          period = int(input('Enter the number of periods taught: '))

          if period <= 100:
              gross_salary = rate * period
          else:
              gross_salary = (rate * 100)\
                  + ((period-100) * (rate + 5))

          return f'Teacher: {name}, \nPeriods: ' \
                  f'{period} \nGross salary:{gross_salary:,}'

          print(your_salary())

          Enter the name of the teacher: John Kelly
          Enter rate per period: 20
          Enter the number of periods taught: 105
          Teacher: John Kelly,
          Periods: 105
          Gross salary:2,125
```

Day 15

This challenge tests your skill to use negative string slicing, if statements, and operators in Python. The `::1` operator reverses the string, and the `==` operator compares the original string to the reversed string.

```
In [32]: def same_in_reverse(a):  
         if a == a[::-1]:  
             return True  
         else:  
             return False  
  
         print(same_in_reverse('mad'))
```

False

Extra Challenge: What's My Age?

This challenge is about working with dictionaries. Since all the keys in the dictionary are lowercase, we convert the input from the user to lowercase. Then, we iterate over the dictionary to see if the input name exists in our database. Once we find it, we use the **get()** method to access the dictionary value (which is the age of the person). If the name is not in the dictionary, we use the **while loop** to update the dictionary with the new information from the user. Once we update the dictionary, we return the updated information to the user. See the code below:

```
In [34]: names_age = {"jane": 23, "kerry": 45, "tim": 34, "peter": 27}

def your_age():
    # Convert name to lowercase letters
    name = input("Please enter your name: ").lower()
    # Use for loop to iterate over the dictionary
    for key in names_age.keys():
        if key == name:
            # use the get method to access a specific value.
            return f'Hi, {name}! You are {names_age.get(key)} years old'
    else:
        # if name not in the dictionary
        while name not in names_age.keys():
            age = input("Your name is not in the dictionary, "
                        "please enter your age? ").lower()
            # Update the dictionary with name and age.
            names_age.update({name: age})
            return f'Hi, {name}! You are {names_age.get(name)} years old'

your_age()
```

Day 16

In this exercise, we use the for loop to flatten the nested lists. We have created a variable called **counta**. Every number on the list is added to the count. By the end of the iteration, all the numbers will have been added to the counter.

```
In [35]: def sum_list(lst1: list):
          counta = 0
          for items in lst1:
              for i in items:
                  counta += i
          return 'The sum is ', counta

print(sum_list([[2, 4, 5, 6], [2, 3, 5, 6]]))

('The sum is ', 33)
```

Extra Challenge: Unpack a Nest

For this challenge, we need to write a **nested loop** to access the inner list of the nested list. We then create a new list of all the numbers that we need from the list.

```
In [36]: nested_list = [[12, 34, 56, 67], [34, 68, 78]]

new_list = []
# A nested for loop to access the inner list
for arr in nested_list:
    for num in arr:
        # Create a list of numbers wanted
        if num in [34, 67, 78]:
            # Checking if number already
            # exists before appending
            if num not in new_list:
                new_list.append(num)
print(new_list)

[34, 67, 78]
```

Day 17

In this exercise, you get to use the **random module**. Every time you call the function, the random module issues a random number between 0 and 10. This number is added to the end of the name using the + sign. We use slicing [::-1] to reverse the name. Another way around it would have been to use the **reverse()** method to reverse the name.

```
In [37]: import random

num = random.randint(0,10)

def user_name():
    name = input('Enter name: ')
    name = name[::-1]
    username = name + str(num)
    return username
print(user_name())
```

Extra Challenge: Sort by Length

For this challenge, we find the longest word and then swap it with the shorter words. The shorter words come first, and the longer words come later.

```
In [38]: def sort_length(arr: list):
    for item in range(len(arr)):
        for j in range(len(arr) - 1):
            # Check if any of the words is longer than the other
            if len(arr[j]) > len(arr[j + 1]):
                # swap the longest for the shortest
                arr[j], arr[j + 1] = arr[j + 1], arr[j]

    return arr

names = ["Peter", "Jon", "Andrew"]

print(sort_length(names))

['Jon', 'Peter', 'Andrew']
```

Day 18

In this challenge, you get to learn about ***args**. You can use any word you want as a parameter as long as you use the ***** operator at the beginning of the word. However, it's a common convention to use **args**. The ***args** simply tells the function that we are not sure how many arguments we will need, so the function lets us add as many positional arguments as possible. In the example below, we add two numbers as arguments. However, we can add as many numbers as we want, and the function will work just fine. The ***args** make functions more flexible.

```
In [39]: def any_num (*args):  
         ave = sum(args)/len(args)  
         return f'The average is {ave}'  
  
print(any_num(12, 90))
```

The average is 51.0

In this function, we have added more arguments.

```
In [40]: def any_num (*args):  
         ave = sum(args)/len(args)  
         return f'The average is {ave}'  
  
print(any_num(12, 90, 12, 34))
```

The average is 37.0

Extra Challenge: Add and Reverse

This challenge is best solved using the **range()** function. The first step is to check if the lists are of equal length; we use the **len()** function for that. Once we confirm that the lists are of equal length, we use the range function to get the index of each element in the list. We then add the results and append them to the empty list we created. We then apply the list method, **reverse()** to our new list to reverse the content of the list.

```
In [41]: def add_reverse(n:list, k:list):
          # Creating an empty list
          new_list = []
          if len(n) == len(k):
              for i in range(0, len(n)):
                  # adding and appending corresponding numbers
                  new_list.append(n[i] + k[i])
                  # Reversing new list
                  new_list.reverse()
              return new_list
          else:
              return f'Lists have different lengths'

          # Lists to add and reverse
          list1 = [10, 12, 34]
          list2 = [12, 56, 78]

          print(add_reverse(list1, list2))

[112, 22, 68]
```

Day 19

The **split()** function splits the string into smaller strings. By default, the **split()** function uses any white spaces to split a string. Once the string is split, we use the **for loop** to iterate through the strings and append them to the **words** variable. The **len()** function counts how many items are on our list.

```
In [42]: def count_words(arr: str):  
        words = []  
        for word in arr.split():  
            words.append(word)  
        return f'There are {len(words)} ' \  
               f'words in the sentence'  
  
        print(count_words('I love learning'))
```

There are 3 words in the sentence

Second function

Since the **len()** function counts white spaces as elements, we use the **replace()** method to remove white spaces from the string before using the **len()** function.

```
In [43]: def count_characters(a):  
        a = a.replace(' ', '')  
        return f'The string has ' \  
               f'{len(a)} elements '  
  
        print(count_characters('I love learning'))
```

The string has 13 elements

Day 20

The **enumerate()** function returns the element and its index while iterating through the string. The **capitalize()** method will capitalize the first letter of a string. In this code, we use the **islower()** method to check which word in our string has lowercase letters. We then use the **capitalize()** method to capitalize these words and append them to a list. We then use the **join()** method to join the elements in our output list into a single string.

```
In [44]: def capitalize(a: str):
          upper = []
          for i, word in enumerate(a.split()):
              if word[0].islower():
                  upper.append(word.capitalize())
              else:
                  upper.append(word)
          return ' '.join(upper)

          print(capitalize('i like learning'))
```

I Like Learning

Extra Challenge: Reversed List

For this challenge, we import the string module to get a list of uppercase letters (**string.ascii_uppercase**). We check if any of the words in the string have uppercase letters, and if they do, we append the words to the **upper_names** list. We then use reverse (**[::-1]**) to reverse the words in the list.

