30 DAYS OF PYTHON

Beginner to Advance Guide

By Skill Foundry

DAY-1 INTRODUCTION AND SETUP

Welcome to your journey into the world of **Python programming** with **Skill Foundry**. Over the next 30 days, you'll explore one of the most popular and beginner-friendly programming languages used across industries like web development, data science, automation, and artificial intelligence.

Python is a high-level, interpreted, and general-purpose programming language. Known for its simple and readable syntax, Python is widely used by both beginners and professionals. It was created by Guido van Rossum and first released in 1991 with the goal of making code more understandable and accessible. Since then, it has evolved into one of the most powerful tools in modern software development.

This course is structured to help you learn Python step by step — from the very basics to real-world applications. Each day introduces a focused topic along with clear explanations, practical examples, and hands-on exercises to help you apply what you learn immediately.

Whether you're starting from scratch or coming back to coding after a break, this course is designed to build your confidence and problem-solving skills with Python. You'll understand how to write clean code, think logically, and begin creating useful programs on your own.

No prior experience is required — just your curiosity and commitment to learning something new every day. By the end of this challenge, you'll not only understand how Python works but also have the foundation to explore more advanced areas like automation, scripting, or data analysis.

Let's begin.

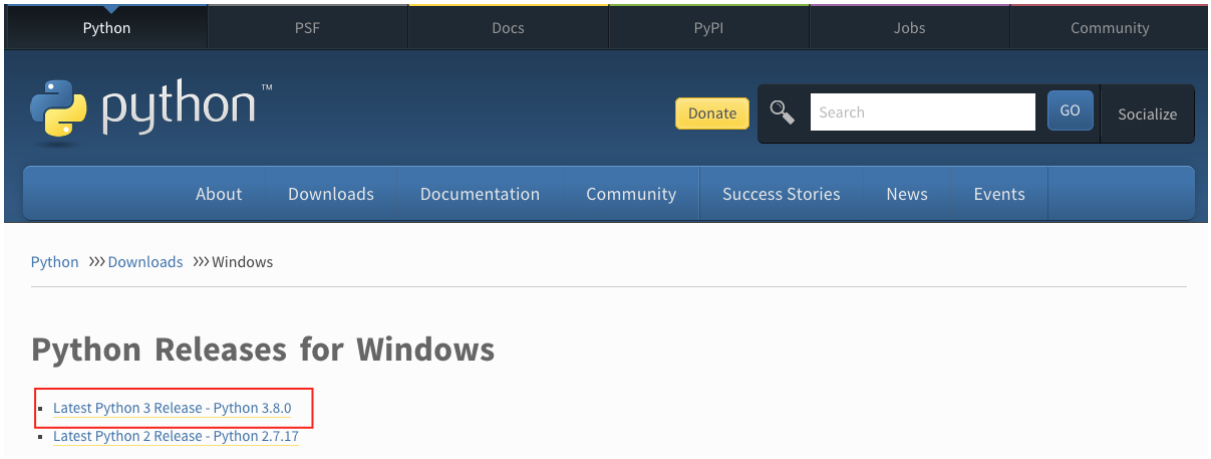
Why Python

It is a programming language which is very close to human language and because of that, it is easy to learn and use. Python is used by various industries and companies (including Google). It has been used to develop web applications, desktop applications, system administration, and machine learning libraries. Python is a highly embraced language in the data science and machine learning community. I hope this is enough to convince you to start learning Python. Python is eating the world and you are killing it before it eats you.

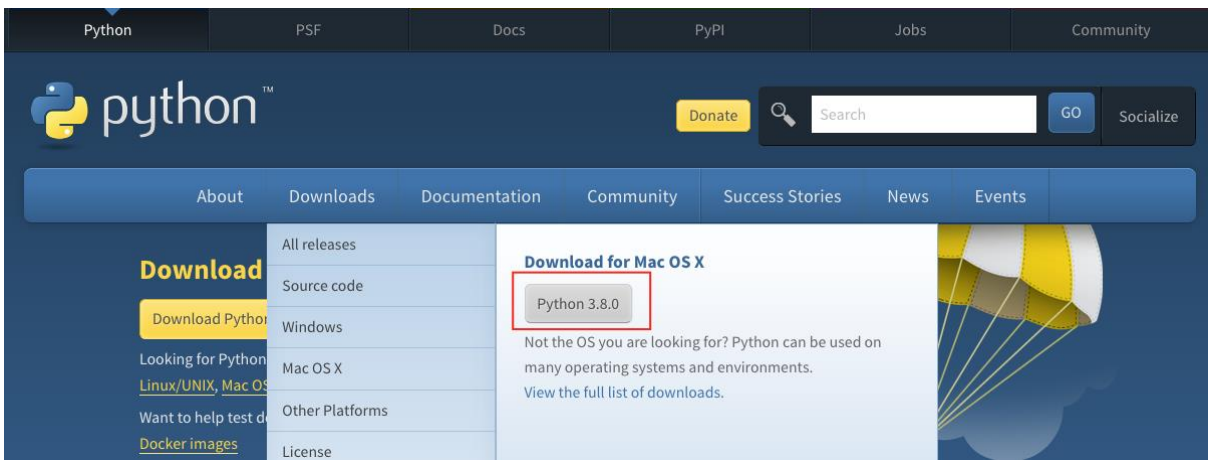
Environment Setup

Installing Python

To run a python script you need to install python. Let's download python. If your are a windows user. Click the button encircled in red.

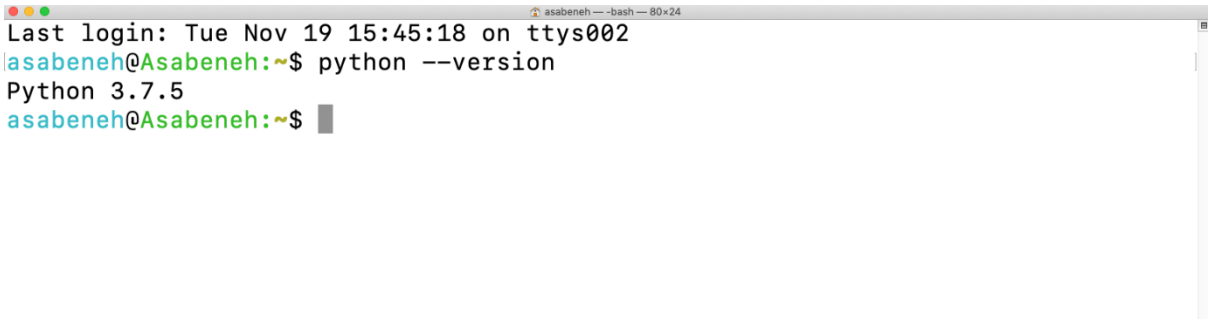


If you are a macOS user. Click the button encircled in red.



To check if python is installed write the following command on your device terminal.

```
python –version
```



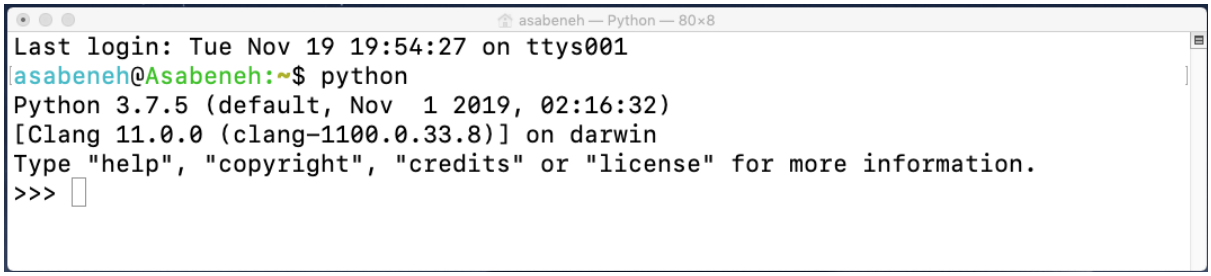
As you can see from the terminal, I am using *Python 3.7.5* version at the moment. Your version of Python might be different from mine by but it should be 3.6 or above. If you mange to see the python version, well done. Python has been installed on your machine. Continue to the next section.

Python Shell

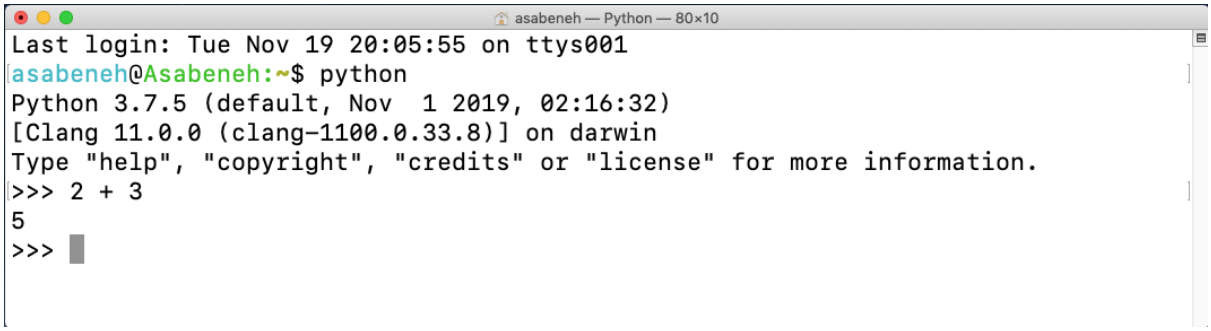
Python is an interpreted scripting language, so it does not need to be compiled. It means it executes the code line by line. Python comes with a *Python Shell (Python Interactive Shell)*. It is used to execute a single python command and get the result.

Python Shell waits for the Python code from the user. When you enter the code, it interprets the code and shows the result in the next line. Open your terminal or command prompt(cmd) and write:

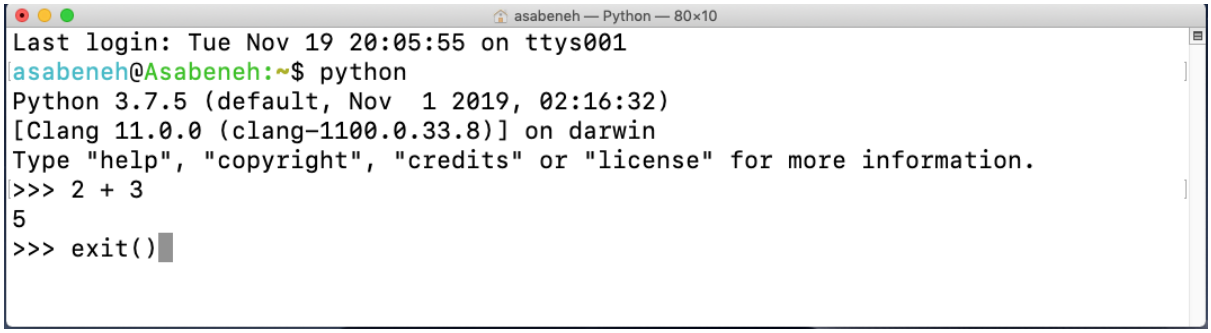
```
python
```



The Python interactive shell is opened and it is waiting for you to write Python code(Python script). You will write your Python script next to this symbol >>> and then click Enter. Let us write our very first script on the Python scripting shell.

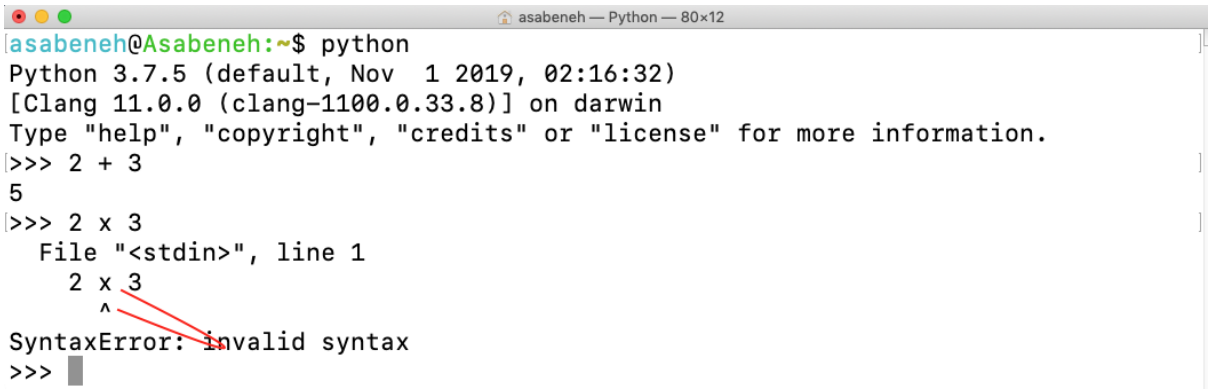


Well done, you wrote your first Python script on Python interactive shell. How do we close the Python interactive shell ? To close the shell, next to this symbol >> write **exit()** command and press Enter.



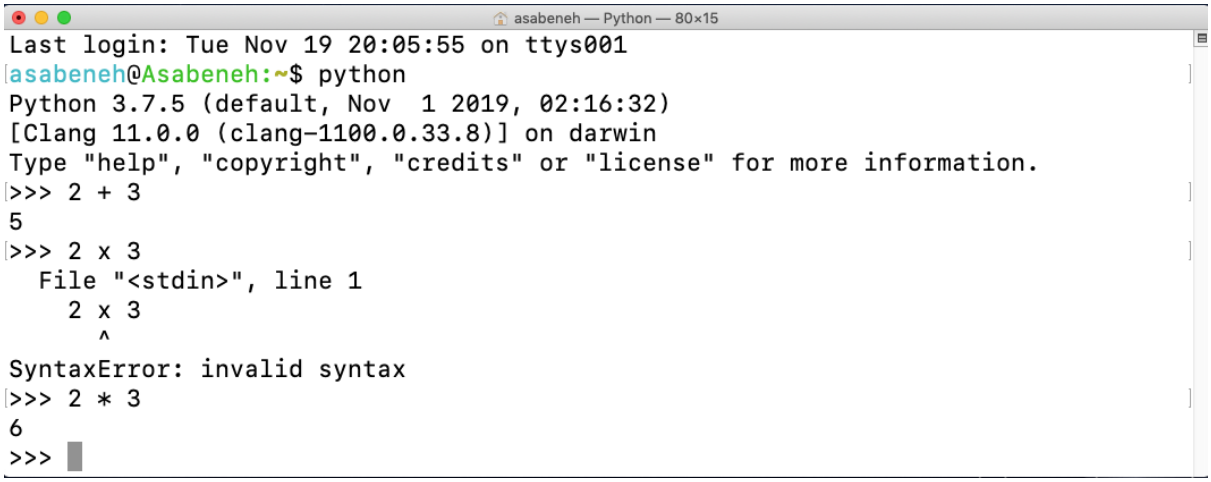
Now, you know how to open the Python interactive shell and how to exit from it.

Python will give you results if you write scripts that Python understands, if not it returns errors. Let's make a deliberate mistake and see what Python will return.



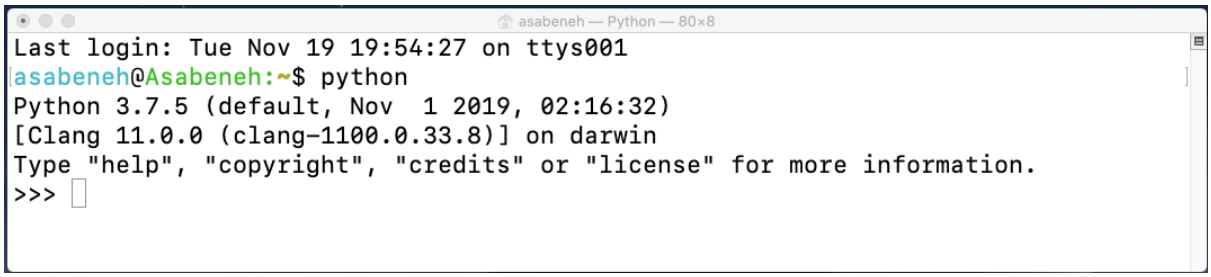
As you can see from the returned error, Python is so clever that it knows the mistake we made and which was *Syntax Error: invalid syntax*. Using x as multiplication in Python is a syntax error because (x) is not a valid syntax in Python. Instead of (**x**) we use asterisk (*) for multiplication. The returned error clearly shows what to fix.

The process of identifying and removing errors from a program is called *debugging*. Let us debug it by putting * in place of **x**.



Our bug was fixed, the code ran and we got a result we were expecting. As a programmer you will see such kind of errors on daily basis. It is good to know how to debug. To be good at debugging you should understand what kind of errors you are facing. Some of the Python errors you may encounter
are *SyntaxError*, *IndexError*, *NameError*, *ModuleNotFoundError*, *KeyError*, *ImportError*, *AttributeError*, *TypeError*, *ValueError*, *ZeroDivisionError* etc. We will see more about different Python ***error types*** in later sections.

