50 DAYS OF PYTHON

A Challenge a Day.

Benjamin Bennett Alexander

50 DAYS PYTHON

A challenge a Day

Benjamin Bennett Alexander

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

Introduction

Challenges in general are the building blocks to a strong character. What does not kill you makes you stronger, they say. Apply the same principle to your Python learning journey and you will be unstoppable. Challenges are what this book is about. Your ability to solve them is what you are about. This book presents you with an opportunity to master some of the most important concepts and fundamentals of the Python language by solving challenges. There are more than 50 challenges in this book and you are required to solve them in 50 days. Solve one challenge a day and go on about your business and by the end of 50 days, you will have mastered Python fundamentals.

Why Solve challenges?

A programmer's job is to write code to solve problems. When you solve a challenge, not only do you get to cement your knowledge, but you master the invaluable skill of finding solutions to problems. Solving challenges helps you think. By solving challenges, you get to see how your code can solve real-life situations that programmers face.

How to Use this Book?

To get the best out of this book, you have to try your level best to solve the problem on your own. Do not rush to look up the answer. Remember, you have the whole day to solve one problem. Some problems may be easy to solve depending on your level of experience. Most of the problems in this book are for people that are beginning (beginners and intermediate) their Python journey. Remember that, in most instances, there is more than one way to solve a problem. Your solution and the solution in the book may differ.

In this book, you will find challenges that cover most of the fundamentals of Python. You will create passwords generators, work with Python modules, translate strings, and build dictionaries. You will work with CSV and JSON files. You will create a website and so much more. Trust me, you will be busy. Without further ado, let's get started.

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, returns 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**.**

Day 2: Strings to Integers

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

Day 3: Register Check

Write a function called ***register_check*** that checks how many students are in school. The function takes a dictionary as a parameter. 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. Use the dictionary below. Your function should return 3.

```
register = {'Michael':'yes','John': 'no',
            'Peter':'yes', 'Mary': 'yes'}
```

Day 4: Only Floats

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

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 price and **15%** as the discount, your function should return **127.5**.

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.

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’**.

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**.

Day 9: Biggest Odd Number

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

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: Strings With a Thousand Separator

b. Your new company has a list of figures saved in a list. The issue is that these numbers have **no separator**. The numbers are saved 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 on 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, a 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**.

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

b. 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 to 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 less than 4 digits long should render input invalid. Notify the user to input a four digits number.
   b. If a user enters a year **before** 1900, your code should tell the user that 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 takes no parameter. 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, VAT is 15% your code should return a vat inclusive price of 253. Make sure that your code can handle **ValueError**. Ensure the code runs until valid numbers are entered. (hint: Your code should include a while loop).

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-dimension list. For example **[[2,4,5,6]]** should return **[2,4,5,6]**.

Extra Challenge: Teacher's Salary

b. 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 monthly gross 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**.**

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**.

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 – 9 at the end of the name. The function should return the **username**.

Day 18: Any Number of Arguments

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

Day 19: Words and Elements

Write two functions. The first function is called ***count_words*** which takes a string of words 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’**.

Day 21: List of Tuples

Write a function called ***make_tuples*** that takes two lists, equal lists, and combines them into a list of tuples. 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)]**.

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 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.**

Day 24: Average Calories

Create a function called **average_calories** that calculates the average calories intake of a user. The function should ask the user to input their calories intake for **any** number of days and once they hit ‘**done**’ it should calculate and **return** the average intake.

Day 25: All the Same

Create 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. 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**.

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']**.

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 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 unique numbers only**. For example **[1, 2, 4, 5, 6, 7, 8, 8]** should return **[1, 2, 4, 5, 6, 7, 8, 8].**

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 lower letters in the string. For example **‘LovE’** should return **[1,2].**

Day 29: Middle Figure

Write a function called ***middle_figure*** that takes 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, 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.

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

Extra Challenge: Sort by Last Name

b. You work for a local school in your area. The school has a list of names of students saved in a list. 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 names. Here is a list of names:

   [‘Beyonce Knowles, ‘Alicia Keys’, ‘Katie Perry’, ‘Chris Brown’,’ Tom Cruise’]

   Your code should not just sort them alphabetically, but 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 Beyonce']

   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.