```
In [45]: import string

          str1 = 'leArning is hard, bUt if You appLy youRself ' \
                'you can achieVe success'

          upper_names = []
          for word in str1.split():
              for letter in word:
                  # Using string module to find uppercase letters
                  if letter in string.ascii_uppercase:
                      upper_names.append(''.join(word[::-1]))
          print(upper_names)

          ['gninrAel', 'tub', 'uoY', 'yLppa', 'flesRuoy', 'eVeihca']
```

Day 21

The easiest way to deal with this problem is to use the built-in **zip()** function in Python. First we use the conditional statement to check the length of the two lists. If they are not of equal length, we raise a `ValueError`. The **zip()** function combines the two lists into pairs of tuples. We use the **list()** function to create a list, and the **tuple()** function to put the numbers in tuples.

```
In [46]: def make_tuples(a,b):
         if len(a) != len(b):
             raise ValueError("Your lists are not of equal length")

         zipped_object = zip(a,b)
         return list(tuple(zipped_object))

print(make_tuples([1,2,3,4],[5,6,7,8]))

[(1, 5), (2, 6), (3, 7), (4, 8)]
```

Another way is to use a **for loop** statement and the range function. We will iterate over the range object and retrieve the index of the lists. We will use the index to access the values of lists **a** and **b**, and append them to the **list_of_tuples**.

```
In [47]: def make_tuples(a,b):
         if len(a) != len(b):
             raise ValueError("Your lists are not of equal length")

         list_of_tuples = []
         for i in range(len(a)):
             list_of_tuples.append((a[i],b[i]))
         return list_of_tuples

print(make_tuples([1,2,3,4],[5,6,7,8]))

[(1, 5), (2, 6), (3, 7), (4, 8)]
```

Extra Challenge: Even Number or Average

In this challenge, we create two empty lists. In the first list (**avg_num**), we append all the numbers from the user; in the other list, we only input even numbers (**even_number**). The code then checks the even numbers list to see if it is empty; if it is empty, the code goes to the **avg_number** list and calculates the average.

```
In [47]: def even_or_average():
    avg_num = []
    even_number = []
    # Using while loop to ensure code keeps running
    while True:
        number = input("Please enter numbers or done: ")
        avg_num.append(int(number))
        if int(number) % 2 == 0:
            even_number.append(number)
            # Once the user inputs five numbers the code breaks
            if len(avg_num) == 5:
                break
    if len(even_number) > 0: # checking if list empty
        return f'The largest even number: {max(even_number)}'
    else:
        return f'The average is : {sum(avg_num) / len(avg_num):.2f}'
print(even_or_average())
```

Day 22

For this challenge, you have to use the **string()** method and **join()** method to add the hash (#). Then you use another string method, **replace()**, to replace the hash with an underscore (_), and we also use the **replace()** method to replace the underscore with nothing.

```
In [49]: def add_hash(a: str):  
         return "#".join(a)  
  
         def add_underscore(a: str):  
             return str(a).replace("#", "_")  
  
         def remove_underscore(a: str):  
             return str(a).replace("_", "")  
  
         print(remove_underscore(add_underscore(add_hash('Python'))))  
  
Python
```

Day 23

This is a simple calculator that performs simple arithmetic operations. We have used the operator module for the math operations. We have added the **try** and **except** blocks to ensure that we handle some exceptions. There are so many ways you can make a calculator. You may have made it different or even better.

```
In [50]: import operator

def calculator():
    try:
        num1 = int(input("Enter number: "))
        # asking the user to pick an operator
        opt = input("Pick operator(+,-,*,/) : ")
        num2 = int(input('Enter another number: '))
        if opt not in ['+', '-', '*', '/'] or len(opt) > 1:
            print('Please enter a valid operator')
    except ValueError:
        print('Please enter a valid number')
    except ZeroDivisionError:
        print('You cannot divide a number by zero.Try again')
    else:
        if opt == '+':
            return f'ans is: {operator.add(num1, num2)}'
        elif opt == '-':
            return f'ans is: {operator.sub(num1, num2)}'
        elif opt == '*':
            return f'ans is: {operator.mul(num1, num2)}'
        elif opt == '/':
            return f'ans is: {operator.truediv(num1, num2)}'
        return 'Try again'

print(calculator())
```

Extra Challenge: Multiply Words

In this function below, we first split the input string into individual words using the **split()** method. Then, we iterate over each word in the words list. If a word contains all lowercase letters (checked using **islower()**), we append the length to the **words_len** variable. Finally, we use **prod()** method to multiply the lengths in the **words_len** list.

```
In [51]: import math

def multiply_words(s: str):
    words_len = []
    for word in s.split():
        if word.islower():
            words_len.append(len(word))
            # Using the math prod to multiply the lengths
            words_prod = math.prod(words_len)
    return 'The prod of the word lengths is', words_prod

# strings
str2 = "love live and laugh"
str3 = "Hate war love Peace"

print(multiply_words(str2))
print(multiply_words(str3))

('The prod of the word lengths is', 240)
('The prod of the word lengths is', 12)
```

Another way

If you do not want to use the prod method from the math module, you can multiply the lengths of each word by the words_len variable. See code below:


```
In [52]: def multiply_words(s: str):
        words_len = 1

        for word in s.split():
            if word.islower():
                words_len *= len(word)
        return 'The prod of the word lengths is', words_len

# strings
str2 = "love live and laugh"
str3 = "Hate war love Peace"

print(multiply_words(str2))
print(multiply_words(str3))

('The prod of the word lengths is', 240)
('The prod of the word lengths is', 12)
```

Day 24

This simple average calorie calculator will run until the user enters "done," then it will calculate the average calories. The input from the user is added to the **scores** variable. We use the **sum()** function and the **len()** function to calculate the average calories in the "**scores**" list. The while loop is what keeps the function running until something happens that makes it break out of the loop. Once we hit "done," the loop is halted by the break statement.

```
In [53]: def average_calories():
        scores = []
        while True:
            score = input('Enter a score or done when quit: ')
            if score == 'done':
                break
            scores.append(int(score))
        return f" Mean of scores is " \
            f"{sum(scores) / len(scores):.2f}"

print(average_calories())
```

Extra Challenge: Create a Nested List

The first thing is that we pass ***args** as a parameter to ensure that our function can take any number of positional arguments. Then we use a for loop to check if the arguments are of list data type. If they are not we will return "Invalid arguments . Please check your arguments". We create an empty list (**list1**) to append the lists to. We use the **for loop** and the **range()** function to keep track of the number lists passed as arguments. We use another **for loop** to iterate through the lists and append them to the empty list.

```
In [54]: def nested_lists(*args: list):
        for lst in args:
            if type(lst) != list:
                return "Invalid arguments. Please check your arguments."

        list1 = []
        for i in range(len(args)):
            for j in args:
                list1.append(j)
                break

        return list1

print(nested_lists([1, 2, 3, 5], [1, 2, 3, 4], [1, 3, 4, 5]))

[[1, 2, 3, 5], [1, 2, 3, 4], [1, 3, 4, 5]]
```

Day 25

Below the **all_the_same** function takes one argument, *data*, which can be a **string**, **list**, or **tuple**. Inside the function, we first check if the data is an instance of either a string, list, or tuple using the **isinstance()** function. If it's not one of those types, we raise a **TypeError**.

If data is a string, list, or tuple, we use a list comprehension and the **all()** function to check if all the elements in data are equal to the first element (*data[0]*). The **all()** function returns **True** if all elements in the iterable are **True**, and **False** otherwise.