Let us practice more how to use Python interactive shell. Go to your terminal or command prompt and write the word **python**.



The Python interactive shell is opened. Let us do some basic mathematical operations (addition, subtraction, multiplication, division, modulus, exponential).

Let us do some maths first before we write any Python code:

- 2 + 3 is 5
- 3 - 2 is 1
- 3 * 2 is 6
- 3 / 2 is 1.5
- 3 ** 2 is the same as 3 * 3

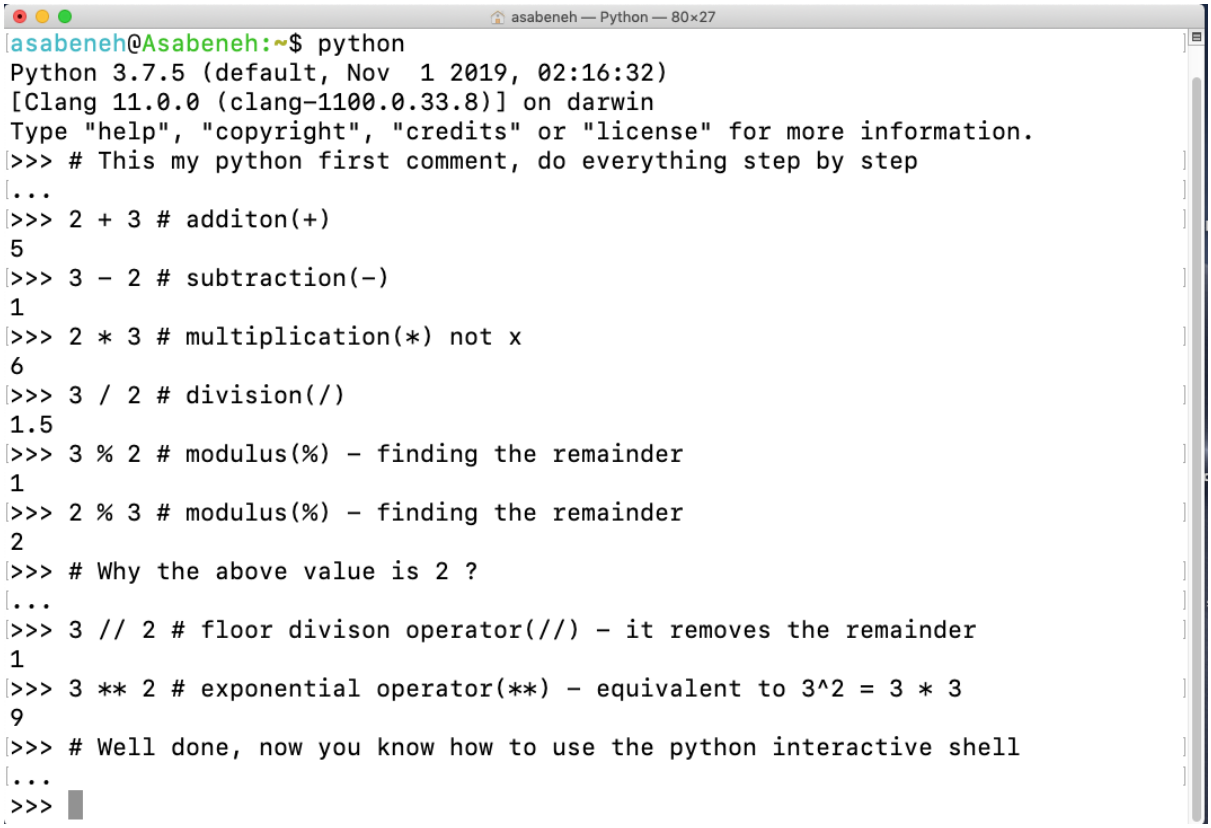
In python we have the following additional operations:

- 3 % 2 = 1 => which means finding the remainder
- 3 // 2 = 1 => which means removing the remainder

Let us change the above mathematical expressions to Python code. The Python shell has been opened and let us write a comment at the very beginning of the shell.

A *comment* is a part of the code which is not executed by python. So we can leave some text in our code to make our code more readable. Python does not run the comment part. A comment in python starts with hash(#) symbol. This is how you write a comment in python

```
 # comment starts with hash
 # this is a python comment, because it starts with a (#)
symbol
```

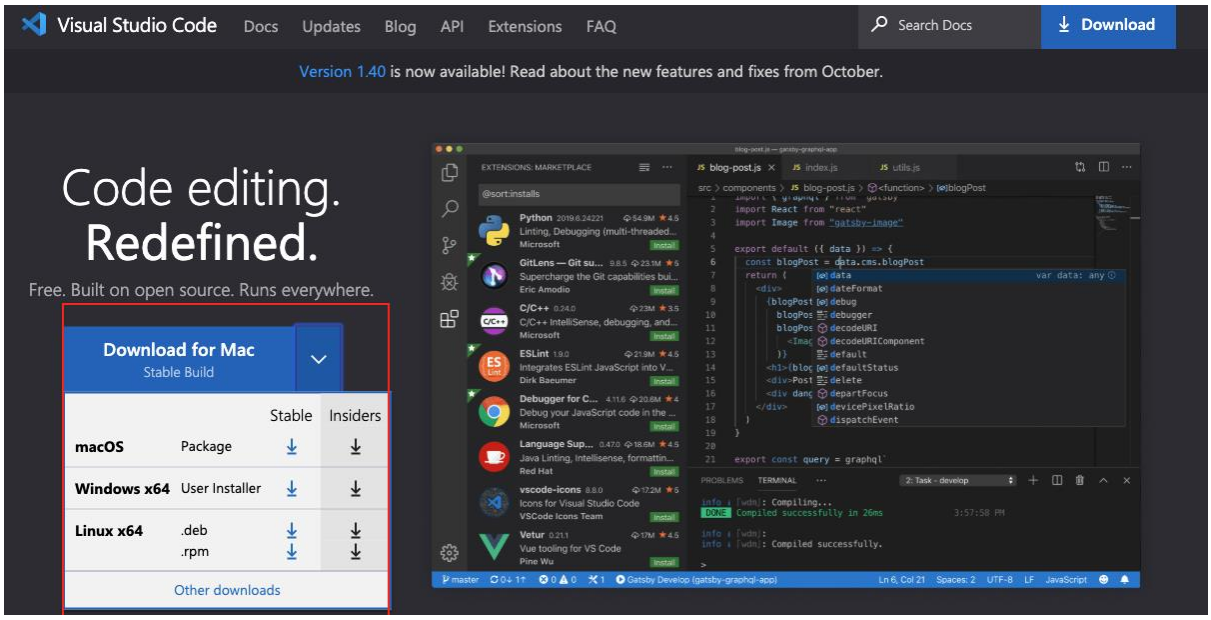


Before we move on to the next section, let us practice more on the Python interactive shell. Close the opened shell by writing *exit()* on the shell and open it again and let us practice how to write text on the Python shell.

```
asabeneh — -bash — 80×25
asabeneh@Asabeneh:~$ python
Python 3.7.5 (default, Nov  1 2019, 02:16:32)
[Clang 11.0.0 (clang-1100.0.33.8)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> # Let's practice how write text on python shell: we use single or double quo
te to make a string. Any text in a qute we call it string
...
>>> 'Asabeneh' # To write my name as string
'Asabeneh'
>>> "Asabeneh" # To write my name as string - now in double quote
'Asabeneh'
>>> # A string could be a single character text or a page
...
>>> 'I love teaching. And thank you so much for joining this course'
'I love teaching. And thank you so much for joining this course'
>>> # If the text is too long we use triple quote to make a string(''''''')
...
>>> '''We use triple quote if the text is more than one line'''
'We use triple quote if the text is more than one line'
>>> # Now you knew string and how to use the pythn shell
...
>>> # Lets continute to the next section and install text editor code editor
...
>>> exit()
asabeneh@Asabeneh:~$
```

Installing Visual Studio Code

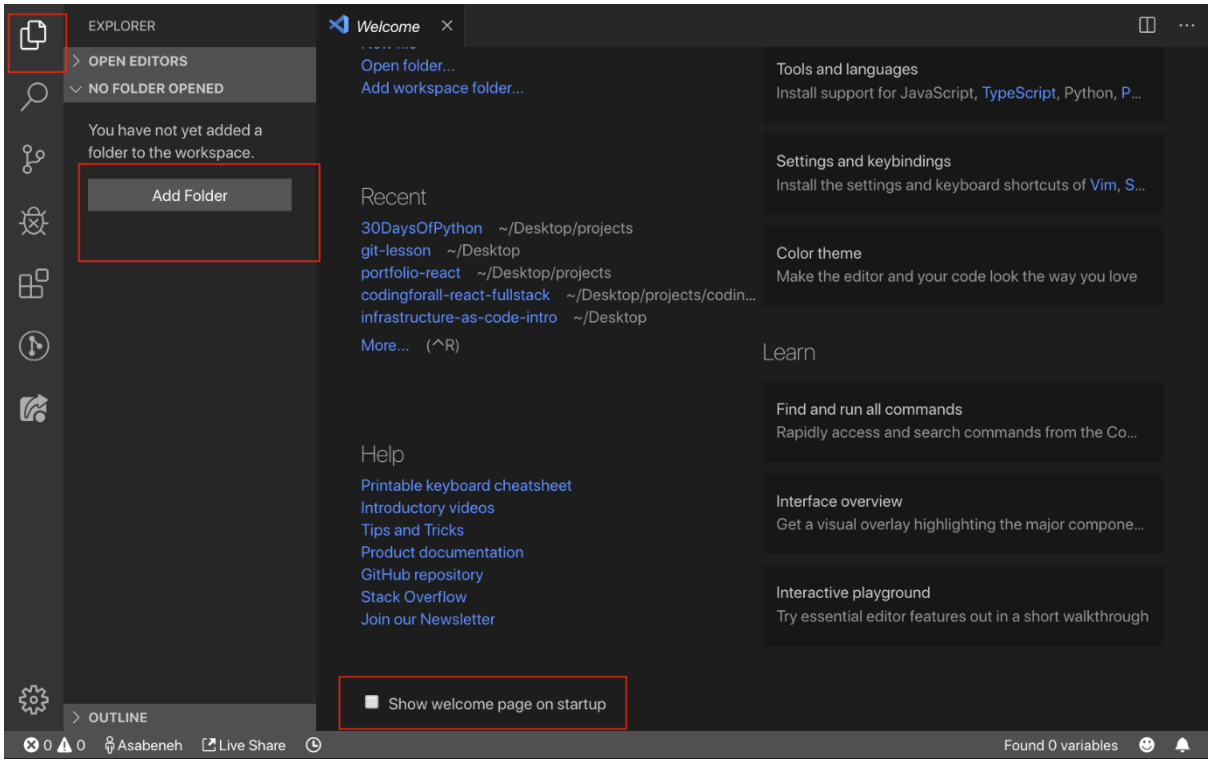
The Python interactive shell is good to try and test small script codes but it will not be for a big project. In real work environment, developers use different code editors to write codes. In this 30 days of Python programming challenge we will use visual studio code. Visual studio code is a very popular open source text editor. I am a fan of vscode and I would recommend to download visual studio code, but if you are in favor of other editors, feel free to follow with what you have.



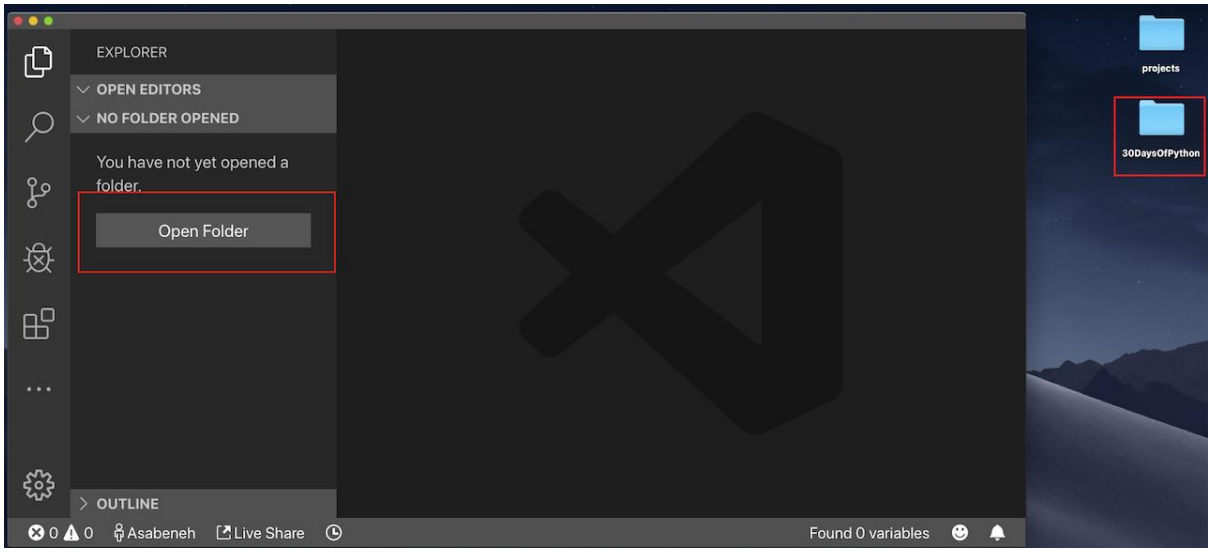
If you installed visual studio code, let us see how to use it. If you prefer a video, you can follow this Visual Studio Code for Python Video tutorial

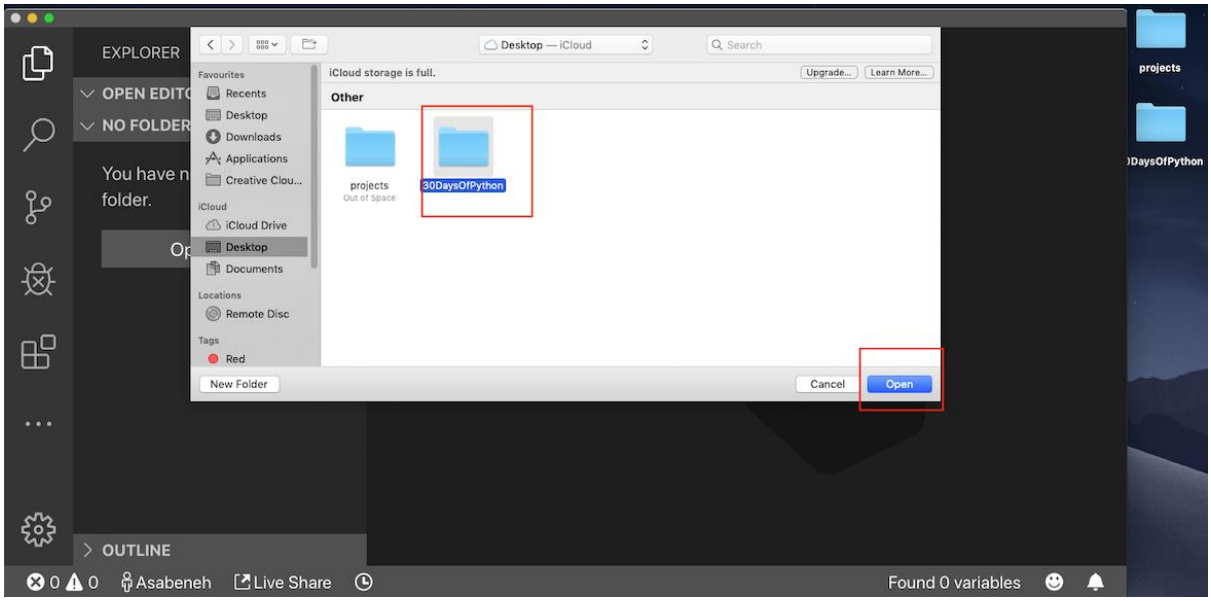
How to use visual studio code

Open the visual studio code by double clicking the visual studio icon. When you open it you will get this kind of interface. Try to interact with the labeled icons.