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 upper letter**, **one lower 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 tell the users 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. (use while loop).

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. Use list comprehension in your solution. Use the lists below. Your function should return **(30, 65, 80)**.
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 a 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(10000000)
x = set(a)
y = list(a)

Let's say you are looking for a number 9999999 in the range above. Search for this number in the **list** and the **set**. Your challenge is to find which code executes faster. You will pick the one that executes quicker, lists, or 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, return **False**. The following sentence is a pangram so it should return **True**:

'the quick brown fox jumps over a lazy dog'

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}.

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. For example '**hello**' should return **2** vowels. If a vowel appears in a string more than once it should be **counted as one**. For example, 'saas' should return 1 vowel. 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 low, the code should tell them that their guess is too low. The user should get a maximum of three guesses. When the user guesses right, the code should declare them a winner. After three wrong guesses, the code should declare them a loser.

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 upper letters, lower 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,** and **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.

```
a = '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 words that have vowels in them. For example **' You have no rhythm'** should return **['You','have', 'no'].**

Extra Challenge: Class of Cars

b. Create three classes of three car brands – **Ford, BMW,** and **Tesla**. The **attributes** of the car's objects will be, **model, color, year, transmission,** and **whether the car is electric or not (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, **Colo**r: beige, **Year:** 2017, **Transmission**: Auto, **Electric:** True
ford1 : **model**: focus, **Color**: white, **Year:** 2020, **Transmission**: Auto, **Electric**: False

You will create a **class 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 a user inputs a 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**.

Day 43: Student Marks

Write a function called ***student_marks*** that records 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 marks are the **value**. When the user enters done, the function should return a **dictionary of names and values entered.**

Day 44: Save Emails

Create a function called **save_emails**. This function takes no arguments but asks the user to input 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 must be in its line. Here is an example of how the emails must be saved:
jj@gmail.com
kate@yahoo.com

and not like this :
jj@gmail.comkate@yahoo.com

Day 45: Words and Special Characters

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

Day 47: Save a JSON

Write a function called **save_json**. This function takes a dictionary below 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.

```
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:

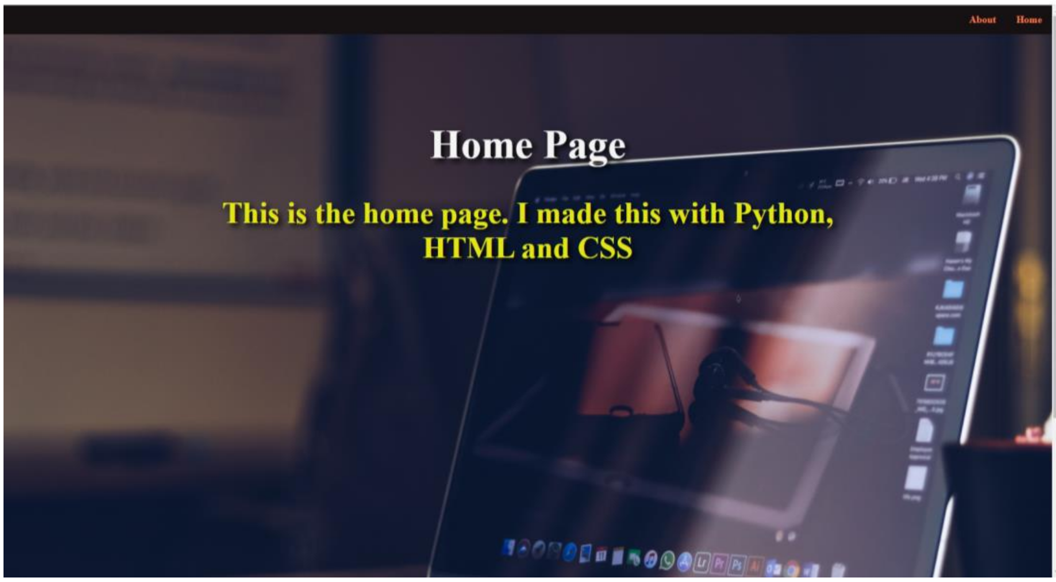
year	**title**	**genre**
2009	Brothers	Drama
2002	Spider Man	Sci-fi
2009	WatchMen	Drama
2010	Inception	Sci-fi
2009	Avatar	Fantasy