```
In [55]: def all_the_same(data: str or list or tuple):
        if isinstance(data, (str, list, tuple)):
            check_lists = all(i == data[0] for i in data)
            return check_lists
        else:
            raise TypeError("Input must be a string, a list, or a tuple.")

        print(all_the_same(['Mary', 'Mary', 'Mary']))

True
```

Extra Challenge: Reverse a String

In this challenge, we split the string on whitespaces using the **split** method. We append the strings to the empty list (list *x*) and reverse the words using **[::-1]**. We then get the words from the list and join them using the join method.

```
In [57]: str1 = "the love is real"

def read_backwards(n: str) -> str:
    # Create an empty list
    x = []
    # split string on whitespaces and reverse it
    for i in n.split()[::-1]:
        x.append(i)
    # using the join method to join string
    return ' '.join(x)

print(read_backwards(str1))

real is love the
```

Day 26

In this challenge, we use the **replace()** method to remove the white spaces between the words. Once the spaces are removed, we use the **for loop** to iterate through the string and append the items to the **list1** variable. We use the **not-in** operator on **list1** to ensure that we do not add duplicates to the list. We use the **join()** method to add commas between the string elements.

```
In [58]: def sort_words(sentence):  
         list1 = []  
         sentence = sentence.replace(' ', '')  
         for i in sentence:  
             if i not in list1:  
                 list1.append(i)  
         list1.sort()  
         sorted_words = ','.join(list1)  
         return sorted_words  
  
print(sort_words('love life'))  
  
['e,f,i,l,o,v']
```

Extra Challenge: Length of Words

This challenge requires that we create a dictionary of word lengths. The word will be the **key**, and the length of the word is the **value**. Below, we use the **split()** function to split the string into a list of strings. We then iterate through the list and **update** our empty dictionary(dict1) with the words (key) and length of the words (value). See the code below:

```
In [59]: def string_length(s: str) -> dict:
        list1 = []
        dict1 = {}
        # converting string into a list of strings
        for word in s.split():
            list1.append(word)
            # update the dictionary with word lengths
        for word in list1:
            dict1.update({word: len(word)})
        return dict1

s = 'Hi my name is Richard'

print(string_length(s))

{'Hi': 2, 'my': 2, 'name': 4, 'is': 2, 'Richard': 7}
```

Day 27

For this challenge, once we append the unique numbers to **list2**, we use the **sum()** function to sum the numbers in **list1**, and **list2**. We use the modulus operator (%) to check if the difference between the two numbers is an even number or an odd number. If it is an even number, we return **list1**, otherwise we return **list2**.

```
In [61]: def uniq_numbers(list1: list):
        list2 = []
        for number in list1:
            if number not in list2:
                list2.append(number)
        dif = sum(list1) - sum(list2)
        if dif % 2 == 0:
            return list1
        else:
            return list2

print(uniq_numbers([1, 2, 4, 5, 6, 7, 8, 8]))

[1, 2, 4, 5, 6, 7, 8, 8]
```

Extra Challenge: Difference of Two Lists

We use two list-comprehension statements: one to find items in **arr1** that are not in **arr2**, and another to find items in **arr2** that are not in **arr1**.

```
In [61]: def difference(arr1: list, arr2: list):
        # Find items in list 1 not in list 2
        list1 = [i for i in arr1 if i not in arr2]
        # Find items in list 2 one not in list 1
        list2 = [j for j in arr2 if j not in arr1]
        # concatenate the two lists
        return 'The difference between two sets is ', list1 + list2

# Two lists
list1 = [1, 2, 4, 5, 6]
list2 = [1, 2, 5, 7, 9]

print(difference(list1, list2))

('The difference between two sets is ', [4, 6, 7, 9])
```

Day 28

Using the **enumerate()** function, we can retrieve the item index or position. The **isupper()** method returns **True** if a letter in the string is capitalized. Once we find the capitalized letter, we want its index to be appended to the **index** list.

```
In [62]: def index_position (lst):
        index = []
        for i, item in enumerate(lst):
            if item.islower():
                index.append(i)
        return index

        print(index_position('Love'))
```

```
[1, 2]
```

Extra Challenge: Largest Number

This challenge requires that we use some string methods, such as **strip** and **replace**, to convert the number into an integer. Here is the full code below:

```
In [63]: def largest_number(arr: list):
        # Create empty list
        list1 = []
        # remove brackets and spaces in the lists
        n = str(arr).strip('[,]').replace(',', '').replace(' ', '')
        # append the numbers to the empty list
        for i in n:
            list1.append(int(i))
        # Sorting the list in descending order
        list1.sort(reverse=True)
        # removing parenthesis and spaces from the sorted
        # list and converting it to int
        large_number = int(''.join(map(str, list1)))
        # return a large formatted number
        return 'The largest number is, {:,}'.format(large_number)

        lst = [3, 67, 87, 9, 2]

        print(largest_number(lst))
```

```
The largest number is, 9,877,632
```


Day 29

The first step in this challenge is to concatenate the two strings and find the length of the new string using the **len()** function. Remember to remove whitespace between the substrings. To find the middle element, we use the floor division (`//`). The floor division will round off to the nearest integer, which is 4. The element sitting in position 4, or index 4, is the middle element. Only strings with odd lengths will return a middle element.

```
In [64]: def middle_figure(a: str,b: str):
          modified_str = (a + b).replace(' ', '')
          len_modified_str = len(modified_str)
          middle = len_modified_str // 2
          if len(modified_str) % 2 == 1:
              return "The middle figure is", modified_str[middle]
          else:
              return 'No middle figure'

          print(middle_figure('make love', 'not wars'))

('The middle figure is', 'e')
```

Day 30

The collections module has a **Counter()** class that we can use to find the most common element of a list. We use it here, and it returns Peter, which appears in the list three times.

```
In [65]: from collections import Counter

name = ["John", "Peter", "John", "Peter", "Jones", "Peter"]

def frequent_name(a):
    return max(Counter(a).most_common(1))

print(frequent_name(name))

('Peter', 3)
```

Extra Challenge: Sort by Last Name

Below, we take a list of names, and split each name into a list of words, and then sort the list based on the last name using the **sorted()** function with a custom key function `x`. The sorted names are then appended to a new list **list3** by joining the last name with the first name in the desired format.

```
In [66]: names = ['Beyonce knowles', 'Alicia Keys',
                  'Katie Perry', 'Chris Brown', 'Tom Cruise']

def sorted_names(list1):
    list2 = []

    for i in list1:
        list2.append(i.split())

    list3 = []
    # creating a key for the sorted function
    x = lambda x: x[-1]
    for j in sorted(list2, key=x):
        # appending sorted names to list3
        list3.append(' '.join([j[-1], j[0]]))
    return list3

print(sorted_names(names))

['Brown Chris', 'Cruise Tom', 'Keys Alicia', 'Perry Katie', 'knowles Beyonce']
```

Day 31

In this challenge, we use the **append()** method to append the strings and their length to list **b**. If you print list **b** you will get a nested list of each word and its length. The **max()** function finds the longest word in list **b**.

```
In [67]: def longest_word(a):  
         b = []  
         for word in a:  
             b.append([len(word), word])  
         return max(b)  
  
print(longest_word(['Java', 'Javascript', 'Python']))  
  
[10, 'Javascript']
```

Extra Challenge: Create User

In this challenge, we create an empty dictionary and use the **update method** to update it with information from the user. We then use a **while loop** to ensure the code runs until the correct user details are entered.