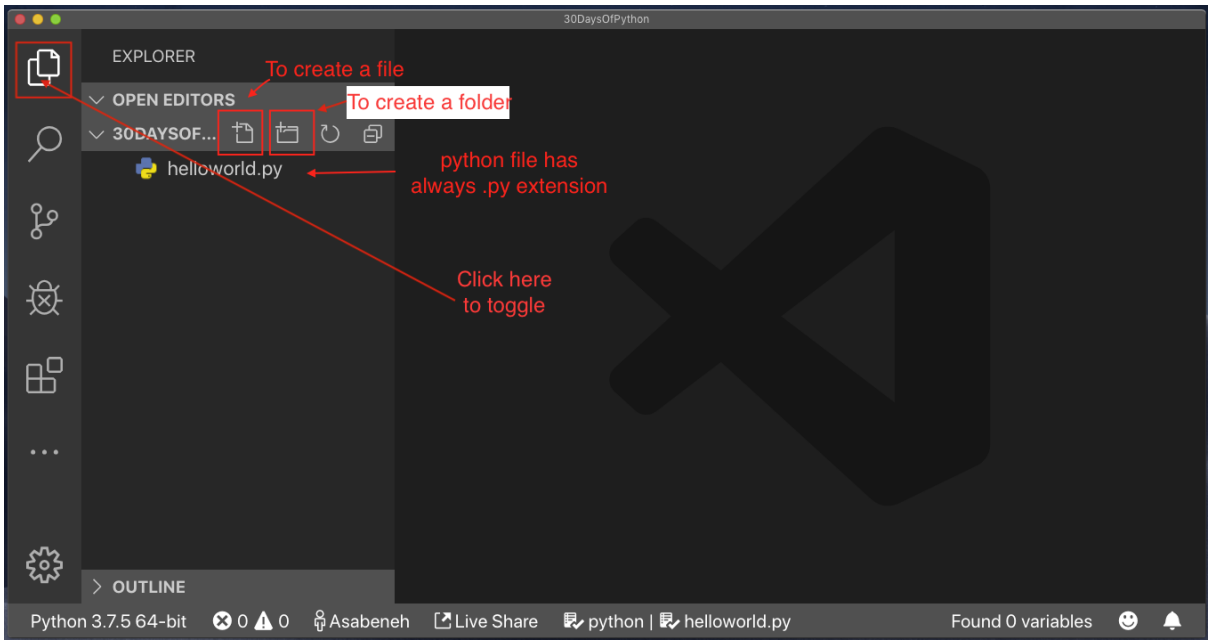


Create a folder named 30DaysOfPython on your desktop. Then open it using visual studio code.

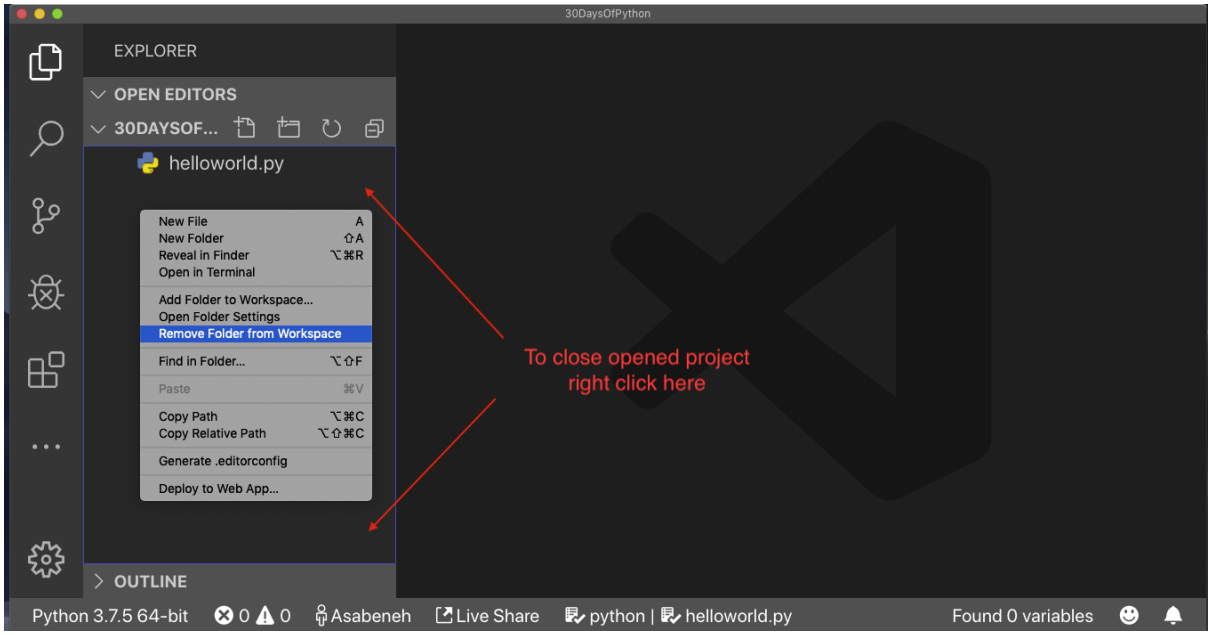




After opening it you will see shortcuts for creating files and folders inside of 30DaysOfPython project's directory. As you can see below, I have created the very first file, helloworld.py. You can do the same.



After a long day of coding, you want to close your code editor, right? This is how you will close the opened project.



Congratulations, you have finished setting up the development environment. Let us start coding.

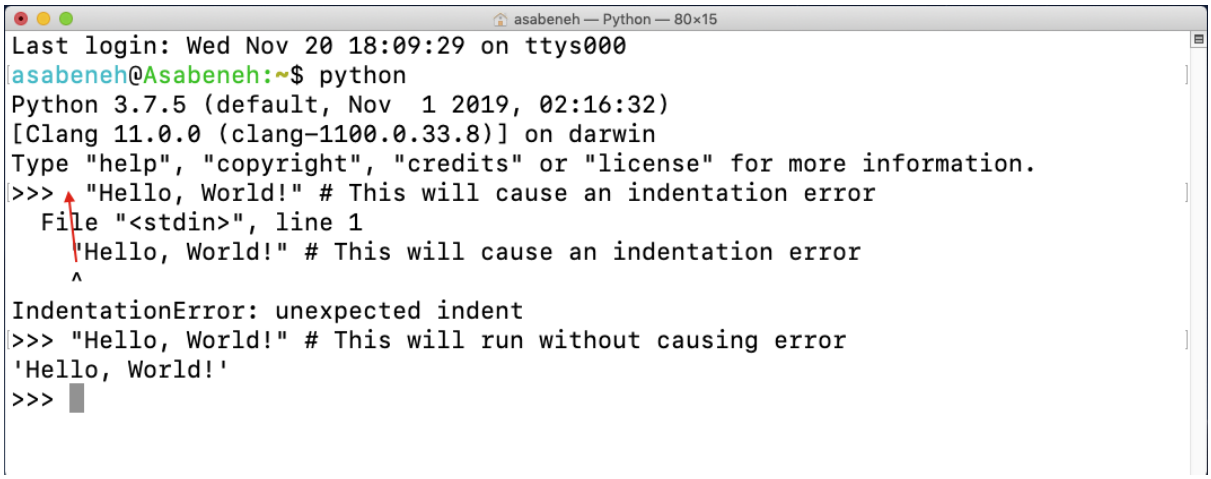
Basic Python

Python Syntax

A Python script can be written in Python interactive shell or in the code editor. A Python file has an extension .py.

Python Indentation

An indentation is a white space in a text. Indentation in many languages is used to increase code readability; however, Python uses indentation to create blocks of code. In other programming languages, curly brackets are used to create code blocks instead of indentation. One of the common bugs when writing Python code is incorrect indentation.



Comments

Comments play a crucial role in enhancing code readability and allowing developers to leave notes within their code. In Python, any text preceded by a hash (#) symbol is considered a comment and is not executed when the code runs.

Example: Single Line Comment

```
    # This is the first comment
    # This is the second comment
    # Python is eating the world
```

Example: Multiline Comment

Triple quote can be used for multiline comment if it is not assigned to a variable

```
"""This is multiline comment
multiline comment takes multiple lines.
python is eating the world
"""
```

Data types

In Python there are several types of data types. Let us get started with the most common ones. Different data types will be covered in detail in other sections. For the time being, let us just go through the different data types and get familiar with them. You do not have to have a clear understanding now.

Number

- Integer: Integer(negative, zero and positive) numbers Example: ... -3, -2, -1, 0, 1, 2, 3 ...
- Float: Decimal number Example ... -3.5, -2.25, -1.0, 0.0, 1.1, 2.2, 3.5 ...
- Complex Example 1 + j, 2 + 4j

String

A collection of one or more characters under a single or double quote. If a string is more than one sentence then we use a triple quote.

Example:

```
'Asabeneh'
'Finland'
'Python'
'I love teaching'
'I hope you are enjoying the first day of 30DaysOfPython Challenge'
```

Booleans

A boolean data type is either a True or False value. T and F should be always uppercase.

Example:

```
    True  #  Is the light on? If it is on, then the value is True
    False # Is the light on? If it is off, then the value is False
```

List

Python list is an ordered collection which allows to store different data type items. A list is similar to an array in JavaScript.

Example:

```
[0, 1, 2, 3, 4, 5]  # all are the same data types - a list of numbers
['Banana', 'Orange', 'Mango', 'Avocado'] # all the same data types - a list of strings (fruits)
['Finland','Estonia', 'Sweden','Norway'] # all the same data types - a list of strings (countries)
```

```
['Banana', 10, False, 9.81] # different data types in the list - string, integer, boolean and float
```

Dictionary

A Python dictionary object is an unordered collection of data in a key value pair format.

Example:

```
{
'first_name':'Asabeneh',
'last_name':'Yetayeh',
'country':'Finland',
'age':250,
'is_married':True,
'skills':['JS', 'React', 'Node', 'Python']
}
```

Tuple

A tuple is an ordered collection of different data types like list but tuples can not be modified once they are created. They are immutable.

Example:

```
('Asabeneh', 'Pawel', 'Brook', 'Abraham', 'Lidiya') # Names
('Earth', 'Jupiter', 'Neptune', 'Mars', 'Venus', 'Saturn', 'Uranus', 'Mercury') # planets
```

Set

A set is a collection of data types similar to list and tuple. Unlike list and tuple, set is not an ordered collection of items. Like in Mathematics, set in Python stores only unique items.

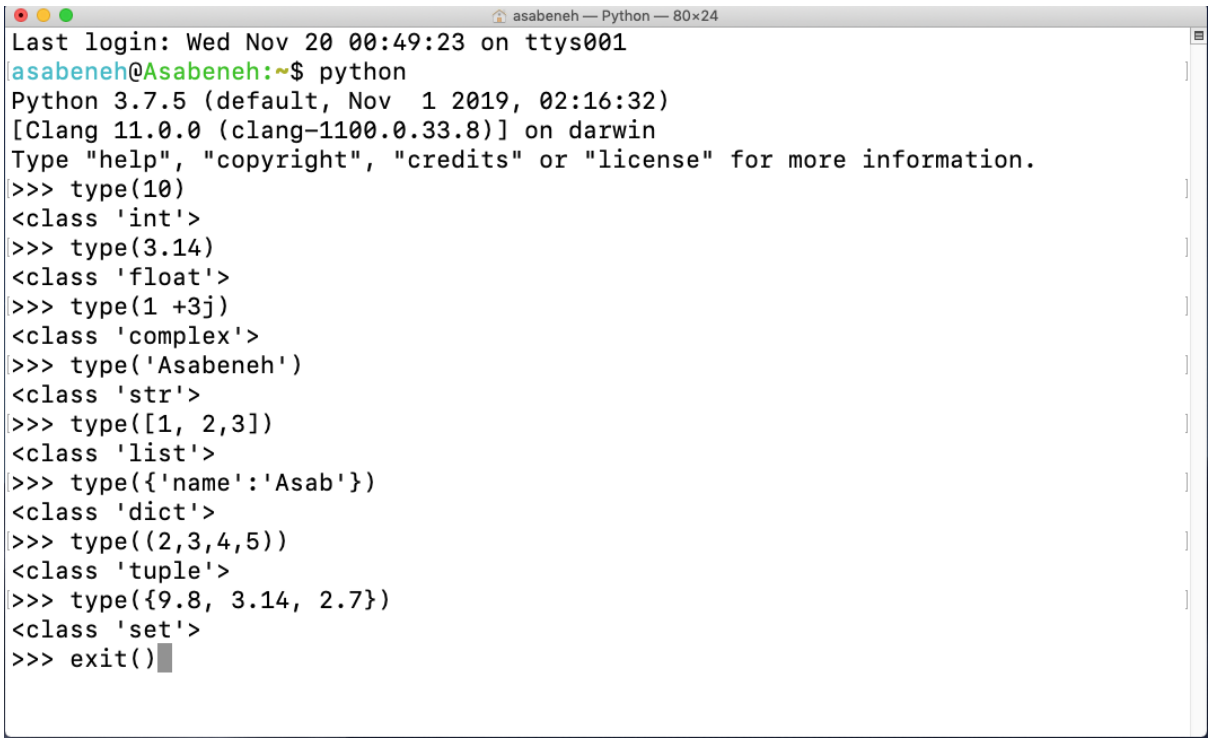
In later sections, we will go in detail about each and every Python data type.

Example:

```
{2, 4, 3, 5}
{3.14, 9.81, 2.7} # order is not important in set
```

Checking Data types

To check the data type of certain data/variable we use the **type** function. In the following terminal you will see different python data types:



Python File

First open your project folder, 30DaysOfPython. If you don't have this folder, create a folder name called 30DaysOfPython. Inside this folder, create a file called helloworld.py. Now, let's do what we did on python interactive shell using visual studio code.

The Python interactive shell was printing without using **print** but on visual studio code to see our result we should use a built in function _print(). The *print()* built-in function takes one or more arguments as follows *print('arument1', 'argument2', 'argument3')*. See the examples below.

Example:

The file name is helloworld.py

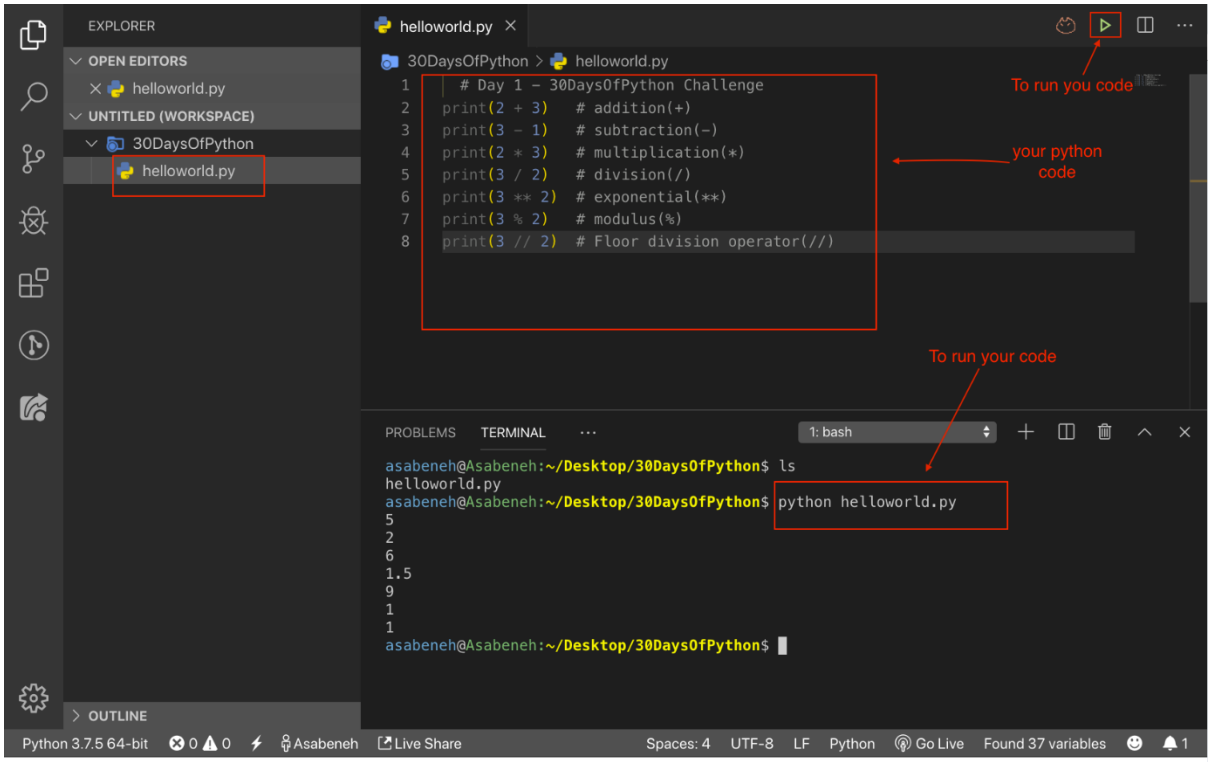
```
# Day 1 - 30DaysOfPython Challenge

print(2 + 3)              # addition(+)
print(3 - 1)              # subtraction(-)
print(2 * 3)              # multiplication(*)
print(3 / 2)              # division(/)
print(3 ** 2)             # exponential(**)
print(3 % 2)              # modulus(%)
print(3 // 2)             # Floor division operator(//)

# Checking data types
```

```
print(type(10))           # Int
print(type(3.14))         # Float
print(type(1 + 3j))       # Complex number
print(type('Asabeneh'))   # String
print(type([1, 2, 3]))    # List
print(type({'name':'Asabeneh'})) # Dictionary
print(type({9.8, 3.14, 2.7}))    # Set
print(type((9.8, 3.14, 2.7)))    # Tuple
```

To run the python file check the image below. You can run the python file either by running the green button on Visual Studio Code or by typing *python helloworld.py* in the terminal .