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

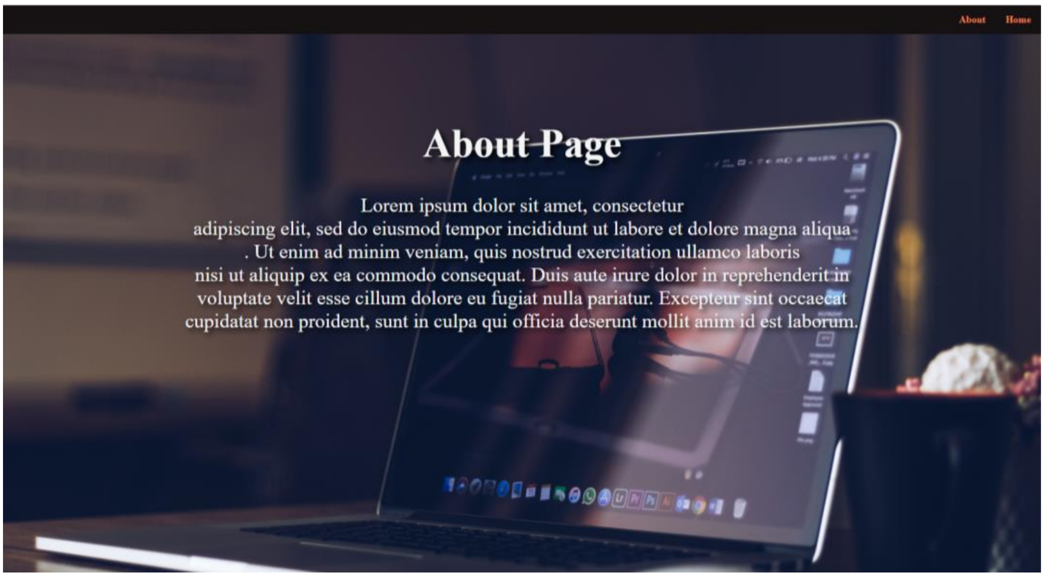
Day 50: Create a Flask App

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 code editor 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 from [Hasan Albari] **from** [Pexels].
Below is what you should make:

Home Page


About
Home
Home Page
This is the home page. I made this with Python, HTML and CSS

About Page



About Page

Lorem ipsum dolor sit amet, consectetur
adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua
. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris
nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in
voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat
cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

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 not made any 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

Day 1

In this challenge, we use **conditional statements** to create two conditions. If any of those conditions evaluate to **True**, that block of code will be executed. The **if statement** checks if the number is divisible by 5, 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.

```python
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}'
    else:
        remainder = number % 5
        return f'The remainder of the division is {remainder}'


print(divide_or_square(10))
```

Output:

```
The square-root of the number is 3.16
```

Day 2

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

```
def convert_add(list1):
    b = []
    for i in list1:
        b.append(int(i))
    return sum(b)


print(convert_add(['1','3','5']))
```

Output:

```
('The sum is ', 9)
```

If you do not want to use the in-built **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.***

```
def convert_add(list1):
    list2 = []
    count = 0
    for i in list1:
        list2.append(int(i))
    for j in list2:
        count += j
    return b


print(convert_add(['1','3','5']))
```

Day 3

To count the yes in the dictionary, we need to access the dictionary values**(a.values)** using the **for loop**. The for loop iterates through the dictionary in search of yes values. Every value is added to the count variable.

```
register = {'Michael':'yes', 'John': 'no',
            'Peter':'yes',  'Mary': 'yes'}


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


print(register_check(register))
```

Output:

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

Another way to go about it:

```
register = {'Michael':'yes', 'John': 'no',
            'Peter':'yes',  'Mary': 'yes'}

def register_check(reg):
    count = 0
    for value, key in a.items():
        if value == 'yes':
            count += 1
    return 'Number of students in school is', count


print(register_check(register))
```

Day 4

In this challenge, we use the **type()** function and conditional statements. The **type()** function is used to get the type of our arguments**.** We also use the **and** and **or** operators to compare the two arguments.

```python
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))
```

Output:

```
1
```

Day 5

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

```
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())
```

Output:

```
('Price after discount is ', 127.5)
```

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, that is why we use **[0]** in our code to access it.