```
In [68]: def create_user():  
         d = {} # create an empty dictionary  
         name = input("Enter your name: ")  
         age = int(input("Enter your age: "))  
         password = input("Enter your password: ")  
         # Updating the dictionary using update method.  
         d.update({'name': name})  
         d.update({'age': age})  
         d.update({'password': password})  
         print("User saved. Please login")  
         # using while loop to ensure the code runs  
         # until user enters correct login details  
         while True:  
             user_name = input("Please enter your user name to login")  
             password = input("Please enter your password")  
             if user_name == d.get('name') and password == d.get('password'):  
                 return "Logged in successfully"  
             else:  
                 print('Wrong password or user_name please try again')  
  
print(create_user())
```

Day 32

In this code, we use the **while loop** to ensure that the code runs until a valid password is entered. All errors in the input password are appended to the **errors** list. The **any()** function checks if any of the characters in the password satisfy a given condition. If a condition is not satisfied, an error is raised.

```
In [69]: def password_checker():
        errors = []
        while True:
            password = input('Enter your password: ')
            if not any(i.isupper() for i in password):
                errors.append("Please add at least one "
                              "capital letter to your password")
            elif not any(i.islower() for i in password):
                errors.append("Please add at least one "
                              "small letter to your password")
            elif not any(i.isdigit() for i in password):
                errors.append('Please add at least one '
                              'number to your password')
            elif len(password) < 8:
                errors.append("Your password is less "
                              "than 8 characters")
            if len(errors) > 0:
                print('Check the following errors:')
                print(str(errors))
                del errors[0:]
            else:
                return f'Your password is {password}'

        print(password_checker())
```

Extra Challenge: Valid Email

This code allows the user to enter a password and validates it against specific criteria, which is the presence of uppercase and lowercase letters, digits, and a minimum length. It accumulates any validation errors encountered and provides feedback to the user about the specific requirements that their password does not meet. If the password passes all the checks, it returns a success message. We use for loops and conditional statements to check if the email is valid

```
In [70]: def email_validator(arr: list):
# Creating an empty list.
valid_emails = []
for email in arr:
    # conditions that make email valid
    if "@" in email and email.count('@') == 1 and email[-4:] == '.com':
        if email[0] != '@':
            valid_emails.append(email)
    # Checking if list with emails is empty.
elif len(valid_emails) == 0:
    return 'All emails are invalid'

return f'The list of valid emails is: {valid_emails}'

emails = ['ben@mail.com', 'john@mail.cm', 'kenny@gmail.com', '@list.com', ]
print(email_validator(emails))

The list of valid emails is: ['ben@mail.com', 'kenny@gmail.com']
```

Day 33

Using a **for loop**, we look for elements that are present in both lists. The **list comprehension** will return a list of elements that are present in **list1** and **list2**. Once we find the numbers present in both lists, we will swap the numbers at index 0 and -1 in the resulting intersection list and convert the list into a tuple using the tuple method.

```
In [71]: def inter_section(a, b):
        inter_list = [num for num in a if num in b]
        # Create a variable for first num
        first_num = inter_list[-1]
        # Create a variable for last num
        last_num = inter_list[0]
        # Assign first num to index 0
        inter_list[0] = first_num
        # Assign last num to index -1
        inter_list[-1] = last_num
        return tuple(inter_list)

        list1 = [20, 30, 60, 65, 75, 80, 85]
        list2 = [ 42, 30, 80, 65, 68, 88, 95]

        print(inter_section(list1, list2))

        (80, 65, 30)
```

Extra challenge: Set vs list

To test the speed of the code, we are going to use the **timeit function** for both codes. We create two variables, **set_test** and **list_search**. We then pass this variable to the **timeit** function, and we set the number of executions to 1000. Here is the code below:

```
In [72]: import timeit

# Set search
set_test = '''
a = range(1000000)
b = set(a)
i = 999999
i in b
'''

set_search_time = timeit.timeit(stmt=set_test, number=1000)

# List search
list_test = '''
a = range(1000000)
b = list(a)
i = 999999
i in b
'''

list_search_time = timeit.timeit(stmt=list_test, number=1000)

print("Set Search Time:", set_search_time)
print("List Search Time:", list_search_time)

Set Search Time: 70.20032260008156
List Search Time: 40.85908710001968
```

From the results, list search is performing faster than set search. Note that the performance of sets and lists can vary depending on the specific use case and the nature of the data being processed. In general, sets are designed for efficient membership testing, while lists provide more flexibility for element access and manipulation.

Day 34

When opening files, it is good practice to open them with the **with statement**. This means that you don't have to close the file at the end of the operation, as the **with statement** will automatically close the file. In this code, we open the file in **read** mode. We use the **split()** method to split the words in the text into strings. By default, the split method splits the text on the whitespaces. The **isdigit()** method checks which string elements are **non-numeric**. These non-numeric elements are appended to the list1 variable.

```
In [73]: def just_digits():
        with open('python.csv', 'r') as file:
            open_file = file.read().split()
            list1 = []
            for item in open_file:
                if item.isdigit():
                    list1.append(item)
            return list1

print(just_digits())

['1991', '2', '2000', '3', '2008']
```


Day 35

We use the **string module** to get the alphabet letters in lowercase. We then remove the whitespaces from our string using the **replace()** method. We use the **for loop** to loop through the string, remove duplicates, and append the elements of the string to **list1**. We sort the elements of **list1** in alphabetical order using the **sort()** method, and then use the **join()** method to join the string elements. We then compare the sorted sentence to the alphabet. If the sentence has all the elements in the alphabet variable, the code will return **True**.

```
In [74]: import string

def check_pangram(sentence):
    list1 = []
    alphabet = string.ascii_lowercase
    # removing white spaces in the string
    sentence = sentence.replace(' ', '')
    for i in sentence:
        if i not in list1:
            list1.append(i)
    list1.sort()
    sentence = ''.join(list1)
    if alphabet == sentence:
        return True
    else:
        return False

print(check_pangram('the quick brown fox '
                    'jumps over a lazy dog'))
```

True

Extra Challenge: Find My Position

In this challenge, it would be easy to use the **enumerate** function. Since we are looking for the index of a number, the enumerate function will return the index of all the numbers in the list. We use the (if j == n) conditional statement to return only indexes of the given number. If the number is not in the list, we convert all the numbers in the list to the given number.

```
In [75]: def find_index(arr: list, n: int) -> list:
        list1 = []
        # Using enumerate to find index of integer
        for i, j in enumerate(arr):
            if j == n:
                list1.append(i)
            # If integer not in list
            if n not in arr:
                for j in arr:
                    list1.append(n)
            return list1

        return list1

lst1 = [1, 2, 4, 6, 7, 7]

print(find_index(lst1, 7))
print(find_index(lst1, 8))
```

```
[4, 5]
[8, 8, 8, 8, 8, 8]
```

Day 36

This code creates a dictionary from a string. The code uses a nested **for loop**. The first **for loop** returns the index elements of the string. The second **for loop** also returns the index. If the element index of loop one is equal to that of loop 2, 1 is added to the count variable. The **if statement** is used to update the dictionary, d.