You are amazing. You have just completed day 1 challenge and you are on your way to greatness. Now do some exercises for your brain and muscles.

Exercises - Day 1

Exercise: Level 1

1. Check the python version you are using
2. Open the python interactive shell and do the following operations. The operands are 3 and 4.
    - addition(+)
    - subtraction(-)
    - multiplication(*)
    - modulus(%)
    - division(/)
    - exponential(**)
    - floor division operator(//)
3. Write strings on the python interactive shell. The strings are the following:
    - Your name
    - Your family name
    - Your country
    - I am enjoying 30 days of python
4. Check the data types of the following data:
    - 10
    - 9.8
    - 3.14
    - 4 - 4j
    - ['Asabeneh', 'Python', 'Finland']
    - Your name
    - Your family name
    - Your country

Exercise: Level 2

1. Create a folder named day_1 inside 30DaysOfPython folder. Inside day_1 folder, create a python file helloworld.py and repeat questions 1, 2, 3 and 4. Remember to use *print()* when you are working on a python file. Navigate to the directory where you have saved your file, and run it.

Exercise: Level 3

1. Write an example for different Python data types such as Number(Integer, Float, Complex), String, Boolean, List, Tuple, Set and Dictionary.
2. Find an Euclidian distance between (2, 3) and (10, 8)

🎉 CONGRATULATIONS ! 🎉

DAY-2 VARIABLES AND FUNCTIONS

Built in functions

In Python we have lots of built-in functions. Built-in functions are globally available for your use that mean you can make use of the built-in functions without importing or configuring. Some of the most commonly used Python built-in functions are the following: *print()*, *len()*, *type()*, *int()*, *float()*, *str()*, *input()*, *list()*, *dict()*, *min()*, *max()*, *sum()*, *sorted()*, *open()*, *file()*, *help()*, and *dir()*. In the following table you will see an exhaustive list of Python built-in functions taken from python documentation.

		Built-in Functions		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

Let us open the Python shell and start using some of the most common built-in functions.

```
asabeneh — Python — 80×21
Last login: Wed Nov 20 20:41:35 on ttys002
asabeneh@Asabeneh:~$ python
Python 3.7.5 (default, Nov  1 2019, 02:16:32)
[Clang 11.0.0 (clang-1100.0.33.8)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Hello, World!') # it prints the text value Hello, World!
Hello, World!
>>> len('Hello, World!') # it counts the number of characters including space
13
>>> type('Hello, World!') # it checks the data type
<class 'str'>
>>> str(10) # it converts number to string
'10'
>>> int('10') # it converts to number
10
>>> float(10) # it converts integer to decimal
10.0
>>> input('Enter your name:') # it takes user input
Enter your name:Asabeneh
'Asabeneh'
>>> exit()
```

Let us practice more by using different built-in functions

```
asabeneh — Python — 80×35
Last login: Wed Nov 20 20:42:55 on ttys002
asabeneh@Asabeneh:~$ python
Python 3.7.5 (default, Nov  1 2019, 02:16:32)
[Clang 11.0.0 (clang-1100.0.33.8)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> help('keywords') # prints all python reserved words

Here is a list of the Python keywords.  Enter any keyword to get more help.

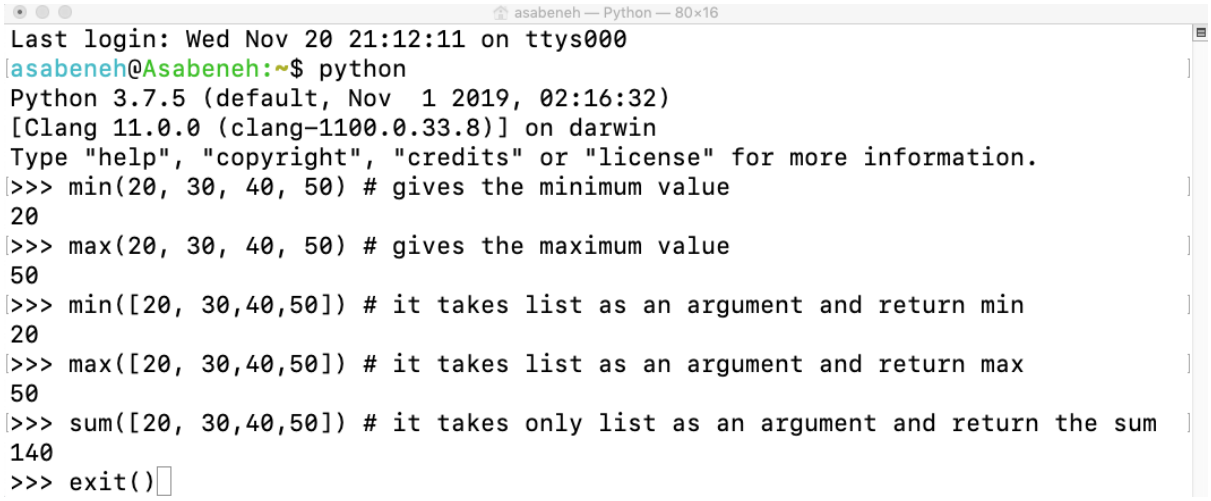
False               class               from                or
None                continue            global              pass
True                def                 if                  raise
and                 del                 import              return
as                  elif                in                  try
assert              else                is                  while
async               except              lambda              with
await               finally             nonlocal            yield
break               for                 not

>>> help(str) # give informaton about string

>>> dir(str) # give information about string
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '_
_eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs
__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__'
, '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__'
, '__str__', '__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'e
ncode', 'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index', 'isal
num', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower', '
isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lo
wer', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust',
 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip',
 'swapcase', 'title', 'translate', 'upper', 'zfill']
>>>
```

As you can see from the terminal above, Python has got reserved words. We do not use reserved words to declare variables or functions. We will cover variables in the next section.

I believe, by now you are familiar with built-in functions. Let us do one more practice of built-in functions and we will move on to the next section.



Variables

Variables store data in a computer memory. Mnemonic variables are recommended to use in many programming languages. A mnemonic variable is a variable name that can be easily remembered and associated. A variable refers to a memory address in which data is stored. Number at the beginning, special character, hyphen are not allowed when naming a variable. A variable can have a short name (like x, y, z), but a more descriptive name (firstname, lastname, age, country) is highly recommended.

Python Variable Name Rules

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (firstname, Firstname, FirstName and FIRSTNAME) are different variables)

Here are some example of valid variable names:

```
firstname
lastname
age
country
city
first_name
last_name
```

```
capital_city
_if # if we want to use reserved word as a variable
year_2021
year2021
current_year_2021
birth_year
num1
num2
```

Invalid variables names

```
first-name
first@name
first$name
num-1
1num
```

We will use standard Python variable naming style which has been adopted by many Python developers. Python developers use snake case(snake_case) variable naming convention. We use underscore character after each word for a variable containing more than one word(eg. first_name, last_name, engine_rotation_speed). The example below is an example of standard naming of variables, underscore is required when the variable name is more than one word.

When we assign a certain data type to a variable, it is called variable declaration. For instance in the example below my first name is assigned to a variable first_name. The equal sign is an assignment operator. Assigning means storing data in the variable. The equal sign in Python is not equality as in Mathematics.

Example:

```
# Variables in Python
first_name = 'Asabeneh'
last_name = 'Yetayeh'
country = 'Finland'
city = 'Helsinki'
age = 250
is_married = True
skills = ['HTML', 'CSS', 'JS', 'React', 'Python']
person_info = {
   'firstname':'Asabeneh',
   'lastname':'Yetayeh',
   'country':'Finland',
   'city':'Helsinki'
   }
```

Let us use the *print()* and *len()* built-in functions. Print function takes unlimited number of arguments. An argument is a value which we can be passed or put inside the function parenthesis, see the example below.

Example:

```
print('Hello, World!') # The text Hello, World! is an argument
print('Hello',',', 'World','!') # it can take multiple
arguments, four arguments have been passed
print(len('Hello, World!')) # it takes only one argument
```

Let us print and also find the length of the variables declared at the top:

Example:

```
# Printing the values stored in the variables

print('First name:', first_name)
print('First name length:', len(first_name))
print('Last name: ', last_name)
print('Last name length: ', len(last_name))
print('Country: ', country)
print('City: ', city)
print('Age: ', age)
print('Married: ', is_married)
print('Skills: ', skills)
print('Person information: ', person_info)
```

Declaring Multiple Variable in a Line

Multiple variables can also be declared in one line:

Example:

```
first_name, last_name, country, age, is_married = 'Asabeneh',
'Yetayeh', 'Helsink', 250, True

print(first_name, last_name, country, age, is_married)
print('First name:', first_name)
print('Last name: ', last_name)
print('Country: ', country)
print('Age: ', age)
print('Married: ', is_married)
```

Getting user input using the *input()* built-in function. Let us assign the data we get from a user into first_name and age variables. **Example:**

```
first_name = input('What is your name: ')
age = input('How old are you? ')

print(first_name)
print(age)
```

Data Types

There are several data types in Python. To identify the data type we use the *type* built-in function. I would like to ask you to focus on understanding different data types very well. When it comes to programming, it is all about data types. I introduced data types at the very beginning and it comes again, because every topic is related to data types. We will cover data types in more detail in their respective sections.

Checking Data types and Casting

- Check Data types: To check the data type of certain data/variable we use the *type* **Examples:**

```
# Different python data types
# Let's declare variables with various data types

first_name = 'Asabeneh'     # str
last_name = 'Yetayeh'       # str
country = 'Finland'         # str
city= 'Helsinki'            # str
age = 250                   # int, it is not my real age,
don't worry about it

# Printing out types
print(type('Asabeneh'))          # str
print(type(first_name))          # str
print(type(10))                  # int
print(type(3.14))                # float
print(type(1 + 1j))              # complex
print(type(True))                # bool
print(type([1, 2, 3, 4]))        # list
print(type({'name':'Asabeneh'})) # dict
print(type((1,2)))               # tuple
print(type(zip([1,2],[3,4])))    # zip
```

- Casting: Converting one data type to another data type. We use *int()*, *float()*, *str()*, *list*, *set* When we do arithmetic operations string numbers should be first converted to int or float otherwise it will return an error. If we concatenate a number with a string, the number should be first converted to a string. We will talk about concatenation in String section.

Examples:

```
# int to float
num_int = 10
print('num_int',num_int)         # 10
num_float = float(num_int)
print('num_float:', num_float)   # 10.0
```

```
# float to int
gravity = 9.81
print(int(gravity))             # 9

# int to str
num_int = 10
print(num_int)                  # 10
num_str = str(num_int)
print(num_str)                  # '10'

# str to int or float
num_str = '10.6'
num_float = float(num_str)
print('num_float', float(num_str))  # 10.6
num_int = int(num_float)
print('num_int', int(num_int))      # 10

# str to list
first_name = 'Asabeneh'
print(first_name)               # 'Asabeneh'
first_name_to_list = list(first_name)
print(first_name_to_list)            # ['A', 's', 'a', 'b',
'e', 'n', 'e', 'h']
```

Numbers

Number data types in Python:

1. Integers: Integer(negative, zero and positive) numbers Example: ... -3, -2, -1, 0, 1, 2, 3 ...
2. Floating Point Numbers(Decimal numbers) Example: ... -3.5, -2.25, -1.0, 0.0, 1.1, 2.2, 3.5 ...
3. Complex Numbers Example: 1 + j, 2 + 4j, 1 - 1j

🎉 You are awesome. You have just completed day 2 challenges and you are two steps ahead on your way to greatness. Now do some exercises for your brain and muscles.

Exercises - Day 2

Exercises: Level 1

1. Inside 30DaysOfPython create a folder called day_2. Inside this folder create a file named variables.py
2. Write a python comment saying 'Day 2: 30 Days of python programming'
3. Declare a first name variable and assign a value to it
4. Declare a last name variable and assign a value to it
5. Declare a full name variable and assign a value to it
6. Declare a country variable and assign a value to it
7. Declare a city variable and assign a value to it
8. Declare an age variable and assign a value to it
9. Declare a year variable and assign a value to it
10. Declare a variable is_married and assign a value to it
11. Declare a variable is_true and assign a value to it
12. Declare a variable is_light_on and assign a value to it
13. Declare multiple variable on one line

Exercises: Level 2

1. Check the data type of all your variables using type() built-in function
2. Using the *len()* built-in function, find the length of your first name
3. Compare the length of your first name and your last name
4. Declare 5 as num_one and 4 as num_two
5. Add num_one and num_two and assign the value to a variable total
6. Subtract num_two from num_one and assign the value to a variable diff
7. Multiply num_two and num_one and assign the value to a variable product
8. Divide num_one by num_two and assign the value to a variable division
9. Use modulus division to find num_two divided by num_one and assign the value to a variable remainder
10. Calculate num_one to the power of num_two and assign the value to a variable exp
11. Find floor division of num_one by num_two and assign the value to a variable floor_division

12. The radius of a circle is 30 meters.
    i. Calculate the area of a circle and assign the value to a variable name of *area_of_circle*
    ii. Calculate the circumference of a circle and assign the value to a variable name of *circum_of_circle*
    iii. Take radius as user input and calculate the area.
13. Use the built-in input function to get first name, last name, country and age from a user and store the value to their corresponding variable names
14. Run help('keywords') in Python shell or in your file to check for the Python reserved words or keywords

🎉 CONGRATULATIONS ! 🎉

Day-3 Operators

Boolean

A boolean data type represents one of the two values: *True* or *False*. The use of these data types will be clear once we start using the comparison operator. The first letter **T** for True and **F** for False should be capital unlike JavaScript. **Example: Boolean Values**

```
print(True)
print(False)
```

Operators

Python language supports several types of operators. In this section, we will focus on few of them.

Assignment Operators

Assignment operators are used to assign values to variables. Let us take = as an example. Equal sign in mathematics shows that two values are equal, however in Python it means we are storing a value in a certain variable and we call it assignment or a assigning value to a variable. The table below shows the different types of python assignment operators, taken from w3school.

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Arithmetic Operators:

- Addition(+): a + b
- Subtraction(-): a - b
- Multiplication(*): a * b
- Division(/): a / b
- Modulus(%): a % b
- Floor division(//): a // b
- Exponentiation(**): a ** b

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Example:Integers

Arithmetic Operations in Python

Integers

```
print('Addition: ', 1 + 2)         # 3
print('Subtraction: ', 2 - 1)      # 1
print('Multiplication: ', 2 * 3)   # 6
print ('Division: ', 4 / 2)        # 2.0  Division in Python
gives floating number
print('Division: ', 6 / 2)         # 3.0
print('Division: ', 7 / 2)         # 3.5
print('Division without the remainder: ', 7 // 2)    # 3,
gives without the floating number or without the remaining
print ('Division without the remainder: ',7 // 3)    # 2
print('Modulus: ', 3 % 2)          # 1, Gives the remainder
print('Exponentiation: ', 2 ** 3) # 9 it means 2 * 2 * 2
```

Example:Floats

Floating numbers

```
print('Floating Point Number, PI', 3.14)
print('Floating Point Number, gravity', 9.81)
```

Example:Complex numbers

```
# Complex numbers
print('Complex number: ', 1 + 1j)
print('Multiplying complex numbers: ',(1 + 1j) * (1 - 1j))
```

Let's declare a variable and assign a number data type. I am going to use single character variable but remember do not develop a habit of declaring such types of variables. Variable names should be all the time mnemonic.