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


print(user_name())
```

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.

```python
def string_range(num):
    x = [str(i) for i in range(num)]
#Using join method to add dots
    x = ".".join(x)
    return x


print(string_range(6))
```

Output:

```
0.1.2.3.4.5
```

Day 8

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

```
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]))
```

Output:

```
5
```

Day 9

Using list-comprehension, we use a **for loop** and **if statement** inside to return a list of odd numbers. Here you get to use the **modulus operator** (%). The modulus operator returns the remainder of a division. If the remainder is zero, the operator returns a zero. 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.

```python
def biggest_odd(string1):
    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'))
```

Output:

```
The biggest odd number is 9
```

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 '*'.

```
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())
```

b. Strings With a thousand Separator.

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

```
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]))
```

Output:

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

Day 11

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

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


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

Output:

```
True
```

Day 12

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

```
def count_dots(word):
    m = word.split(".")
    return f'The string has {len(m)-1} dots'


print(count_dots("h.e.l.p."))
```

Output:

```
The string has 4 dots
```

b. Age in Minutes

Remember to import the datetime module for this challenge. You will need to use the while loop to keep the code running until a valid year is inputted.

```
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. 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.

```
def your_vat():
    while True:
        try:
            price = int(input("Enter the price of item: "))
            vat = int(input('Enter vat: '))
        except ValueError:
            print("Enter a valid number")
        else:
            total_price = price + \
                          (price * vat / 100 + 1) - 1
            return 'The price VAT inclusive is', total_price


print(your_vat())
```

Output:

```
('The price VAT inclusive is', 253.0)
```

Day 14

Notice how we use nested **for loops** in the code below? The first **for loop** access the outer list, and the inner **for loop** access the inner list. Once we access the elements of the inner list, we append them to **list1**. With this small code, we have flattened the list.

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


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

Output:

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

b. Your salary

```
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())
```

Day 15

This challenge tests your ability to use negative string **slicing**, **if statements**, and **operators** in Python. The [::-1] reverses the string and the == operator compares the original string to the reversed string.

```python
def same_in_reverse(a):
    if a == a[::-1]:
        return True
    else:
        return False


print(same_in_reverse('mad'))
```

Output:

```
False
```

Day 16

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

```
def sum_list(lst1):
    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]]))
```

Output:

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

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.

```python
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())
```

Day 18

In this exercise, you get to know about ***args**. You can use any word you want as a parameter as long as you use the * sign 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 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.

```
def any_num (*args):
    ave = sum(args)/len(args)
    return f'The average is {ave}'


print(any_num(12, 90))
```

Output:

```
The average is 51.0
```

In this function, we have added more arguments.

```
def any_num (*args):
    ave = sum(args)/len(args)
    return f'The average is {ave}'


print(any_num(12, 90, 12, 34))
```

Output:

```
The average is 37.0
```

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.

```
def count_words(a):
    words = []
    for i in a.split():
        words.append(i)
        print(words)
    return f'There are {len(words)} ' \
           f'words in the sentence'


print(count_words('I love learning'))
```

Output:

```
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 in the string before using the **len()** function.

```
def count_characters(a):
    a = a.replace(' ', '')
    return f'The string has ' \
           f'{len(a)} elements '

print(count_characters('I love learning'))
```

Output:

```
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 lower 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.

```
def capitalize(a):
    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 love python'))
```

Output:

```
I Love Python
```

Day 21

The easiest way to deal with this problem is to use the Python built-in **zip()** function. The function combines the two lists into pairs of tuples. We use the **list()** and the **tuple()** function to ensure that the tuples will be in a list.

```
def make_tuples(a,b):
    a = zip(a,b)
    return list(tuple(a))

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

Output:

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

Day 22

For this challenge you have to use the **string()** method, **join()** 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.

```python
def add_hash(a):
    return "#".join(a)


def add_underscore(a):
    return str(a).replace("#", "_")


def remove_underscore(a):
    return str(a).replace("_", "")

print(remove_underscore(add_underscore(add_hash('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** block 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.

```
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())
```

Day 24

This simple average calories 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.**

```
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())
```

Day 25

We are using the in-built **all()** function. The **all()** function will return **True** if all the elements of iterable are true.

```
def all_the_same(a):
    a = all(i == a[0] for i in a)
    return a

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

Output:

```
True
```

We can also use the built-in **enumerate()** function. The enumerate function will return an item and its index. We can use that to check if all the items in the string, tuple, or lists are equal to the item sitting at index 0.