```
In [76]: def count(str1):
        d = {}
        for i in range(len(str1)):
            x = str1[i]
            count = 0
            for j in range(i, len(str1)):
                if str1[j] == str1[i]:
                    count = count + 1
            countz = dict({x: count})
            # updating the dictionary
            if x not in d.keys():
                d.update(countz)
        return d

print(count('hello'))

{'h': 1, 'e': 1, 'l': 2, 'o': 1}
```

Extra Challenge: Sum Nested Dictionary

In the above code, the **sum_nested_dict** function takes a dictionary as an argument. It iterates over the values of the dictionary, and for each value, it checks if it is another dictionary using the **isinstance()** function. If it is a dictionary, the function makes a recursive call to **sum_nested_dict** to calculate the sum of the nested dictionary. If it is not a dictionary, the value is added to the **total_sum**. Finally, the function returns the **total_sum**. We are using recursion to ensure that the function can retrieve the values at any depth and calculate the sum

```
In [77]: def sum_nested_dict(dictionary: dict):
        total_sum = 0
        for value in dictionary.values():
            if isinstance(value, dict):
                # Recursive call for nested dicts
                total_sum += sum_nested_dict(value)
            else:
                total_sum += value
        return total_sum

nested_dict = {'a': 1, 'b': {'c': 2, 'd': {'e': 3, 'f': 4}}, 'g': 5}

sum_of_values = sum_nested_dict(nested_dict)
print("Sum of all values:", sum_of_values)

Sum of all values: 15
```

Day 37

The code checks if the letters in the string argument are vowels. First, we check if the input argument is of the string data type. If it is not of the string data type, the function will return **"invalid input."** We use a **for loop** to loop over a string and capture vowels. These vowels are appended to the `vowels_in_str` variable. If the `vowels_in_str` is not empty, we return the length of the list; otherwise, we return **"The string has no vowels."**

```
In [78]: def count_vowels(str1):  
  
    vowels_in_str = []  
  
    if type(str1) != str:  
        return "Invalid input"  
  
    for letter in str1:  
        if letter in "AEIOUaeiou":  
            if letter not in vowels_in_str:  
                vowels_in_str.append(letter)  
  
    if vowels_in_str:  
        return len(vowels_in_str)  
    else:  
        return "The string has no vowels"  
  
print(count_vowels('hello'))  
print(count_vowels('saas'))  
print(count_vowels('sly'))  
print(count_vowels(3))  
  
2  
1  
The string has no vowels  
Invalid input
```

Day 38

We use the **random** module to issue a random number between 0 and 10. We use the **while loop** to ensure the code keeps running until the condition becomes False. The condition becomes False when the user guesses the right number or runs out of guesses. The user gets a maximum of three (3) guesses.

```
In [79]: import random

random_number = random.randint(0,10)

def guess_number():
    c = 0
    while c < 4:
        guess = int(input("Guess a number "
                           "between 1 and 10: "))

        c += 1
        if c == 3:
            print("You have run out of guesses. "
                  "You lose")
            break
        elif guess > random_number:
            print('Your guess is too high. '
                  'Try again')
        elif guess < random_number:
            print('Your guess is too low. '
                  'Try again')
        else:
            return 'Correct. You win'
    return ''

print(guess_number())
```

Extra Challenge: Find Missing Numbers

There are many ways you can solve this challenge. Here we using the **range()** function to find the missing numbers in the sequence. We create a range object from the beginning of the list to the end. We search for missing values in the object and append them to the **missing_numz** variable.

```
In [80]: list3 = [1, 2, 3, 5, 6, 7, 9, 11, 12, 23, 14, 15, 17,
                18, 19, 20, 21, 22, 24, 25, 26, 27, 28, 31]

def missing_numbers(arr: list) -> list:
    missing_numz = []
    # find the missing numbers using range
    for i in range(arr[0], arr[-1] + 1):
        if i not in arr:
            missing_numz.append(i)

    return missing_numz

print(missing_numbers(list3))

[4, 8, 10, 13, 16, 29, 30]
```

Day 39

For this challenge, we are using two modules: the **string** module and the **random** module. The string module provides all the characters we need for the password. We use the random module (**random.choice**) to shuffle variable **a**. We ask the user to pick the length of the password they want, so after we shuffle the characters of the password, the issued password is the length requested by the user.

```
In [81]: import string
import random

def password_generator():
    # string module constants
    a = string.ascii_letters + \
        string.digits + string.punctuation
    password1 = []
    length = input("Pick your password length "
                  "a,b or c: \na. weak \nb.strong \nc."
                  "Very strong...: ")

    if length == 'a':
        length = 5
    elif length == 'b':
        length = 8
    elif length == 'c':
        length = 12
    for i in range(length):
        password = str(random.choice(a))
        password1.append(password)
    return 'You password is', ''.join(password1)

print(password_generator())
```


Day 40

Using the **enumerate()** function, we obtain the index of each letter in the string. The **split()** method turns the words in the string into strings. We use the if statement to check if the first letter of the word is a vowel. If it is a vowel, we append "yay" at the end. If it is not, we move the first letter to the end and append "ay."

```
In [82]: def pig_latin(a):
          output = []
          for i, word in enumerate(a.split()):
              if word[0] in 'aeiou':
                  output.append(word[i] + 'yay')
              else:
                  output.append(word[1:] + word[0] + 'ay')
          return ' '.join(output)

          print(pig_latin('i love python'))

          iyay ovelay ythonpay
```

Day 41

Notice that we are using the **nested for loop** to find the letters in the words. We iterate over the words to check if any of the letters in the words are vowels. If a word has a vowel in it, we append it to the **vowels list**. In the **vowels list**, we only have words that have vowels in them.

```
In [83]: def vowels_count(a):  
         vowels = []  
         for word in a.split():  
             for i in word:  
                 if i in 'aeiou':  
                     if word not in vowels:  
                         vowels.append(word)  
         return vowels  
         print(vowels_count('You have no rhythm'))  
  
         ['You', 'have', 'no']
```

Extra Challenge: Class of Cars

In this challenge, we create the **Cars class**. This class is the parent class of all car brand classes. This is called **inheritance**. The child classes inherit the properties and methods of the parent class. The child classes have "Cars" as an argument.

```

In [84]: class Cars:
    def __init__(self, model, color, year,
                 transmission, electric=False):
        self.model = model
        self.color = color
        self.year = year
        self.transmission = transmission
        self.electric = electric
    # Creating the instance method
    def print_car(self):
        return f'car_model = {self.model}\nColor = ' \
               f'{self.color} \nYear = {self.year}' \
               f' \nTransmission = {self.transmission} ' \
               f'\nElectric = {self.electric}'

class BMW(Cars):
    def __init__(self, model, color, year,
                 transmission, electric=False):
        Cars.__init__(self, model, color, year,
                      transmission, electric=electric)

class Tesla(Cars):
    def __init__(self, model, color, year,
                 transmission, electric=False):
        Cars.__init__(self, model, color, year,
                      transmission, electric=electric)

class Ford(Cars):
    def __init__(self, model, color, year,
                 transmission, electric=False):
        Cars.__init__(self, model, color, year,
                      transmission, electric=electric)

# instantiating class objects
ford1 = Ford("Focus", "White", 2017, "Auto")
print(ford1.print_car())
tesla1 = Tesla("S", "Grey", 2016, "Manual", True)
print(tesla1.print_car())
bmw1 = BMW('X6', 'silver', 2018, 'Auto')
print(bmw1.print_car())

```

Day 42

The **textblob module** is being used to validate the user's input. If the word is not spelled correctly, the module suggests a correct spelling to the user. If the user accepts the word, the word is returned. If the user does not accept the word, the code asks the user to enter another word. We are using a while loop, so this code will only break once the user inputs a correct word or accepts the correctly spelled word suggested by the code.

```
In [85]: from textblob import TextBlob

def spelling_checker():
    while True:
        word = input('Enter a word : ')
        # checking if the input word is correct
        if word != TextBlob(word).correct():
            # suggesting the correct word to the user
            question = input(f'Did you mean '
                             f'{TextBlob(word).correct()}?. '
                             f'type yes/no :')

            if question == 'yes':
                break
            else:
                print('Please try again')
        # if the word entered is correct
        # the code breaks and returns word
        elif word == TextBlob(word).correct():
            break
    return f'Your word is, {TextBlob(word).correct()}'

print(spelling_checker())
```

Extra Challenge: Set Alarm

We first import the **winsound** module to provide the sound when the alarm goes off. The **datetime** module is imported to work with date and time. We use **while** loops to validated inputs from user. The **while** loops will keep the code running until the user enters a valid hour value in the range of 0-23; and a validate minute value in the range of 0-59. If the input is not valid an error message is displayed and the user is asked to enter valid inputs.. We format the time (**alarm_time**) by combining the inputted hour and minute using the **format()** method. When the **alarm_time** matches the now time, the alarm goes off.