Example:

```
# Declaring the variable at the top first

a = 3 # a is a variable name and 3 is an integer data type
b = 2 # b is a variable name and 3 is an integer data type

# Arithmetic operations and assigning the result to a variable
total = a + b
diff = a - b
product = a * b
division = a / b
remainder = a % b
floor_division = a // b
exponential = a ** b

# I should have used sum instead of total but sum is a built-
in function - try to avoid overriding built-in functions
print(total) # if you do not label your print with some
string, you never know where the result is coming from
print('a + b = ', total)
print('a - b = ', diff)
print('a * b = ', product)
print('a / b = ', division)
print('a % b = ', remainder)
print('a // b = ', floor_division)
print('a ** b = ', exponentiation)
```

Example:

```
print('== Addition, Subtraction, Multiplication, Division,
Modulus ==')

# Declaring values and organizing them together
num_one = 3
num_two = 4

# Arithmetic operations
total = num_one + num_two
diff = num_two - num_one
product = num_one * num_two
div = num_two / num_one
remainder = num_two % num_one

# Printing values with label
print('total: ', total)
print('difference: ', diff)
print('product: ', product)
print('division: ', div)
print('remainder: ', remainder)
```

Let us start start connecting the dots and start making use of what we already know to calculate (area, volume,density, weight, perimeter, distance, force).

Example:

```
# Calculating area of a circle
radius = 10                                  # radius of a
circle
area_of_circle = 3.14 * radius ** 2          # two * sign means
exponent or power
print('Area of a circle:', area_of_circle)

# Calculating area of a rectangle
length = 10
width = 20
area_of_rectangle = length * width
print('Area of rectangle:', area_of_rectangle)

# Calculating a weight of an object
mass = 75
gravity = 9.81
weight = mass * gravity
print(weight, 'N')                           # Adding unit to
the weight

# Calculate the density of a liquid
mass = 75 # in Kg
volume = 0.075 # in cubic meter
density = mass / volume # 1000 Kg/m^3
```

Comparison Operators

In programming we compare values, we use comparison operators to compare two values. We check if a value is greater or less or equal to other value. The following table shows Python comparison operators which was taken from w3shool.

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Example: Comparison Operators

```
print(3 > 2)     # True, because 3 is greater than 2
print(3 >= 2)    # True, because 3 is greater than 2
print(3 < 2)     # False,  because 3 is greater than 2
print(2 < 3)     # True, because 2 is less than 3
print(2 <= 3)    # True, because 2 is less than 3
print(3 == 2)    # False, because 3 is not equal to 2
print(3 != 2)    # True, because 3 is not equal to 2
print(len('mango') == len('avocado'))  # False
print(len('mango') != len('avocado'))  # True
print(len('mango') < len('avocado'))   # True
print(len('milk') != len('meat'))      # False
print(len('milk') == len('meat'))      # True
print(len('tomato') == len('potato'))  # True
print(len('python') > len('dragon'))   # False


# Comparing something gives either a True or False

print('True == True: ', True == True)
print('True == False: ', True == False)
print('False == False:', False == False)
```

In addition to the above comparison operator Python uses:

- *is*: Returns true if both variables are the same object(x is y)
- *is not*: Returns true if both variables are not the same object(x is not y)
- *in*: Returns True if the queried list contains a certain item(x in y)

- *not in*: Returns True if the queried list doesn't have a certain item(x in y)

```
print('1 is 1', 1 is 1)                    # True - because the
data values are the same
print('1 is not 2', 1 is not 2)            # True - because 1
is not 2
print('A in Asabeneh', 'A' in 'Asabeneh') # True - A found in
the string
print('B in Asabeneh', 'B' in 'Asabeneh') # False - there is
no uppercase B
print('coding' in 'coding for all') # True - because coding
for all has the word coding
print('a in an:', 'a' in 'an')      # True
print('4 is 2 ** 2:', 4 is 2 ** 2)   # True
```

Logical Operators

Unlike other programming languages python uses keywords *and*, *or* and *not* for logical operators. Logical operators are used to combine conditional statements:

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

```
print(3 > 2 and 4 > 3) # True - because both statements are
true
print(3 > 2 and 4 < 3) # False - because the second statement
is false
print(3 < 2 and 4 < 3) # False - because both statements are
false
print('True and True: ', True and True)
print(3 > 2 or 4 > 3)  # True - because both statements are
true
print(3 > 2 or 4 < 3)  # True - because one of the statements
is true
print(3 < 2 or 4 < 3)  # False - because both statements are
false
print('True or False:', True or False)
print(not 3 > 2)     # False - because 3 > 2 is true, then not
True gives False
print(not True)      # False - Negation, the not operator
turns true to false
print(not False)     # True
```

```
print(not not True)  # True
print(not not False) # False
```

🥳 You have boundless energy. You have just completed day 3 challenges and you are three steps ahead on your way to greatness. Now do some exercises for your brain and your muscles.

💻 Exercises - Day 3

1. Declare your age as integer variable
2. Declare your height as a float variable
3. Declare a variable that store a complex number
4. Write a script that prompts the user to enter base and height of the triangle and calculate an area of this triangle (area = 0.5 x b x h).

```
    Enter base: 20
    Enter height: 10
    The area of the triangle is 100
```

5. Write a script that prompts the user to enter side a, side b, and side c of the triangle. Calculate the perimeter of the triangle (perimeter = a + b + c).

```
Enter side a: 5
Enter side b: 4
Enter side c: 3
The perimeter of the triangle is 12
```

6. Get length and width of a rectangle using prompt. Calculate its area (area = length x width) and perimeter (perimeter = 2 x (length + width))
7. Get radius of a circle using prompt. Calculate the area (area = pi x r x r) and circumference (c = 2 x pi x r) where pi = 3.14.
8. Calculate the slope, x-intercept and y-intercept of y = 2x -2
9. Slope is (m = y2-y1/x2-x1). Find the slope and Euclidean distance between point (2, 2) and point (6,10)
10. Compare the slopes in tasks 8 and 9.
11. Calculate the value of y (y = x^2 + 6x + 9). Try to use different x values and figure out at what x value y is going to be 0.
12. Find the length of 'python' and 'dragon' and make a falsy comparison statement.
13. Use *and* operator to check if 'on' is found in both 'python' and 'dragon'
14. *I hope this course is not full of jargon*. Use *in* operator to check if *jargon* is in the sentence.
15. There is no 'on' in both dragon and python
16. Find the length of the text *python* and convert the value to float and convert it to string
17. Even numbers are divisible by 2 and the remainder is zero. How do you check if a number is even or not using python?

18. Check if the floor division of 7 by 3 is equal to the int converted value of 2.7.
19. Check if type of '10' is equal to type of 10
20. Check if int('9.8') is equal to 10
21. Writ a script that prompts the user to enter hours and rate per hour. Calculate pay of the person?

```
Enter hours: 40
Enter rate per hour: 28
Your weekly earning is 1120
```

22. Write a script that prompts the user to enter number of years. Calculate the number of seconds a person can live. Assume a person can live hundred years

```
Enter number of years you have lived: 100
You have lived for 3153600000 seconds.
```

23. Write a Python script that displays the following table

```
1 1 1 1 1
2 1 2 4 8
3 1 3 9 27
4 1 4 16 64
5 1 5 25 125
```

🎉 CONGRATULATIONS ! 🎉

DAY-4 STRINGS

Strings

Text is a string data type. Any data type written as text is a string. Any data under single, double or triple quote are strings. There are different string methods and built-in functions to deal with string data types. To check the length of a string use the len() method.

Creating a String

```
letter = 'P'                 # A string could be a single
character or a bunch of texts
print(letter)                # P
print(len(letter))           # 1
greeting = 'Hello, World!'  # String could be made using a
single or double quote,"Hello, World!"
print(greeting)              # Hello, World!
print(len(greeting))         # 13
sentence = "I hope you are enjoying 30 days of Python
Challenge"
print(sentence)
```

Multiline string is created by using triple single (''') or triple double quotes (""""). See the example below.

```
multiline_string = '''I am a teacher and enjoy teaching.
I didn't find anything as rewarding as empowering people.
That is why I created 30 days of python.'''
print(multiline_string)

# Another way of doing the same thing
multiline_string = """I am a teacher and enjoy teaching.
I didn't find anything as rewarding as empowering people.
That is why I created 30 days of python."""
print(multiline_string)
```

String Concatenation

We can connect strings together. Merging or connecting strings is called concatenation. See the example below:

```
first_name = 'Asabeneh'
last_name = 'Yetayeh'
space = ' '
full_name = first_name  +  space + last_name
print(full_name) # Asabeneh Yetayeh
```

```
# Checking the length of a string using len() built-in
function
print(len(first_name))  # 8
print(len(last_name))   # 7
print(len(first_name) > len(last_name)) # True
print(len(full_name)) # 16
```

Escape Sequences in Strings

In Python and other programming languages \ followed by a character is an escape sequence. Let us see the most common escape characters:

- \n: new line
- \t: Tab means(8 spaces)
- \\: Back slash
- \': Single quote (')
- \": Double quote (")

Now, let us see the use of the above escape sequences with examples.

```
print('I hope everyone is enjoying the Python Challenge.\nAre
you ?') # line break
print('Days\tTopics\tExercises') # adding tab space or 4
spaces
print('Day 1\t5\t5')
print('Day 2\t6\t20')
print('Day 3\t5\t23')
print('Day 4\t1\t35')
print('This is a backslash  symbol (\\)') # To write a
backslash
print('In every programming language it starts with \"Hello,
World!\"') # to write a double quote inside a single quote

# output
I hope every one is enjoying the Python Challenge.
Are you ?
Days Topics    Exercises
Day 1     5        5
Day 2     6        20
Day 3     5        23
Day 4     1        35
This is a backslash  symbol (\)
In every programming language it starts with "Hello, World!"
```

String formatting

Old Style String Formatting (% Operator)

In Python there are many ways of formatting strings. In this section, we will cover some of them. The "%" operator is used to format a set of variables enclosed in a "tuple" (a fixed size list), together with a format string, which contains normal text together with "argument specifiers", special symbols like "%s", "%d", "%f", "%.number of digitsf".

- %s - String (or any object with a string representation, like numbers)
- %d - Integers
- %f - Floating point numbers
- "%.number of digitsf" - Floating point numbers with fixed precision

```
# Strings only
first_name = 'Asabeneh'
last_name = 'Yetayeh'
language = 'Python'
formated_string = 'I am %s %s. I teach %s' %(first_name,
last_name, language)
print(formated_string)

# Strings  and numbers
radius = 10
pi = 3.14
area = pi * radius ** 2
formated_string = 'The area of circle with a radius %d is
%.2f.' %(radius, area) # 2 refers the 2 significant digits
after the point

python_libraries = ['Django', 'Flask', 'NumPy',
'Matplotlib','Pandas']
formated_string = 'The following are python libraries:%s' %
(python_libraries)
print(formated_string) # "The following are python
libraries:['Django', 'Flask', 'NumPy', 'Matplotlib','Pandas']"
```

New Style String Formatting (str.format)

This formatting is introduced in Python version 3.

```
first_name = 'Asabeneh'
last_name = 'Yetayeh'
language = 'Python'
formated_string = 'I am {} {}. I teach {}'.format(first_name,
last_name, language)
```

```
print(formated_string)
a = 4
b = 3

print('{} + {} = {}'.format(a, b, a + b))
print('{} - {} = {}'.format(a, b, a - b))
print('{} * {} = {}'.format(a, b, a * b))
print('{} / {} = {:.2f}'.format(a, b, a / b)) # limits it to
two digits after decimal
print('{} % {} = {}'.format(a, b, a % b))
print('{} // {} = {}'.format(a, b, a // b))
print('{} ** {} = {}'.format(a, b, a ** b))

# output
4 + 3 = 7
4 - 3 = 1
4 * 3 = 12
4 / 3 = 1.33
4 % 3 = 1
4 // 3 = 1
4 ** 3 = 64

# Strings  and numbers
radius = 10
pi = 3.14
area = pi * radius ** 2
formated_string = 'The area of a circle with a radius {} is
{:.2f}.'.format(radius, area) # 2 digits after decimal
print(formated_string)
```

String Interpolation / f-Strings (Python 3.6+)

Another new string formatting is string interpolation, f-strings. Strings start with f and we can inject the data in their corresponding positions.

```
a = 4
b = 3
print(f'{a} + {b} = {a +b}')
print(f'{a} - {b} = {a - b}')
print(f'{a} * {b} = {a * b}')
print(f'{a} / {b} = {a / b:.2f}')
print(f'{a} % {b} = {a % b}')
print(f'{a} // {b} = {a // b}')
print(f'{a} ** {b} = {a ** b}')
```

Python Strings as Sequences of Characters

Python strings are sequences of characters, and share their basic methods of access with other Python ordered sequences of objects – lists and tuples. The simplest way of extracting single characters from strings (and individual members from any sequence) is to unpack them into corresponding variables.

Unpacking Characters

language = 'Python'

a,b,c,d,e,f = language # unpacking sequence characters into variables

print(a) # P

print(b) # y

print(c) # t

print(d) # h

print(e) # o

print(f) # n

Accessing Characters in Strings by Index

In programming counting starts from zero. Therefore the first letter of a string is at zero index and the last letter of a string is the length of a string minus one.


```
language = 'Python'
first_letter = language[0]
print(first_letter) # P
second_letter = language[1]
print(second_letter) # y
last_index = len(language) - 1
last_letter = language[last_index]
print(last_letter) # n
```

If we want to start from right end we can use negative indexing. -1 is the last index.

```
language = 'Python'
last_letter = language[-1]
print(last_letter) # n
second_last = language[-2]
```

```
print(second_last) # o
```

Slicing Python Strings

In python we can slice strings into substrings.

```
language = 'Python'
first_three = language[0:3] # starts at zero index and up to 3
but not include 3
print(first_three) #Pyt
last_three = language[3:6]
print(last_three) # hon
# Another way
last_three = language[-3:]
print(last_three)   # hon
last_three = language[3:]
print(last_three)   # hon
```

Reversing a String

We can easily reverse strings in python.

```
greeting = 'Hello, World!'
print(greeting[::-1]) # !dlroW ,olleH
```

Skipping Characters While Slicing

It is possible to skip characters while slicing by passing step argument to slice method.

```
language = 'Python'
pto = language[0:6:2] #
print(pto) # Pto
```

String Methods

There are many string methods which allow us to format strings. See some of the string methods in the following example:

- capitalize(): Converts the first character of the string to capital letter

```
challenge = 'thirty days of python'
print(challenge.capitalize()) # 'Thirty days of python'
```

- count(): returns occurrences of substring in string, count(substring, start=.., end=..). The start is a starting indexing for counting and end is the last index to count.

```
challenge = 'thirty days of python'
print(challenge.count('y')) # 3
print(challenge.count('y', 7, 14)) # 1,
print(challenge.count('th')) # 2`
```

- endswith(): Checks if a string ends with a specified ending

```
challenge = 'thirty days of python'
print(challenge.endswith('on'))   # True
print(challenge.endswith('tion')) # False
```

- expandtabs(): Replaces tab character with spaces, default tab size is 8. It takes tab size argument

```
challenge = 'thirty\tdays\tof\tpython'
print(challenge.expandtabs())   # 'thirty  days    of
python'
print(challenge.expandtabs(10)) # 'thirty    days      of
python'
```

- find(): Returns the index of the first occurrence of a substring, if not found returns -1

```
challenge = 'thirty days of python'
print(challenge.find('y'))  # 5
print(challenge.find('th')) # 0
```

- rfind(): Returns the index of the last occurrence of a substring, if not found returns -1

```
challenge = 'thirty days of python'
print(challenge.rfind('y'))  # 16
print(challenge.rfind('th')) # 17
```

- format(): formats string into a nicer output
  More about string formatting check this link

```
first_name = 'Asabeneh'
last_name = 'Yetayeh'
age = 250
job = 'teacher'
country = 'Finland'
sentence = 'I am {} {}. I am a {}. I am {} years old. I live
in {}.'.format(first_name, last_name, age, job, country)
print(sentence) # I am Asabeneh Yetayeh. I am 250 years old. I
am a teacher. I live in Finland.

radius = 10
pi = 3.14
area = pi * radius ** 2
result = 'The area of a circle with radius {} is
{}'.format(str(radius), str(area))
```

```
print(result) # The area of a circle with radius 10 is 314
```

- index(): Returns the lowest index of a substring, additional arguments indicate starting and ending index (default 0 and string length - 1). If the substring is not found it raises a valueError.

```
challenge = 'thirty days of python'
sub_string = 'da'
print(challenge.index(sub_string))  # 7
print(challenge.index(sub_string, 9)) # error
```

- rindex(): Returns the highest index of a substring, additional arguments indicate starting and ending index (default 0 and string length - 1)

```
challenge = 'thirty days of python'
sub_string = 'da'
print(challenge.rindex(sub_string))  # 7
print(challenge.rindex(sub_string, 9)) # error
print(challenge.rindex('on', 8)) # 19
```

- isalnum(): Checks alphanumeric character

```
challenge = 'ThirtyDaysPython'
print(challenge.isalnum()) # True

challenge = '30DaysPython'
print(challenge.isalnum()) # True

challenge = 'thirty days of python'
print(challenge.isalnum()) # False, space is not an
alphanumeric character

challenge = 'thirty days of python 2019'
print(challenge.isalnum()) # False
```