```
def all_the_same(a):
    for i, item in enumerate(a):
        if item[i] == item[0]:
            return True
        else:
            return False


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

Output:

```
True
```

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 in the **list1** variable. We use the **not-in** the list1 operator to ensure that we do add duplicates to the list. We use the **join()** method to add commas, between the string elements.

```
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'))
```

Output:

```
['e,f,i,l,o,v']
```

Day 27

For this challenge, once we append the unique numbers to **list1**, we use the **sum()** function to sum the original **list(a)** and the list with unique numbers, **(list1)**. We use the modulus operator(%) to check if the difference between the two numbers, is an even number or an odd number.

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


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

Output:

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

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 idex list.

```
def index_position (a):
    idex = []
    for i, item in enumerate(a):
        if item.islower():
            idex.append(i)
    return idex


print(index_position('LovE'))
```

Output:

```
[1, 2]
```

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 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 on position or index 4 is the middle element. Only strings with **odd number** lengths will return a middle element.

```
def middle_figure(a,b):
    c = (a + b).replace(' ','')
    print(c)
    b = len(c)
    middle = b//2
    if len(c)% 2 == 1:
        return c[middle]
    else:
        return 'No middle figure'


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

Output:

```
('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 3 times.

```
from collections import Counter

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

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


print(frequent_name(name))
```

Output:

```
Peter
```

b. Extra Challenge

```
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))
```

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**.

```
def longest_word(a):
    b =[]
    for word in a:
        b.append([len(word), word])
    return max(b)


print(longest_word(['Java','Javascript','Python']))
```

Output:

```
[10, 'Javascript']
```

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.

```python
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())
```

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**,

```
def inter_section(a, b):
    inter_list = tuple([i for i in a if i in b])
    return inter_list


print(inter_section(list1, list2))
```

Output

```
(30, 65, 80)
```

c. Set vs list

To test the speed of the code, we are going to use the **timeit function** for both codes. We create a variable called **speed_test** that has the code. We then pass this variable to the timeit function and we set the number of execution to 3.

Set

```
import timeit

# Testing the speed of execution in a set
speed_test = '''
a = range(1000000)
b = set(a)
i = 999999

for i in b:
    print('')
'''
print(timeit.timeit(stmt=speed_test, number=3))
```

Output

```
42.5450625
```

List

```
# Testing the speed of execution in a list
speed_test = '''
a = range(1000000)
b = list(a)
i = 999999

for i in b:
    print('')
'''
print(timeit.timeit(stmt=speed_test, number=3))
```

Output:

```
42.667658
```

From the results, we can conclude that sets execute slightly faster than lists. What is your conclusion?

Day 34

When opening files it is good practice to open them with a **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 the read mode. We use the split() method to split the words in the text into strings. By default, the split method splits the text in the whitespaces. The **isdigit()** method checks which string elements are non-numeric. These non-numeric elements are appended to the list1 variable.

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


print(just_digits())
```

Output:

```
['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 use the **join()** method to join the string elements. We then compare the sorted sentence to the alphabet. If the sentence variable has all the elements in the alphabet variable, the code will return **True**.

```python
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'))
```

Output:

```
True
```

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.

```
def count(a):
    d = {}
    for i in range(len(a)):
        x = a[i]
        count = 0
        for j in range(i, len(a)):
            if a[j] == a[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'))
```

Output:

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

Day 37

For this challenge, we create a string of vowels. The code checks if the letters in the string, are in the vowels string. If the letter is in the string, then it is appended to the **vowels list**. We then use the **len()** function to count the number of elements in the vowels list.

```python
def count_vowels(a):
    vowels = []
    for letter in a:
        if letter in 'AEIOUaeiou':
            if letter not in vowels:
                vowels.append(letter)
    return len(vowels)


print(count_vowels('hello'))
```

Output:

```
2
```

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 when they run out of guesses**.** The user gets a maximum of three(3) guesses.

```
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())
```

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.

```python
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 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'**.