```
In [86]: import winsound
import datetime

def set_alarm():
    # Input Validation
    while True:
        hour = input("Please enter the hour (0-23): ")
        if hour.isdigit() and 0 <= int(hour) <= 23:
            break
        print("Invalid hour. Please enter a valid hour.")

    while True:
        minute = input("Please enter the minute (0-59): ")
        if minute.isdigit() and 0 <= int(minute) <= 59:
            break
        print("Invalid minute. Please enter a valid minute.")

    # String Formatting
    alarm_time = "{:02d}:{:02d}".format(int(hour), int(minute))
    print("You have set the alarm for", alarm_time)

    while True:
        now = datetime.datetime.now().strftime("%H:%M")
        if alarm_time == now:
            print("Wake up")
            print("Wake up")
            print("Wake up")
            winsound.PlaySound("SystemExit", winsound.SND_ALIAS)
            break

set_alarm()
```

Day 43

In this code, we ask the user to input the name of the student and their marks in the test. First, we check the **name** input to ensure that it contains only valid inputs. We use the string library to check if the input name contains **punctuation** values or **numbers** (digits). If it does, we raise a **NameError**. If a user inputs an invalid value, the code will raise a **ValueError**, which is handled in the **except** block. The code only breaks once the user enters "done." When that happens, the dictionary is returned. We use the **get()** method (d.get) to add values to the dictionary. The **get()** method is used to retrieve the existing value for the key (marks) and adds the new marks to it. If the key doesn't exist, it initializes it with 0 and adds the marks. We use the **while loop** to keep the code running until a valid input is made.

```
In [87]: import string

def get_marks():
    dict1 = {}
    while True:
        try:
            name = input('Enter the student name or "done" when finished: ')
            for letter in name:
                if letter in string.punctuation or letter in string.digits:
                    raise NameError('Please enter a valid name')
            if name == 'done':
                break
            marks = int(input('Enter the marks: '))
            dict1[name] = dict1.get(marks, 0) + marks
        except (NameError, ValueError) as e:
            print(f'Error: {e}')

    return dict1

print(get_marks())
```

Day 44

There are two functions in this code. The first one appends or writes the email onto the file. We open the file (emails.csv) in **append** mode "a" so we do not overwrite content that is already on the file. If you open a file in **append** mode, if the file does not exist, it will be created. We use the escape character (**\n**) to ensure that every email is appended to a new line. Since we are using the with statement, the file will be automatically closed. The **second** function opens and reads the file. Since we are just reading the file, we open the file in read mode ("r").

```
In [88]: def save_emails():
        w = []
        while True:
            email = input("Enter your email or done when finished: ")
            w.append(email)
            if email == 'done':
                break
            with open('emails.csv', 'a') as f:
                f.write(email)
                f.write('\n')

        #save_emails()

        # Second function to read emails
        def open_emails():
            with open('emails.csv', 'r') as f:
                return f.read()

        print(open_emails())
```

Day 45

We are using the string module to generate all the elements that we are looking for in the string. We have created variables for special characters (**special_cha**), numeric characters (**numeric_cha**), and the d variable for our dictionary. The **replace()** method is used to get rid of all the whitespace between string elements.

```
In [89]: import string

def analyse_string(s):
    special_cha = []
    numeric_cha = []
    d = {
        'numeric': '',
        'special_cha': '',
        'total_char': ''
    }

    for i in s.replace(' ', ''):
        if i in string.punctuation:
            if i not in special_cha:
                special_cha.append(i)
            d['special_cha'] = len(special_cha)
        if i in string.digits:
            if i not in numeric_cha:
                numeric_cha.append(i)
            d['numeric'] = len(numeric_cha)
        else:
            d['total_char'] = len(s.replace(' ', ''))
    return d

print(analyse_string('Python has a string format operator %. '
    'This functions analogously to '
    'printf format strings in C, e.g.'
    '"spam=%s eggs=%d" % ("blah", 2) '
    'evaluates to "spam=blah eggs=2?'))

{'numeric': 1, 'special_cha': 8, 'total_char': 140}
```


Day 46

The first thing that we did was **import pandas**. Then we put the information in a dictionary. The **pd.DataFrame** creates a two-dimensional table. This is one way you can create a DataFrame using pandas. You may have figured out another way to create it.

```
In [90]: import pandas as pd

# Creating a dictionary
table = {'year': [2009, 2002, 2009, 2010, 2009],|
        'title': ["Brothers", "Spider-Man", "WatchMen", "Inception", "Avatar"],
        'genre': ["Drama", "Sci-fi", "Drama", "Sci-fi", "Fantasy"]}

movies = pd.DataFrame(table)
movies
```

Out[90]:

	year	title	genre
0	2009	Brothers	Drama
1	2002	Spider-Man	Sci-fi
2	2009	WatchMen	Drama
3	2010	Inception	Sci-fi
4	2009	Avatar	Fantasy

Extra Challenge: Website Data with Pandas

The first thing is to import pandas. Since we are reading from a website, we need to use the **read_html** method. The data obtained from the website is in the form of a list, so we use slicing [1] to get the part or table that we want.

Out[91]:

	Type	Mutability	Description	Syntax examples
0	bool	immutable	Boolean value	True False
1	bytearray	mutable	Sequence of bytes	bytearray(b'Some ASCII') bytearray(b'Some ASCII...')
2	bytes	immutable	Sequence of bytes	b'Some ASCII' b'Some ASCII' bytes([119, 105, 104])
3	complex	immutable	Complex number with real and imaginary parts	3+2.7j 3 + 2.7j
4	dict	mutable	Associative array (or dictionary) of key and value	{'key1': 1.0, 3: False} {}

```
In [92]: # getting the two columns from table.
data_types = table[['Type', 'Mutability']]

print(data_types)
```

	Type	Mutability
0	bool	immutable
1	bytearray	mutable
2	bytes	immutable
3	complex	immutable
4	dict	mutable
5	types.EllipsisType	immutable
6	float	immutable
7	frozenset	immutable
8	int	immutable
9	list	mutable
10	types.NoneType	immutable
11	types.NotImplementedType	immutable
12	range	immutable
13	set	mutable
14	str	immutable
15	tuple	immutable

Day 47

For this challenge, we import the JSON module. This code demonstrates how to save a dictionary as JSON data in a file using **json.dump()**, and how to read the JSON data from the file and load it back into a dictionary using **json.load()**. We write two functions, one to save the file (**save_json**) and one to read the file (**read_json**). Here are the two functions below:

```
In [93]: import json

names = {'name': 'Carol', 'sex': 'female', 'age': 55}

def save_json(dict1):
    with open('file.json', 'w') as my_file:
        #saving to file and adding indent
        json.dump(dict1, my_file, indent=4)

save_json(names)

# Second function

def read_json():
    with open('file.json', 'r') as my_file:
        json_file = json.load(my_file)
    return json_file

print(read_json())

{'name': 'Carol', 'sex': 'female', 'age': 55}
```

Day 48

The first step to solving a binary problem is to ensure that the list is sorted. A binary search does not work on a list that is not sorted. For this challenge, the first thing you had to do was sort the array or list. At its most basic, the binary search looks for the middle element in a list. If the list's middle element is not what you are searching for, it will check whether the element you are searching for is greater than or less than the middle element. If it is **greater**, then it will search for the **middle** element of the upper half of the list; if it is less, then it will search for the middle element of the lower part of the list. This process is repeated until the element is found. So, it picks a portion of the list and searches for the middle element. We use the iterative method for this challenge.