- isalpha(): Checks if all string elements are alphabet characters (a-z and A-Z)

```
challenge = 'thirty days of python'
print(challenge.isalpha()) # False, space is once again
excluded
challenge = 'ThirtyDaysPython'
print(challenge.isalpha()) # True
num = '123'
print(num.isalpha())      # False
```

- isdecimal(): Checks if all characters in a string are decimal (0-9)

```
challenge = 'thirty days of python'
print(challenge.isdecimal())  # False
challenge = '123'
print(challenge.isdecimal())  # True
challenge = '\u00B2'
print(challenge.isdigit())   # False
```

```
challenge = '12 3'
print(challenge.isdecimal())  # False, space not allowed
```

- isdigit(): Checks if all characters in a string are numbers (0-9 and some other unicode characters for numbers)

```
challenge = 'Thirty'
print(challenge.isdigit()) # False
challenge = '30'
print(challenge.isdigit())   # True
challenge = '\u00B2'
print(challenge.isdigit())   # True
```

- isnumeric(): Checks if all characters in a string are numbers or number related (just like isdigit(), just accepts more symbols, like ½)

```
num = '10'
print(num.isnumeric()) # True
num = '\u00BD' # ½
print(num.isnumeric()) # True
num = '10.5'
print(num.isnumeric()) # False
```

- isidentifier(): Checks for a valid identifier - it checks if a string is a valid variable name

```
challenge = '30DaysOfPython'
print(challenge.isidentifier()) # False, because it starts
with a number
challenge = 'thirty_days_of_python'
print(challenge.isidentifier()) # True
```

- islower(): Checks if all alphabet characters in the string are lowercase

```
challenge = 'thirty days of python'
print(challenge.islower()) # True
challenge = 'Thirty days of python'
print(challenge.islower()) # False
```

- isupper(): Checks if all alphabet characters in the string are uppercase

```
challenge = 'thirty days of python'
print(challenge.isupper()) #  False
challenge = 'THIRTY DAYS OF PYTHON'
print(challenge.isupper()) # True
```

- join(): Returns a concatenated string

```
web_tech = ['HTML', 'CSS', 'JavaScript', 'React']
result = ' '.join(web_tech)
print(result) # 'HTML CSS JavaScript React'
web_tech = ['HTML', 'CSS', 'JavaScript', 'React']
result = '# '.join(web_tech)
print(result) # 'HTML# CSS# JavaScript# React'
```

- strip(): Removes all given characters starting from the beginning and end of the string

```
challenge = 'thirty days of pythoonnn'
print(challenge.strip('noth')) # 'irty days of py'
```

- replace(): Replaces substring with a given string

```
challenge = 'thirty days of python'
print(challenge.replace('python', 'coding')) # 'thirty days of
coding'
```

- split(): Splits the string, using given string or space as a separator

```
challenge = 'thirty days of python'
print(challenge.split()) # ['thirty', 'days', 'of', 'python']
challenge = 'thirty, days, of, python'
print(challenge.split(', ')) # ['thirty', 'days', 'of',
'python']
```

- title(): Returns a title cased string

```
challenge = 'thirty days of python'
print(challenge.title()) # Thirty Days Of Python
```

- swapcase(): Converts all uppercase characters to lowercase and all lowercase characters to uppercase characters

```
challenge = 'thirty days of python'
print(challenge.swapcase())   # THIRTY DAYS OF PYTHON
challenge = 'Thirty Days Of Python'
print(challenge.swapcase())  # tHIRTY dAYS oF pYTHON
```

- startswith(): Checks if String Starts with the Specified String

```
challenge = 'thirty days of python'
print(challenge.startswith('thirty')) # True

challenge = '30 days of python'
print(challenge.startswith('thirty')) # False
```

🎉 You are an extraordinary person and you have a remarkable potential. You have just completed day 4 challenges and you are four steps a head in to your way to greatness. Now do some exercises for your brain and muscles.

💻 Exercises - Day 4

1. Concatenate the string 'Thirty', 'Days', 'Of', 'Python' to a single string, 'Thirty Days Of Python'.
2. Concatenate the string 'Coding', 'For' , 'All' to a single string, 'Coding For All'.
3. Declare a variable named company and assign it to an initial value "Coding For All".
4. Print the variable company using *print()*.
5. Print the length of the company string using *len()* method and *print()*.
6. Change all the characters to uppercase letters using *upper()* method.
7. Change all the characters to lowercase letters using *lower()* method.
8. Use capitalize(), title(), swapcase() methods to format the value of the string *Coding For All*.
9. Cut(slice) out the first word of *Coding For All* string.
10. Check if *Coding For All* string contains a word Coding using the method index, find or other methods.
11. Replace the word coding in the string 'Coding For All' to Python.
12. Change Python for Everyone to Python for All using the replace method or other methods.
13. Split the string 'Coding For All' using space as the separator (split()) .
14. "Facebook, Google, Microsoft, Apple, IBM, Oracle, Amazon" split the string at the comma.
15. What is the character at index 0 in the string *Coding For All*.
16. What is the last index of the string *Coding For All*.
17. What character is at index 10 in "Coding For All" string.
18. Create an acronym or an abbreviation for the name 'Python For Everyone'.
19. Create an acronym or an abbreviation for the name 'Coding For All'.
20. Use index to determine the position of the first occurrence of C in Coding For All.
21. Use index to determine the position of the first occurrence of F in Coding For All.
22. Use rfind to determine the position of the last occurrence of l in Coding For All People.

23. Use index or find to find the position of the first occurrence of the word 'because' in the following sentence: 'You cannot end a sentence with because because because is a conjunction'
24. Use rindex to find the position of the last occurrence of the word because in the following sentence: 'You cannot end a sentence with because because because is a conjunction'
25. Slice out the phrase 'because because because' in the following sentence: 'You cannot end a sentence with because because because is a conjunction'
26. Find the position of the first occurrence of the word 'because' in the following sentence: 'You cannot end a sentence with because because because is a conjunction'
27. Slice out the phrase 'because because because' in the following sentence: 'You cannot end a sentence with because because because is a conjunction'
28. Does ''Coding For All' start with a substring *Coding*?
29. Does 'Coding For All' end with a substring *coding*?
30. ' Coding For All ', remove the left and right trailing spaces in the given string.
31. Which one of the following variables return True when we use the method isidentifier():
    - 30DaysOfPython
    - thirty_days_of_python
32. The following list contains the names of some of python libraries: ['Django', 'Flask', 'Bottle', 'Pyramid', 'Falcon']. Join the list with a hash with space string.
33. Use the new line escape sequence to separate the following sentences.

```
I am enjoying this challenge.
I just wonder what is next.
```

34. Use a tab escape sequence to write the following lines.

```
Name      Age     Country   City
Asabeneh  250     Finland   Helsinki
```

35. Use the string formatting method to display the following:

```
radius = 10
area = 3.14 * radius ** 2
The area of a circle with radius 10 is 314 meters square.
```

36. Make the following using string formatting methods:

```
8 + 6 = 14
```

```
8 - 6 = 2
8 * 6 = 48
8 / 6 = 1.33
8 % 6 = 2
8 // 6 = 1
8 ** 6 = 262144
```

🎉 CONGRATULATIONS ! 🎉

DAY-5 LISTS

Lists

There are four collection data types in Python :

- List: is a collection which is ordered and changeable(modifiable). Allows duplicate members.
- Tuple: is a collection which is ordered and unchangeable or unmodifiable(immutable). Allows duplicate members.
- Set: is a collection which is unordered, un-indexed and unmodifiable, but we can add new items to the set. Duplicate members are not allowed.
- Dictionary: is a collection which is unordered, changeable(modifiable) and indexed. No duplicate members.

A list is collection of different data types which is ordered and modifiable(mutable). A list can be empty or it may have different data type items.

How to Create a List

In Python we can create lists in two ways:

- Using list built-in function

```
# syntax
lst = list()
empty_list = list() # this is an empty list, no item in the list
print(len(empty_list)) # 0
```

- Using square brackets, []

```
# syntax
lst = []
empty_list = [] # this is an empty list, no item in the list
print(len(empty_list)) # 0
```

Lists with initial values. We use *len()* to find the length of a list.

```
fruits = ['banana', 'orange', 'mango', 'lemon']
# list of fruits
```

```
vegetables = ['Tomato', 'Potato', 'Cabbage','Onion', 'Carrot']
# list of vegetables
animal_products = ['milk', 'meat', 'butter', 'yoghurt']
# list of animal products
web_techs = ['HTML', 'CSS', 'JS', 'React','Redux', 'Node',
'MongDB'] # list of web technologies
countries = ['Finland', 'Estonia', 'Denmark', 'Sweden',
'Norway']

# Print the lists and its length
print('Fruits:', fruits)
print('Number of fruits:', len(fruits))
print('Vegetables:', vegetables)
print('Number of vegetables:', len(vegetables))
print('Animal products:',animal_products)
print('Number of animal products:', len(animal_products))
print('Web technologies:', web_techs)
print('Number of web technologies:', len(web_techs))
print('Countries:', countries)
print('Number of countries:', len(countries))
```

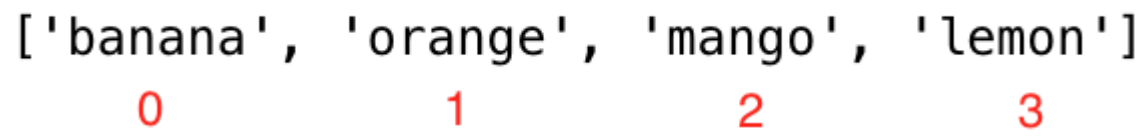
```
output
Fruits: ['banana', 'orange', 'mango', 'lemon']
Number of fruits: 4
Vegetables: ['Tomato', 'Potato', 'Cabbage', 'Onion', 'Carrot']
Number of vegetables: 5
Animal products: ['milk', 'meat', 'butter', 'yoghurt']
Number of animal products: 4
Web technologies: ['HTML', 'CSS', 'JS', 'React', 'Redux',
'Node', 'MongDB']
Number of web technologies: 7
Countries: ['Finland', 'Estonia', 'Denmark', 'Sweden',
'Norway']
Number of countries: 5
```

- Lists can have items of different data types

```
 lst = ['Asabeneh', 250, True, {'country':'Finland',
'city':'Helsinki'}] # list containing different data types
```

Accessing List Items Using Positive Indexing

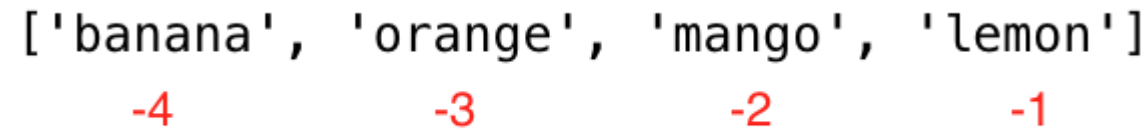
We access each item in a list using their index. A list index starts from 0. The picture below shows clearly where the index
starts




```
fruits = ['banana', 'orange', 'mango', 'lemon']
first_fruit = fruits[0] # we are accessing the first item
using its index
print(first_fruit)      # banana
second_fruit = fruits[1]
print(second_fruit)     # orange
last_fruit = fruits[3]
print(last_fruit) # lemon
# Last index
last_index = len(fruits) - 1
last_fruit = fruits[last_index]
```

Accessing List Items Using Negative Indexing

Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second last item.




```
fruits = ['banana', 'orange', 'mango', 'lemon']
first_fruit = fruits[-4]
last_fruit = fruits[-1]
second_last = fruits[-2]
print(first_fruit)      # banana
print(last_fruit)       # lemon
print(second_last)      # mango
```

Unpacking List Items

```
lst = ['item1','item2','item3', 'item4', 'item5']
first_item, second_item, third_item, *rest = lst
print(first_item)     # item1
print(second_item)    # item2
print(third_item)     # item3
print(rest)           # ['item4', 'item5']
```

```
# First Example
fruits = ['banana', 'orange', 'mango', 'lemon','lime','apple']
first_fruit, second_fruit, third_fruit, *rest = fruits
print(first_fruit)     # banana
print(second_fruit)    # orange
print(third_fruit)     # mango
print(rest)           # ['lemon','lime','apple']
# Second Example about unpacking list
first, second, third,*rest, tenth = [1,2,3,4,5,6,7,8,9,10]
print(first)          # 1
print(second)         # 2
print(third)          # 3
print(rest)           # [4,5,6,7,8,9]
print(tenth)          # 10
# Third Example about unpacking list
countries = ['Germany',
'France','Belgium','Sweden','Denmark','Finland','Norway','Icel
and','Estonia']
gr, fr, bg, sw, *scandic, es = countries
print(gr)
print(fr)
print(bg)
print(sw)
print(scandic)
print(es)
```

Slicing Items from a List

- Positive Indexing: We can specify a range of positive indexes by specifying the start, end and step, the return value will be a new list. (default values for start = 0, end = len(lst) - 1 (last item), step = 1)

```
fruits = ['banana', 'orange', 'mango', 'lemon']
all_fruits = fruits[0:4] # it returns all the fruits
# this will also give the same result as the one above
all_fruits = fruits[0:] # if we don't set where to stop it
takes all the rest
orange_and_mango = fruits[1:3] # it does not include the first
index
orange_mango_lemon = fruits[1:]
```

```
orange_and_lemon = fruits[::2] # here we used a 3rd argument,
step. It will take every 2cnd item - ['banana', 'mango']
```

- Negative Indexing: We can specify a range of negative indexes by specifying the start, end and step, the return value will be a new list.

```
fruits = ['banana', 'orange', 'mango', 'lemon']
all_fruits = fruits[-4:] # it returns all the fruits
orange_and_mango = fruits[-3:-1] # it does not include the
last index,['orange', 'mango']
orange_mango_lemon = fruits[-3:] # this will give starting
from -3 to the end,['orange', 'mango', 'lemon']
reverse_fruits = fruits[::-1] # a negative step will take the
list in reverse order,['lemon', 'mango', 'orange', 'banana']
```

Modifying Lists

List is a mutable or modifiable ordered collection of items. Lets modify the fruit list.

```
fruits = ['banana', 'orange', 'mango', 'lemon']
fruits[0] = 'avocado'
print(fruits)       #  ['avocado', 'orange', 'mango', 'lemon']
fruits[1] = 'apple'
print(fruits)       #  ['avocado', 'apple', 'mango', 'lemon']
last_index = len(fruits) - 1
fruits[last_index] = 'lime'
print(fruits)        #  ['avocado', 'apple', 'mango', 'lime']
```

Checking Items in a List

Checking an item if it is a member of a list using *in* operator. See the example below.

```
fruits = ['banana', 'orange', 'mango', 'lemon']
does_exist = 'banana' in fruits
print(does_exist)  # True
does_exist = 'lime' in fruits
print(does_exist)  # False
```

Adding Items to a List

To add item to the end of an existing list we use the method *append()*.

```
# syntax
lst = list()
lst.append(item)

fruits = ['banana', 'orange', 'mango', 'lemon']
fruits.append('apple')
print(fruits)           # ['banana', 'orange', 'mango',
'lemon', 'apple']
fruits.append('lime')   # ['banana', 'orange', 'mango',
'lemon', 'apple', 'lime']
```

```
print(fruits)
```

Inserting Items into a List

We can use *insert()* method to insert a single item at a specified index in a list. Note that other items are shifted to the right. The *insert()* methods takes two arguments:index and an item to insert.

```
# syntax
lst = ['item1', 'item2']
lst.insert(index, item)
```

```
fruits = ['banana', 'orange', 'mango', 'lemon']
fruits.insert(2, 'apple') # insert apple between orange and
mango
print(fruits)           # ['banana', 'orange', 'apple',
'mango', 'lemon']
fruits.insert(3, 'lime')   # ['banana', 'orange', 'apple',
'lime', 'mango', 'lemon']
print(fruits)
```

Removing Items from a List

The remove method removes a specified item from a list

```
# syntax
lst = ['item1', 'item2']
lst.remove(item)
```

```
fruits = ['banana', 'orange', 'mango', 'lemon', 'banana']
fruits.remove('banana')
print(fruits)  # ['orange', 'mango', 'lemon', 'banana'] - this
method removes the first occurrence of the item in the list
fruits.remove('lemon')
print(fruits)  # ['orange', 'mango', 'banana']
```

Removing Items Using Pop

The *pop()* method removes the specified index, (or the last item if index is not specified):

```
# syntax
lst = ['item1', 'item2']
lst.pop()       # last item
lst.pop(index)
```

```
fruits = ['banana', 'orange', 'mango', 'lemon']
fruits.pop()
print(fruits)       # ['banana', 'orange', 'mango']

fruits.pop(0)
print(fruits)       # ['orange', 'mango']
```

Removing Items Using Del

The *del* keyword removes the specified index and it can also be used to delete items within index range. It can also delete the list completely

```
# syntax
lst = ['item1', 'item2']
del lst[index] # only a single item
del lst        # to delete the list completely
```

```
fruits = ['banana', 'orange', 'mango', 'lemon', 'kiwi',
'lime']
del fruits[0]
print(fruits)       # ['orange', 'mango', 'lemon', 'kiwi',
'lime']
del fruits[1]
print(fruits)       # ['orange', 'lemon', 'kiwi', 'lime']
del fruits[1:3]     # this deletes items between given
indexes, so it does not delete the item with index 3!
print(fruits)       # ['orange', 'lime']
del fruits
print(fruits)       # This should give: NameError: name
'fruits' is not defined
```

Clearing List Items

The *clear()* method empties the list:

```
# syntax
lst = ['item1', 'item2']
lst.clear()

fruits = ['banana', 'orange', 'mango', 'lemon']
fruits.clear()
print(fruits)       # []
```

Copying a List

It is possible to copy a list by reassigning it to a new variable in the following way: list2 = list1. Now, list2 is a reference of list1, any changes we make in list2 will also modify the original, list1. But there are lots of case in which we do not like to modify

the original instead we like to have a different copy. One of way of avoiding the problem above is using *copy()*.

```
# syntax
lst = ['item1', 'item2']
lst_copy = lst.copy()
```

```
fruits = ['banana', 'orange', 'mango', 'lemon']
fruits_copy = fruits.copy()
print(fruits_copy)       # ['banana', 'orange', 'mango',
'lemon']
```

Joining Lists

There are several ways to join, or concatenate, two or more lists in Python.