```
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'))
```

Output:

```
iyay ovelay ythonpay
```

Day 41

Notice that we are using the **nested for loop** to find the letters in the words. We iterate through 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.

```python
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'))
```

Output:

```
['You', 'have', 'no']
```

b. In this challenge, we create the **Cars class**. This class is the parent class to class of car brands. This is called **inheritance**. The child classes are inheriting the properties and methods of the parent class. The child classes have Cars as an argument.

```
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 class 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", False)
print(ford1.print_car())
tesla1 = Tesla("S", "Grey", 2016, "Manual", True)
print(tesla1.print_car())
bmw1 = BMW('X6', 'silver', 2018, 'Auto', False)
print(bmw1.print_car())
```

Day 42

The **textblob module** is being used to check the word from the user. 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.

```
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())
```

Day 43

In this code, we ask the user to **input** the name of the student and their marks in the test. 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()**, searches for the key, and then it adds value to that key.

```
def get_marks():
    d = {}
    while True:
        name = input('Enter the student name or done: ')
        if name == 'done':
            break
        marks = int(input('Enter the marks: '))
        d[name] = d.get(marks, 0) + marks


    return d


print(get_marks())
```

Day 44

There are two functions in this code. The first one appends or writes the email on 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 in the 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’).

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

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


save_emails()

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 (**specia_cha**) numeric characters(**numeric_cha**) and the **d** variable for our dictionary. The **replace()** method is to get rid of all the whitespaces between string elements.

```
import string


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

    for i in a.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(a.replace(' ', ''))
    return d


print(char('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?'))
```

Output:

```
{'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-dimension table. This is one way you can create a Dataframe using pandas. You may have figured out another way to create it.

```
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)

print(movies)
```

Output:

```
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
```

Day 47

For this challenge, we import the JSON module. Here are the two functions below:

```
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())
```

Output:

```
{'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 basic form, the binary search looks for the middle element in a list. If the list 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 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.

```python
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}')
```

Output:

```
The element index is 2
```

Day 49

Creating and 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 do this we create a list with all the movies that we want to upload (movies variable). Once we upload, we need to commit and close to make those changes permanent.

```
import sqlite3

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

# Creating a table in the database
cur.execute('CREATE TABLE movies(year,title, genre)')

# 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 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:

```
con = sqlite3.connect('movies.db')
cur = con.cursor()

row = cur.fetchall()
for row in cur.execute('SELECT * FROM Movies;'):
    print(row)
```

Output:

```
 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.

```
con = sqlite3.connect('movies.db')
cur = con.cursor()

row = cur.fetchall()
for row in cur.execute('SELECT * FROM Movies WHERE '
                       'title ="Brothers";'):
    print(row)
```

Output:

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

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

```
con = sqlite3.connect('movies.db')
cur = con.cursor()

row = cur.fetchall()
for row in cur.execute('SELECT * FROM Movies WHERE '
                       'year = 2009'):
    print(row)
```

Output:

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

d. This question requires that we put multiple conditions in our query. We need movies that are in the **'Drama'** and **'Fantasy'** genres. Here is the query below:

```
con = sqlite3.connect('movies.db')
cur = con.cursor()

row = cur.fetchall()
for row in cur.execute('SELECT * FROM Movies WHERE '
                       'genre = "Fantasy" OR genre = "Drama";'):
    print(row)
```

Output:

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

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

```
con = sqlite3.connect('movies.db')
cur = con.cursor()
cur.execute('DELETE FROM movies;')
con.commit()
```

Day 50

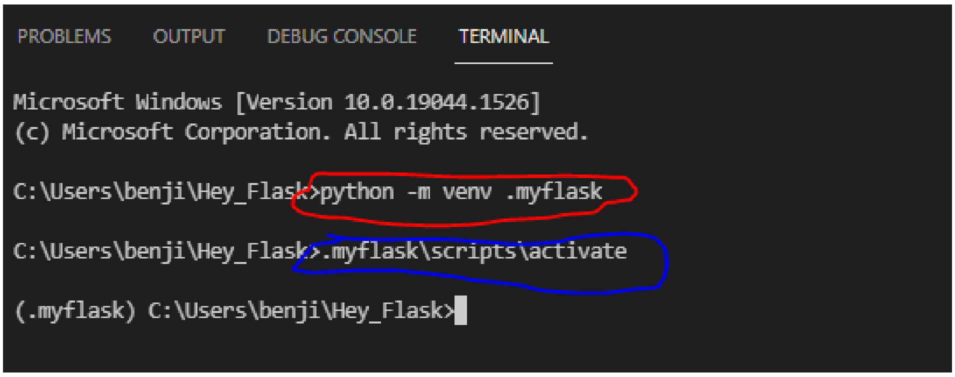
The first part of the challenge is to create the backend end of the app. The backend of the app is powered by Python. With Just a few lines of code, Flask will have the app running. When you install Flask you will have to import it into your file. You will have to import **Flask** and **render_templates**.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