```
In [94]: def search_binary(list1, x):
        low = 0
        high = len(list1) - 1
        mid_index = 0

        while low <= high:
            mid_index = (high + low) // 2

            if list1[mid_index] == x:
                return mid_index
            # if an element is lower than the middle index
            # then search the lower part of the list
            elif list1[mid_index] > x:
                high = mid_index - 1
            # if an element is higher than the middle index
            # then search the upper part of the list
            elif list1[mid_index] < x:
                low = mid_index + 1
        # if an element is not found, return -1
        return 'no element'

list1 = [12, 34, 56, 78, 98, 22, 45, 13]
list1 = sorted(list1)
number = 22
results = search_binary(list1, number)
if results == 'no element':
    print('Element is not in the list')
else:
    print(f'The element index is {results}')
```

The element index is 2

Day 49

In this challenge, we are saving the movies to the database. The first step is to import SQLite and create a database (movies.db). Then we create a table called "movies" with all the columns that we need (year, title, and genre). There are two ways to upload data to our database. The first one is using the command **cur.execute**. This method only uploads one item at a time. Since we want to upload many items at once, we use **cur.executemany**. To accomplish this, we make a list of all the movies we want to upload (movies variable). Once we upload, we need to commit and close to make those changes permanent.

```
In [95]: import sqlite3

con = sqlite3.connect('my_movies.db')
cur = con.cursor()

# Creating a table in the database
try:
    cur.execute('CREATE TABLE movies_table(year,title, genre)')
except sqlite3.OperationalError:
    print("Table already exists")

# Creating a variable of all the movies
movies = [(2009, 'Brothers', 'Drama'),
          (2002, 'Spider-Man', 'Sci-fi'),
          (2009, 'WatchMen', 'Drama'),
          (2010, 'Inception', 'Sci-fi'),
          (2009, 'Avatar', 'Fantasy')]

cur.executemany('INSERT INTO movies_table VALUES(?, ?, ?);',
               movies)

# commit and close
con.commit()
con.close()
```

- a. Creating a query to see all the movies in the table. First, we connect back to the database, and then we run the query below:

```
In [96]: con = sqlite3.connect('my_movies.db')
cur = con.cursor()

# Select all from table
table = cur.execute('SELECT * FROM movies_table;')

for movie in table:
    print(movie)

(2009, 'Brothers', 'Drama')
(2002, 'Spider-Man', 'Sci-fi')
(2009, 'WatchMen', 'Drama')
(2010, 'Inception', 'Sci-fi')
(2009, 'Avatar', 'Fantasy')
```

- b. To see the movie "Brothers" we add **WHERE** to our SQL query. WHERE is a conditional statement.

```
In [97]: row_brothers = cur.execute('SELECT * FROM Movies_table WHERE '
                                     'title ="Brothers";')
for movie in row_brothers:
    print(movie)

(2009, 'Brothers', 'Drama')
```

- c. For this query, we want to see movies released in **2009**. Here is the SQL query to make it happen.

```
In [98]: year_2009 = cur.execute('SELECT * FROM Movies_table WHERE '
                                   'year = 2009')
for year in year_2009:
    print(year)

(2009, 'Brothers', 'Drama')
(2009, 'WatchMen', 'Drama')
(2009, 'Avatar', 'Fantasy')
```

- d. This question requires that we put multiple conditions in our query. We need movies in the "drama" and "fantasy" genres. Here is the query below:

```
In [99]: for genre in cur.execute('SELECT * FROM Movies_table WHERE '
                                     'genre = "Fantasy" OR genre = "Drama";'):
           print(genre)

(2009, 'Brothers', 'Drama')
(2009, 'WatchMen', 'Drama')
(2009, 'Avatar', 'Fantasy')
```

- e. To delete the contents of our table we can run this query:

```
In [100]: # Delete movies from table
cur.execute('DELETE FROM movies_table;')
con.commit()
```

Day 50

The first part of the challenge is to create the backend end of the app. The backend of the app is powered by Python. With just a few lines of code, Flask will have the app running. When you install **Flask**, you will have to import it into your file. You will have to import **Flask** and **render_templates**. The **render_templates** file is used to render or generate our HTML templates. Below is the script that runs the app. This file is saved as app.py.

```
from flask import Flask , render_template

# instantiating flask app
app = Flask (__name__)

@app.route ( "/" )
def home():
    return render_template ( 'home.html' )

@app.route ( "/about" )
def about():
    return render_template ( 'about.html' )

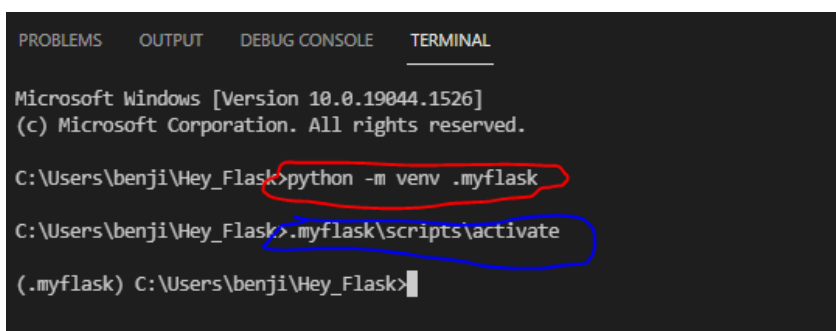
if __name__ == "__main__" :
    app.run(debug=True)
```


Let me give a brief overview of what is going on with this project. Even though I have faith that you have completed this project on your own using your powerful research skills.

When working on projects, it is recommended to create a **virtual environment**. A virtual environment is an environment that holds all the dependencies for our project. Instead of installing dependencies such as Flask on the system, you can install them in the virtual environment that you have created. For this project, I used **Visual Studio Code**. I will give you steps that you can use to create a virtual environment easily in Visual Studio. These commands are for Windows. Look up other commands if you are not on Windows.

1. The first step is to go into your **c** drive and create a folder that you will use for the project.
2. The second step is to open VSC. Go to File and select **Open Folder** on the drop-down menu.
3. Now open the terminal in VSC (click on terminal and select **New terminal**) and type the following to create a virtual environment (highlighted in red in the picture below)

python -m venv myflask



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
Microsoft Windows [Version 10.0.19044.1526]
(c) Microsoft Corporation. All rights reserved.

C:\Users\benji\Hey_Flask>python -m venv .myflask
C:\Users\benji\Hey_Flask>.myflask\scripts\activate
(.myflask) C:\Users\benji\Hey_Flask>
```

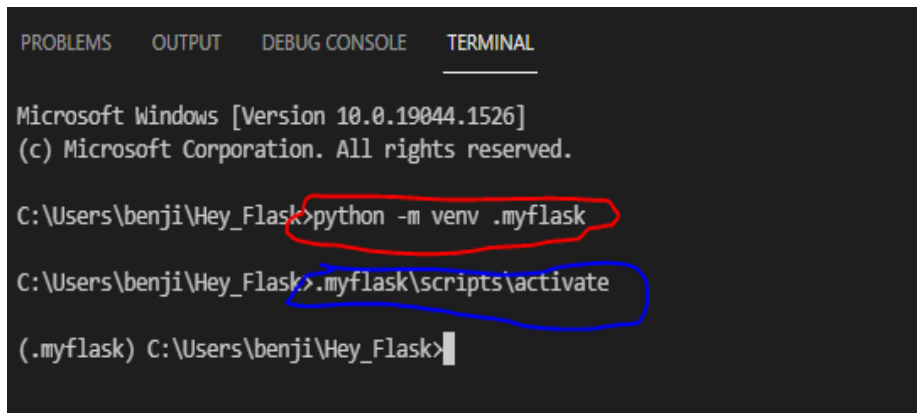
The last part **.myflask** is the name of the environment, so you can name it whatever you want. On the left side of your screen, Just

below the name of your folder, a new folder name (the name of your environment) will be added.