- Plus Operator (+)

```
# syntax
list3 = list1 + list2
```

```
positive_numbers = [1, 2, 3, 4, 5]
zero = [0]
negative_numbers = [-5,-4,-3,-2,-1]
integers = negative_numbers + zero + positive_numbers
print(integers) # [-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5]
fruits = ['banana', 'orange', 'mango', 'lemon']
vegetables = ['Tomato', 'Potato', 'Cabbage', 'Onion',
'Carrot']
fruits_and_vegetables = fruits + vegetables
print(fruits_and_vegetables ) # ['banana', 'orange', 'mango',
'lemon', 'Tomato', 'Potato', 'Cabbage', 'Onion', 'Carrot']
```

- Joining using extend() method The *extend()* method allows to append list in a list. See the example below.

```
# syntax
list1 = ['item1', 'item2']
list2 = ['item3', 'item4', 'item5']
list1.extend(list2)
```

```
num1 = [0, 1, 2, 3]
num2= [4, 5, 6]
num1.extend(num2)
print('Numbers:', num1) # Numbers: [0, 1, 2, 3, 4, 5, 6]
negative_numbers = [-5,-4,-3,-2,-1]
positive_numbers = [1, 2, 3,4,5]
```

```
zero = [0]

negative_numbers.extend(zero)
negative_numbers.extend(positive_numbers)
print('Integers:', negative_numbers) # Integers: [-5, -4, -3,
-2, -1, 0, 1, 2, 3, 4, 5]
fruits = ['banana', 'orange', 'mango', 'lemon']
vegetables = ['Tomato', 'Potato', 'Cabbage', 'Onion',
'Carrot']
fruits.extend(vegetables)
print('Fruits and vegetables:', fruits ) # Fruits and
vegetables: ['banana', 'orange', 'mango', 'lemon', 'Tomato',
'Potato', 'Cabbage', 'Onion', 'Carrot']
```

Counting Items in a List

The *count()* method returns the number of times an item appears in a list:

```
# syntax
lst = ['item1', 'item2']
lst.count(item)
```

```
fruits = ['banana', 'orange', 'mango', 'lemon']
print(fruits.count('orange'))   # 1
ages = [22, 19, 24, 25, 26, 24, 25, 24]
print(ages.count(24))           # 3
```

Finding Index of an Item

The *index()* method returns the index of an item in the list:

```
# syntax
lst = ['item1', 'item2']
lst.index(item)
```

```
fruits = ['banana', 'orange', 'mango', 'lemon']
print(fruits.index('orange'))   # 1
ages = [22, 19, 24, 25, 26, 24, 25, 24]
print(ages.index(24))           # 2, the first occurrence
```

Reversing a List

The *reverse()* method reverses the order of a list.

```
# syntax
lst = ['item1', 'item2']
lst.reverse()
```

```
fruits = ['banana', 'orange', 'mango', 'lemon']
fruits.reverse()
print(fruits) # ['lemon', 'mango', 'orange', 'banana']
```

```
ages = [22, 19, 24, 25, 26, 24, 25, 24]
ages.reverse()
print(ages) # [24, 25, 24, 26, 25, 24, 19, 22]
```

Sorting List Items

To sort lists we can use *sort()* method or *sorted()* built-in functions. The *sort()* method reorders the list items in ascending order and modifies the original list. If an argument of *sort()* method reverse is equal to true, it will arrange the list in descending order.

- sort(): this method modifies the original list

```
# syntax
lst = ['item1', 'item2']
lst.sort()                # ascending
lst.sort(reverse=True)    # descending
```

Example:

```
fruits = ['banana', 'orange', 'mango', 'lemon']
fruits.sort()
print(fruits)             # sorted in alphabetical order,
['banana', 'lemon', 'mango', 'orange']
fruits.sort(reverse=True)
print(fruits) # ['orange', 'mango', 'lemon', 'banana']
ages = [22, 19, 24, 25, 26, 24, 25, 24]
ages.sort()
print(ages) #  [19, 22, 24, 24, 24, 25, 25, 26]

ages.sort(reverse=True)
print(ages) #  [26, 25, 25, 24, 24, 24, 22, 19]
```

sorted(): returns the ordered list without modifying the original list **Example:**

```
fruits = ['banana', 'orange', 'mango', 'lemon']
print(sorted(fruits))   # ['banana', 'lemon', 'mango',
'orange']
# Reverse order
fruits = ['banana', 'orange', 'mango', 'lemon']
fruits = sorted(fruits,reverse=True)
print(fruits)     # ['orange', 'mango', 'lemon', 'banana']
```

You are diligent and you have already achieved quite a lot. You have just completed day 5 challenges and you are 5 steps a head in to your way to greatness. Now do some exercises for your brain and muscles.

💻 Exercises: Day 5

Exercises: Level 1

1. Declare an empty list
2. Declare a list with more than 5 items
3. Find the length of your list
4. Get the first item, the middle item and the last item of the list
5. Declare a list called mixed_data_types, put your(name, age, height, marital status, address)
6. Declare a list variable named it_companies and assign initial values Facebook, Google, Microsoft, Apple, IBM, Oracle and Amazon.
7. Print the list using *print()*
8. Print the number of companies in the list
9. Print the first, middle and last company
10. Print the list after modifying one of the companies
11. Add an IT company to it_companies
12. Insert an IT company in the middle of the companies list
13. Change one of the it_companies names to uppercase (IBM excluded!)
14. Join the it_companies with a string '#; '
15. Check if a certain company exists in the it_companies list.
16. Sort the list using sort() method
17. Reverse the list in descending order using reverse() method
18. Slice out the first 3 companies from the list
19. Slice out the last 3 companies from the list
20. Slice out the middle IT company or companies from the list
21. Remove the first IT company from the list
22. Remove the middle IT company or companies from the list
23. Remove the last IT company from the list
24. Remove all IT companies from the list
25. Destroy the IT companies list
26. Join the following lists:

```
front_end = ['HTML', 'CSS', 'JS', 'React', 'Redux']
back_end = ['Node','Express', 'MongoDB']
```

27. After joining the lists in question 26. Copy the joined list and assign it to a variable full_stack, then insert Python and SQL after Redux.

Exercises: Level 2

1. The following is a list of 10 students ages:

```
ages = [19, 22, 19, 24, 20, 25, 26, 24, 25, 24]
```

- Sort the list and find the min and max age
- Add the min age and the max age again to the list
- Find the median age (one middle item or two middle items divided by two)
- Find the average age (sum of all items divided by their number)
- Find the range of the ages (max minus min)
- Compare the value of (min - average) and (max - average), use *abs()* method

1. Find the middle country(ies) in the countries list
2. Divide the countries list into two equal lists if it is even if not one more country for the first half.
3. ['China', 'Russia', 'USA', 'Finland', 'Sweden', 'Norway', 'Denmark']. Unpack the first three countries and the rest as scandic countries.

🎉 CONGRATULATIONS ! 🎉

DAY-6 TUPLES

Tuples

A tuple is a collection of different data types which is ordered and unchangeable (immutable). Tuples are written with round brackets, (). Once a tuple is created, we cannot change its values. We cannot use add, insert, remove methods in a tuple because it is not modifiable (mutable). Unlike list, tuple has few methods. Methods related to tuples:

- tuple(): to create an empty tuple
- count(): to count the number of a specified item in a tuple
- index(): to find the index of a specified item in a tuple
- operator: to join two or more tuples and to create a new tuple

Creating a Tuple

- Empty tuple: Creating an empty tuple

```
# syntax
empty_tuple = ()
# or using the tuple constructor
empty_tuple = tuple()
```

- Tuple with initial values

```
# syntax
tpl = ('item1', 'item2','item3')
fruits = ('banana', 'orange', 'mango', 'lemon')
```

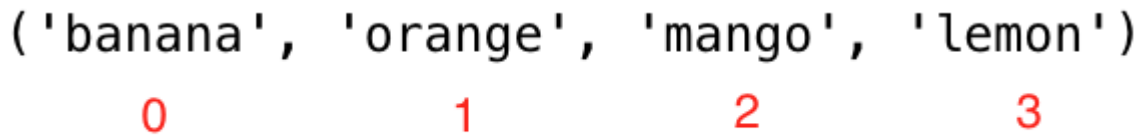
Tuple length

We use the *len()* method to get the length of a tuple.

```
# syntax
tpl = ('item1', 'item2', 'item3')
len(tpl)
```

Accessing Tuple Items

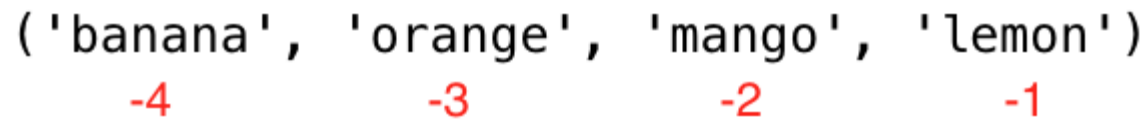
- Positive Indexing Similar to the list data type we use positive or negative indexing to access tuple
  items.




```
# Syntax
tpl = ('item1', 'item2', 'item3')
first_item = tpl[0]
second_item = tpl[1]

fruits = ('banana', 'orange', 'mango', 'lemon')
first_fruit = fruits[0]
second_fruit = fruits[1]
last_index =len(fruits) - 1
last_fruit = fruits[las_index]
```

- Negative indexing Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second last and the negative of the list/tuple length refers to the first
  item.




```
# Syntax
tpl = ('item1', 'item2', 'item3','item4')
first_item = tpl[-4]
second_item = tpl[-3]

fruits = ('banana', 'orange', 'mango', 'lemon')
first_fruit = fruits[-4]
second_fruit = fruits[-3]
last_fruit = fruits[-1]
```

Slicing tuples

We can slice out a sub-tuple by specifying a range of indexes where to start and where to end in the tuple, the return value will be a new tuple with the specified items.

- Range of Positive Indexes

```
# Syntax
tpl = ('item1', 'item2', 'item3','item4')
all_items = tpl[0:4]          # all items
all_items = tpl[0:]          # all items
middle_two_items = tpl[1:3]  # does not include item at index
3
```

```
fruits = ('banana', 'orange', 'mango', 'lemon')
all_fruits = fruits[0:4]    # all items
all_fruits= fruits[0:]      # all items
orange_mango = fruits[1:3]  # doesn't include item at index 3
orange_to_the_rest = fruits[1:]
```

- Range of Negative Indexes

```
# Syntax
tpl = ('item1', 'item2', 'item3','item4')
all_items = tpl[-4:]          # all items
middle_two_items = tpl[-3:-1]  # does not include item at
index 3 (-1)

fruits = ('banana', 'orange', 'mango', 'lemon')
all_fruits = fruits[-4:]    # all items
orange_mango = fruits[-3:-1]  # doesn't include item at index
3
orange_to_the_rest = fruits[-3:]
```

Changing Tuples to Lists

We can change tuples to lists and lists to tuples. Tuple is immutable if we want to modify a tuple we should change it to a list.

```
# Syntax
tpl = ('item1', 'item2', 'item3','item4')
lst = list(tpl)
```

```
fruits = ('banana', 'orange', 'mango', 'lemon')
fruits = list(fruits)
fruits[0] = 'apple'
print(fruits)     # ['apple', 'orange', 'mango', 'lemon']
fruits = tuple(fruits)
print(fruits)     # ('apple', 'orange', 'mango', 'lemon')
```

Checking an Item in a Tuple

We can check if an item exists or not in a tuple using *in*, it returns a boolean.

```
# Syntax
tpl = ('item1', 'item2', 'item3','item4')
'item2' in tpl # True
```

```
fruits = ('banana', 'orange', 'mango', 'lemon')
print('orange' in fruits) # True
print('apple' in fruits) # False
fruits[0] = 'apple' # TypeError: 'tuple' object does not
support item assignment
```

Joining Tuples

We can join two or more tuples using + operator

```
# syntax
tpl1 = ('item1', 'item2', 'item3')
tpl2 = ('item4', 'item5','item6')
tpl3 = tpl1 + tpl2
```

```
fruits = ('banana', 'orange', 'mango', 'lemon')
vegetables = ('Tomato', 'Potato', 'Cabbage','Onion', 'Carrot')
fruits_and_vegetables = fruits + vegetables
```

Deleting Tuples

It is not possible to remove a single item in a tuple but it is possible to delete the tuple itself using *del*.

```
# syntax
tpl1 = ('item1', 'item2', 'item3')
del tpl1

fruits = ('banana', 'orange', 'mango', 'lemon')
del fruits
```

You are so brave, you made it to this far. You have just completed day 6 challenges and you are 6 steps a head in to your way to greatness. Now do some exercises for your brain and for your muscle.

💻 Exercises: Day 6

Exercises: Level 1

1. Create an empty tuple
2. Create a tuple containing names of your sisters and your brothers (imaginary siblings are fine)
3. Join brothers and sisters tuples and assign it to siblings
4. How many siblings do you have?
5. Modify the siblings tuple and add the name of your father and mother and assign it to family_members

Exercises: Level 2

1. Unpack siblings and parents from family_members
2. Create fruits, vegetables and animal products tuples. Join the three tuples and assign it to a variable called food_stuff_tp.
3. Change the about food_stuff_tp tuple to a food_stuff_lt list
4. Slice out the middle item or items from the food_stuff_tp tuple or food_stuff_lt list.
5. Slice out the first three items and the last three items from food_staff_lt list
6. Delete the food_staff_tp tuple completely
7. Check if an item exists in tuple:

- Check if 'Estonia' is a nordic country
- Check if 'Iceland' is a nordic country

```
nordic_countries = ('Denmark', 'Finland','Iceland', 'Norway', 'Sweden')
```

DAY-7 SETS

Sets

Set is a collection of items. Let me take you back to your elementary or high school Mathematics lesson. The Mathematics definition of a set can be applied also in Python. Set is a collection of unordered and un-indexed distinct elements. In Python set is used to store unique items, and it is possible to find the *union*, *intersection*, *difference*, *symmetric difference*, *subset*, *super set* and *disjoint set* among sets.

Creating a Set

We use the *set()* built-in function.

- Creating an empty set

```
# syntax
st = set()
```

- Creating a set with initial items

```
# syntax
st = {'item1', 'item2', 'item3', 'item4'}
```

Example:

```
# syntax
fruits = {'banana', 'orange', 'mango', 'lemon'}
```

Getting Set's Length

We use **len()** method to find the length of a set.

```
# syntax
st = {'item1', 'item2', 'item3', 'item4'}
len(st)
```

Example:

```
fruits = {'banana', 'orange', 'mango', 'lemon'}
len(fruits)
```

Accessing Items in a Set

We use loops to access items. We will see this in loop section

Checking an Item

To check if an item exist in a list we use *in* membership operator.

```
# syntax
st = {'item1', 'item2', 'item3', 'item4'}
```

```
print("Does set st contain item3? ", 'item3' in st) # Does set st contain item3? True
```

Example:

```
fruits = {'banana', 'orange', 'mango', 'lemon'}
print('mango' in fruits ) # True
```

Adding Items to a Set

Once a set is created we cannot change any items and we can also add additional items.