```python
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 of our project. Instead of installing dependencies such as flask on the system, you can install it 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 the Visual studio. These commands are for windows. Look up other commands if you are not on windows.

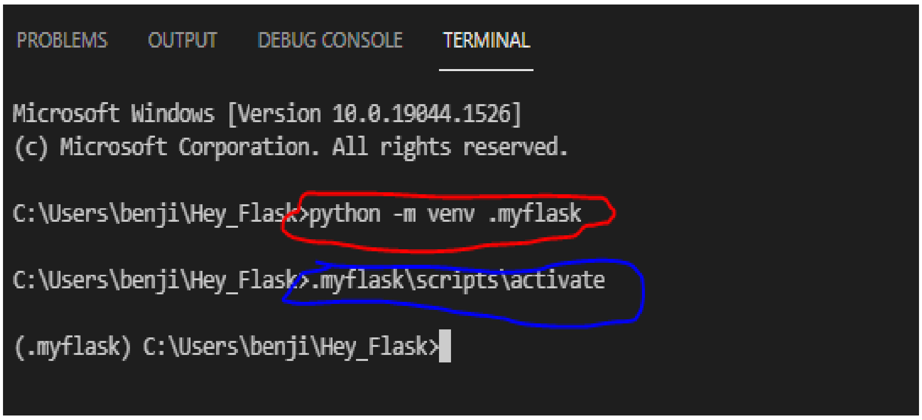
1. The first step is to go into your c drive and create a folder that you will use for the project.
2. The second step is to open VSC. Go to File and select **open folder** on the drop-down menu.
3. Now open the terminal in VSC (click on terminal and select **New terminal**) and type the following to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



```
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>
```

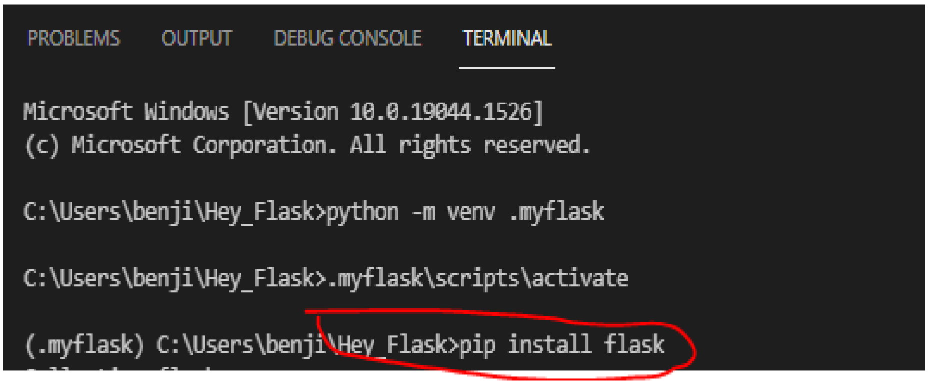
The last part **.myflask** is the name of the environment, so you can name it whatever you want**.** On the left side of your screen just below the name of your folder, a new folder name will be added(the name of your environment)

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.



```
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 is activated, it's time to install flask into our project. Type **pip install flask** in the terminal. See the example below.



```
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
```

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 on the folder, on the right you will see the option to create a folder or file. Select **new file** and name it app.py. This is the file with the Python code above.

8. When you hover on your folder on the left side of 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 sit 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 commodo 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

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <link rel="stylesheet" href="{{url_for('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 to 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.