4. The fourth step is to activate your virtual environment. Type the following in your terminal.

Type:

.myflask\scripts\activate (highlighted in blue below). Do you see **(.myflask)** at the beginning of the next line after the one highlighted in blue? If you see the name of your virtual environment there, it means your virtual environment has been activated.

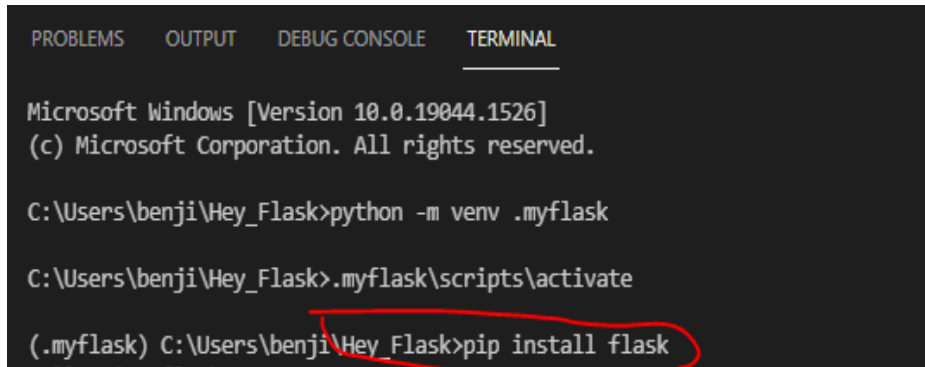


```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Microsoft Windows [Version 10.0.19044.1526]
(c) Microsoft Corporation. All rights reserved.

C:\Users\benji\Hey_Flask>python -m venv .myflask
C:\Users\benji\Hey_Flask>.myflask\scripts\activate
(myflask) C:\Users\benji\Hey_Flask>
```

5. Now that the virtual environment has been activated, we can add flask to our project. Type "pip install flask" in the terminal. See the example below.



The screenshot shows a terminal window with the following text:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Microsoft Windows [Version 10.0.19044.1526]
(c) Microsoft Corporation. All rights reserved.

C:\Users\benji\Hey_Flask>python -m venv .myflask

C:\Users\benji\Hey_Flask>.myflask\scripts\activate

(.myflask) C:\Users\benji\Hey_Flask>pip install flask
```

A red circle highlights the command `pip install flask` in the terminal.

6. The next step is to activate the interpreter for the project. Type Ctrl + Shift + P, and it will open a command palette. Click on "Python select interpreter" and select the recommended Python interpreter for your project.
7. Now create a Python file for your project called **app.py**. On the top left of VSC, you will see the name of the folder you created for the project. When you hover your mouse over the folder, on the right, you will see the option to create a folder or file. Select a new file and name it app.py. This is the file with the Python code above.

8. When you hover over your folder on the left side of the VSC code again, select **Create a folder** and name your folder templates. This is where you will save your **HTML file**. Create another folder called "static." This is where you will save your **CSS ("styles.css")** and picture files for the project.

Below are the **HTML** and **CSS** codes for the project.

home.html

```
{%extends "index.html"%}
{%block content%}
<html>
<head>
  <link rel = "stylesheet" href = "{{url_for('static',filename = 'styles.css')}}" >>
</head>
<body>

  <div class = 'container' >
    <h1 class = " >Home Page /h1>
    <p>
    <h2 class ='container_2' >This is the home page. I made this with Python,
      <br> HTML and CSS /h2>
    </p>
  </div >
```

about.html

```
{%extends "index.html"%}
{%block content%}
<html>
<head>
  <link rel = "stylesheet" href = "{{url_for('static',filename='styles.css')}}" >>
</head>
<body>

  <div class = container >

    <h1 class = "" > About Page</h1>

    <p style = "text - align: center;" >Lorem ipsum dolor si t amet,
    consectetur <br>
    adipiscing elit,
    sed do eiusmod tempor incididunt ut labore et
    dolore magna aliqua <br>.
    Ut enim ad minim veniam, quis nostrud
    exercitation ullamco laboris <br>
    nisi ut aliquip ex ea com modo consequat.
    Duis aute irure dolor in reprehenderit in <br>
    voluptate velit esse
    cillum dolore eu fugiat nulla pariatur.
    Excepteur sint occaecat <br>
    cupidatat non proident,
    sunt in culpa qui officia deserunt
    mollit anim id est laborum. </p>

  </div >

</body>
</html>
{%endblock%}
</html>
{%endblock%}
```

index.html

```
<!DOCTYPEhtml>
<html lang ="en" >
<head>
    <meta charset ="UTF- 8">
    <title >Title </title >
    <link rel ="stylesheet"
href ="{{url_f or('static',filename='styles.css')}}">>
</head>
<body>
    <head>

        <div class = 'navbar' >
            <strong><a href ="{{url_for ('home')}}"> Home/a></strong>
            <strong><a href ="{{url_for ('about')}}"> About/a></strong>

        </div >

    </head>

</body>
{% block content %}
{%endblock%}
</html>
```

styles.css

```
body{
    background-size : cover;
    background-image: url ( pix.jpg );
    justify-content: center;
    padding: 0%
    margin: 0%
}

.container {text-align : center;
    font-size : 200%
    margin-top: 10%
    color : whitesmoke
    margin-bottom 30%
text-shadow 5px 5px 10px black;}

.container h2 {text-align : center;
    font-size : 150%
    margin-top: 0%
    color : rgb(226 230 23);
    margin-bottom 50%
text-shadow 5px 5px 10px black;}

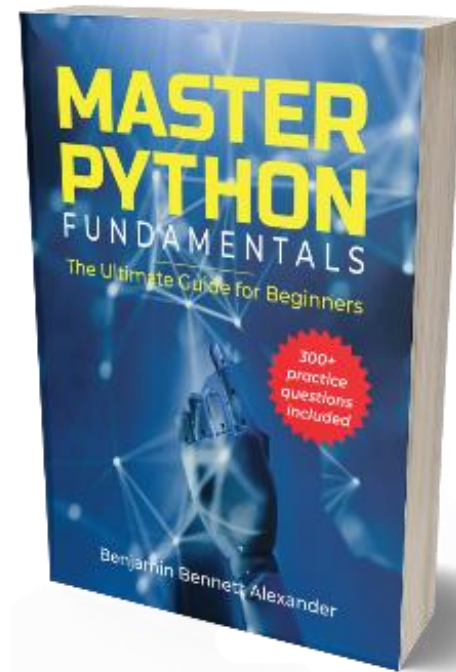
.navbar {
    overflow: hidden;
    background-color : rgb(22 18 20);
    position : fixed ;
    width: 100%
    top: 0;
}

.navbar a {
    float : right ;
    display : block;
    text-align : center;
    color : coral ;
    padding: 14px 16px
    text-decoration : none}
```


The best that you can do is get this code running and try to understand how everything is working together. Don't be afraid to take it apart and add new features. That is how you learn.

Other Books by Author

Master Python Fundamentals: The Ultimate Guide for Beginners.



Amazon link: <https://www.amazon.com/dp/BoBKVCM6DR>

End