- Add one item using *add()*

```
# syntax
st = {'item1', 'item2', 'item3', 'item4'}
st.add('item5')
```

Example:

```
fruits = {'banana', 'orange', 'mango', 'lemon'}
fruits.add('lime')
```

- Add multiple items using *update()* The *update()* allows to add multiple items to a set. The *update()* takes a list argument.

```
# syntax
st = {'item1', 'item2', 'item3', 'item4'}
st.update(['item5','item6','item7'])
```

Example:

```
fruits = {'banana', 'orange', 'mango', 'lemon'}
vegetables = ('tomato', 'potato', 'cabbage','onion', 'carrot')
fruits.update(vegetables)
```

Removing Items from a Set

We can remove an item from a set using *remove()* method. If the item is not found *remove()* method will raise errors, so it is good to check if the item exist in the given set. However, *discard()* method doesn't raise any errors.

```
# syntax
st = {'item1', 'item2', 'item3', 'item4'}
st.remove('item2')
```

The pop() methods remove a random item from a list and it returns the removed item.

Example:

```
fruits = {'banana', 'orange', 'mango', 'lemon'}
fruits.pop()  # removes a random item from the set
```

If we are interested in the removed item.

```
fruits = {'banana', 'orange', 'mango', 'lemon'}
removed_item = fruits.pop()
```

Clearing Items in a Set

If we want to clear or empty the set we use *clear* method.

```
# syntax
st = {'item1', 'item2', 'item3', 'item4'}
```

st.clear()

Example:

```
fruits = {'banana', 'orange', 'mango', 'lemon'}
fruits.clear()
print(fruits) # set()
```

Deleting a Set

If we want to delete the set itself we use *del* operator.

```
# syntax
st = {'item1', 'item2', 'item3', 'item4'}
del st
```

Example:

```
fruits = {'banana', 'orange', 'mango', 'lemon'}
del fruits
```

Converting List to Set

We can convert list to set and set to list. Converting list to set removes duplicates and only unique items will be reserved.

```
# syntax
lst = ['item1', 'item2', 'item3', 'item4', 'item1']
st = set(lst)  # {'item2', 'item4', 'item1', 'item3'} - the
order is random, because sets in general are unordered
```

Example:

```
fruits = ['banana', 'orange', 'mango', 'lemon','orange',
'banana']
fruits = set(fruits) # {'mango', 'lemon', 'banana', 'orange'}
```

Joining Sets

We can join two sets using the *union()* or *update()* method.

- Union This method returns a new set

```
# syntax
st1 = {'item1', 'item2', 'item3', 'item4'}
st2 = {'item5', 'item6', 'item7', 'item8'}
st3 = st1.union(st2)
```

Example:

```
fruits = {'banana', 'orange', 'mango', 'lemon'}
vegetables = {'tomato', 'potato', 'cabbage','onion', 'carrot'}
print(fruits.union(vegetables)) # {'lemon', 'carrot',
'tomato', 'banana', 'mango', 'orange', 'cabbage', 'potato',
'onion'}
```

- Update This method inserts a set into a given set

```
# syntax
st1 = {'item1', 'item2', 'item3', 'item4'}
st2 = {'item5', 'item6', 'item7', 'item8'}
st1.update(st2) # st2 contents are added to st1
```

Example:

```
fruits = {'banana', 'orange', 'mango', 'lemon'}
vegetables = {'tomato', 'potato', 'cabbage','onion', 'carrot'}
fruits.update(vegetables)
print(fruits) # {'lemon', 'carrot', 'tomato', 'banana',
'mango', 'orange', 'cabbage', 'potato', 'onion'}
```

Finding Intersection Items

Intersection returns a set of items which are in both the sets. See the example

```
# syntax
st1 = {'item1', 'item2', 'item3', 'item4'}
st2 = {'item3', 'item2'}
st1.intersection(st2) # {'item3', 'item2'}
```

Example:

```
whole_numbers = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
even_numbers = {0, 2, 4, 6, 8, 10}
whole_numbers.intersection(even_numbers) # {0, 2, 4, 6, 8, 10}

python = {'p', 'y', 't', 'h', 'o','n'}
dragon = {'d', 'r', 'a', 'g', 'o','n'}
python.intersection(dragon)     # {'o', 'n'}
```

Checking Subset and Super Set

A set can be a subset or super set of other sets:

- Subset: *issubset()*
- Super set: *issuperset*

```
# syntax
st1 = {'item1', 'item2', 'item3', 'item4'}
st2 = {'item2', 'item3'}
st2.issubset(st1) # True
st1.issuperset(st2) # True
```

Example:

```
whole_numbers = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
even_numbers = {0, 2, 4, 6, 8, 10}
whole_numbers.issubset(even_numbers) # False, because it is a
super set
whole_numbers.issuperset(even_numbers) # True

python = {'p', 'y', 't', 'h', 'o','n'}
dragon = {'d', 'r', 'a', 'g', 'o','n'}
python.issubset(dragon)     # False
```

Checking the Difference Between Two Sets

It returns the difference between two sets.

```
# syntax
st1 = {'item1', 'item2', 'item3', 'item4'}
st2 = {'item2', 'item3'}
st2.difference(st1) # set()
st1.difference(st2) # {'item1', 'item4'} => st1\st2
```

Example:

```
whole_numbers = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
even_numbers = {0, 2, 4, 6, 8, 10}
whole_numbers.difference(even_numbers) # {1, 3, 5, 7, 9}

python = {'p', 'y', 't', 'o','n'}
dragon = {'d', 'r', 'a', 'g', 'o','n'}
python.difference(dragon)     # {'p', 'y', 't'}  - the result
is unordered (characteristic of sets)
dragon.difference(python)     # {'d', 'r', 'a', 'g'}
```

Finding Symmetric Difference Between Two Sets

It returns the symmetric difference between two sets. It means that it returns a set that contains all items from both sets, except items that are present in both sets, mathematically: (A\B) ∪ (B\A)

```
# syntax
st1 = {'item1', 'item2', 'item3', 'item4'}
st2 = {'item2', 'item3'}
# it means (A\B)∪(B\A)
st2.symmetric_difference(st1) # {'item1', 'item4'}
```

Example:

```
whole_numbers = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
some_numbers = {1, 2, 3, 4, 5}
whole_numbers.symmetric_difference(some_numbers) # {0, 6, 7,
8, 9, 10}

python = {'p', 'y', 't', 'h', 'o','n'}
dragon = {'d', 'r', 'a', 'g', 'o','n'}
python.symmetric_difference(dragon)  # {'r', 't', 'p', 'y',
'g', 'a', 'd', 'h'}
```

Joining Sets

If two sets do not have a common item or items we call them disjoint sets. We can check if two sets are joint or disjoint using *isdisjoint()* method.

```
# syntax
st1 = {'item1', 'item2', 'item3', 'item4'}
st2 = {'item2', 'item3'}
st2.isdisjoint(st1) # False
```

Example:

```
even_numbers = {0, 2, 4 ,6, 8}
odd_numbers = {1, 3, 5, 7, 9}
even_numbers.isdisjoint(odd_numbers) # True, because no common
item

python = {'p', 'y', 't', 'h', 'o','n'}
dragon = {'d', 'r', 'a', 'g', 'o','n'}
python.isdisjoint(dragon)  # False, there are common items
{'o', 'n'}
```

🎱 You are a rising star . You have just completed day 7 challenges and you are 7 steps ahead in to your way to greatness. Now do some exercises for your brain and muscles.

💻 Exercises: Day 7

```
# sets

it_companies = {'Facebook', 'Google', 'Microsoft', 'Apple', 'IBM', 'Oracle', 'Amazon'}

A = {19, 22, 24, 20, 25, 26}

B = {19, 22, 20, 25, 26, 24, 28, 27}

age = [22, 19, 24, 25, 26, 24, 25, 24]
```

Exercises: Level 1

1. Find the length of the set it_companies
2. Add 'Twitter' to it_companies
3. Insert multiple IT companies at once to the set it_companies
4. Remove one of the companies from the set it_companies
5. What is the difference between remove and discard

Exercises: Level 2

1. Join A and B
2. Find A intersection B
3. Is A subset of B
4. Are A and B disjoint sets
5. Join A with B and B with A
6. What is the symmetric difference between A and B
7. Delete the sets completely

Exercises: Level 3

1. Convert the ages to a set and compare the length of the list and the set, which one is bigger?
2. Explain the difference between the following data types: string, list, tuple and set
3. *I am a teacher and I love to inspire and teach people.* How many unique words have been used in the sentence? Use the split methods and set to get the unique words.

🎉 CONGRATULATIONS ! 🎉

DAY-8 DICTIONARIES

Dictionaries

A dictionary is a collection of unordered, modifiable(mutable) paired (key: value) data type.

Creating a Dictionary

To create a dictionary we use curly brackets, {} or the *dict()* built-in function.

```
# syntax
empty_dict = {}
# Dictionary with data values
dct = {'key1':'value1', 'key2':'value2', 'key3':'value3',
'key4':'value4'}
```

Example:

```
person = {
    'first_name':'Asabeneh',
    'last_name':'Yetayeh',
    'age':250,
    'country':'Finland',
    'is_marred':True,
    'skills':['JavaScript', 'React', 'Node', 'MongoDB',
'Python'],
    'address':{
        'street':'Space street',
        'zipcode':'02210'
    }
    }
```

The dictionary above shows that a value could be any data types:string, boolean, list, tuple, set or a dictionary.

Dictionary Length

It checks the number of 'key: value' pairs in the dictionary.

```
# syntax
dct = {'key1':'value1', 'key2':'value2', 'key3':'value3',
'key4':'value4'}
print(len(dct)) # 4
```

Example:

```
person = {
    'first_name':'Asabeneh',
    'last_name':'Yetayeh',
    'age':250,
    'country':'Finland',
```

```
    'is_married':True,
    'skills':['JavaScript', 'React', 'Node', 'MongoDB',
'Python'],
    'address':{
        'street':'Space street',
        'zipcode':'02210'
    }
    }
print(len(person)) # 7
```

Accessing Dictionary Items

We can access Dictionary items by referring to its key name.

```
# syntax
dct = {'key1':'value1', 'key2':'value2', 'key3':'value3',
'key4':'value4'}
print(dct['key1']) # value1
print(dct['key4']) # value4
```

Example:

```
person = {
    'first_name':'Asabeneh',
    'last_name':'Yetayeh',
    'age':250,
    'country':'Finland',
    'is_marred':True,
    'skills':['JavaScript', 'React', 'Node', 'MongoDB',
'Python'],
    'address':{
        'street':'Space street',
        'zipcode':'02210'
    }
    }
print(person['first_name']) # Asabeneh
print(person['country'])    # Finland
print(person['skills'])     # ['JavaScript', 'React', 'Node',
'MongoDB', 'Python']
print(person['skills'][0])  # JavaScript
print(person['address']['street']) # Space street
print(person['city'])       # Error
```

Accessing an item by key name raises an error if the key does not exist. To avoid this error first we have to check if a key exist or we can use the *get* method. The get method returns None, which is a NoneType object data type, if the key does not exist.

```
person = {
    'first_name':'Asabeneh',
    'last_name':'Yetayeh',
```

```
    'age':250,
    'country':'Finland',
    'is_marred':True,
    'skills':['JavaScript', 'React', 'Node', 'MongoDB',
'Python'],
    'address':{
        'street':'Space street',
        'zipcode':'02210'
    }
    }
print(person.get('first_name')) # Asabeneh
print(person.get('country'))    # Finland
print(person.get('skills')) #['HTML','CSS','JavaScript',
'React', 'Node', 'MongoDB', 'Python']
print(person.get('city'))   # None
```

Adding Items to a Dictionary

We can add new key and value pairs to a dictionary

```
# syntax
dct = {'key1':'value1', 'key2':'value2', 'key3':'value3',
'key4':'value4'}
dct['key5'] = 'value5'
```

Example:

```
person = {
    'first_name':'Asabeneh',
    'last_name':'Yetayeh',
    'age':250,
    'country':'Finland',
    'is_marred':True,
    'skills':['JavaScript', 'React', 'Node', 'MongoDB',
'Python'],
    'address':{
        'street':'Space street',
        'zipcode':'02210'
        }
}
person['job_title'] = 'Instructor'
person['skills'].append('HTML')
print(person)
```

Modifying Items in a Dictionary

We can modify items in a dictionary

```
# syntax
```

```
dct = {'key1':'value1', 'key2':'value2', 'key3':'value3',
'key4':'value4'}
dct['key1'] = 'value-one'
```

Example:

```
person = {
    'first_name':'Asabeneh',
    'last_name':'Yetayeh',
    'age':250,
    'country':'Finland',
    'is_marred':True,
    'skills':['JavaScript', 'React', 'Node', 'MongoDB',
'Python'],
    'address':{
        'street':'Space street',
        'zipcode':'02210'
    }
    }
person['first_name'] = 'Eyob'
person['age'] = 252
```

Checking Keys in a Dictionary

We use the *in* operator to check if a key exist in a dictionary

```
# syntax
dct = {'key1':'value1', 'key2':'value2', 'key3':'value3',
'key4':'value4'}
print('key2' in dct) # True
print('key5' in dct) # False
```

Removing Key and Value Pairs from a Dictionary

- *pop(key)*: removes the item with the specified key name:
- *popitem()*: removes the last item
- *del*: removes an item with specified key name

```
# syntax
dct = {'key1':'value1', 'key2':'value2', 'key3':'value3',
'key4':'value4'}
dct.pop('key1') # removes key1 item
dct = {'key1':'value1', 'key2':'value2', 'key3':'value3',
'key4':'value4'}
dct.popitem() # removes the last item
del dct['key2'] # removes key2 item
```

Example:

```
person = {
    'first_name':'Asabeneh',
    'last_name':'Yetayeh',
```

```
    'age':250,
    'country':'Finland',
    'is_marred':True,
    'skills':['JavaScript', 'React', 'Node', 'MongoDB',
'Python'],
    'address':{
        'street':'Space street',
        'zipcode':'02210'
    }
    }
person.pop('first_name')        # Removes the firstname item
person.popitem()                # Removes the address item
del person['is_married']        # Removes the is_married item
```

Changing Dictionary to a List of Items

The *items()* method changes dictionary to a list of tuples.

```
# syntax
dct = {'key1':'value1', 'key2':'value2', 'key3':'value3',
'key4':'value4'}
print(dct.items()) # dict_items([('key1', 'value1'), ('key2',
'value2'), ('key3', 'value3'), ('key4', 'value4')])
```

Clearing a Dictionary

If we don't want the items in a dictionary we can clear them using *clear()* method

```
# syntax
dct = {'key1':'value1', 'key2':'value2', 'key3':'value3',
'key4':'value4'}
print(dct.clear()) # None
```

Deleting a Dictionary

If we do not use the dictionary we can delete it completely

```
# syntax
dct = {'key1':'value1', 'key2':'value2', 'key3':'value3',
'key4':'value4'}
del dct
```

Copy a Dictionary

We can copy a dictionary using a *copy()* method. Using copy we can avoid mutation of the original dictionary.

```
# syntax
dct = {'key1':'value1', 'key2':'value2', 'key3':'value3',
'key4':'value4'}
dct_copy = dct.copy() # {'key1':'value1', 'key2':'value2',
'key3':'value3', 'key4':'value4'}
```

Getting Dictionary Keys as a List

The *keys()* method gives us all the keys of a a dictionary as a list.

```
# syntax
dct = {'key1':'value1', 'key2':'value2', 'key3':'value3',
'key4':'value4'}
keys = dct.keys()
print(keys)     # dict_keys(['key1', 'key2', 'key3', 'key4'])
```

Getting Dictionary Values as a List

The *values* method gives us all the values of a a dictionary as a list.

```
# syntax
dct = {'key1':'value1', 'key2':'value2', 'key3':'value3',
'key4':'value4'}
values = dct.values()
print(values)     # dict_values(['value1', 'value2', 'value3',
'value4'])
```

🎉 You are astonishing. Now, you are super charged with the power of dictionaries. You have just completed day 8 challenges and you are 8 steps a head in to your way to greatness. Now do some exercises for your brain and muscles.

💻 Exercises: Day 8

1. Create an empty dictionary called dog
2. Add name, color, breed, legs, age to the dog dictionary
3. Create a student dictionary and add first_name, last_name, gender, age, marital status, skills, country, city and address as keys for the dictionary
4. Get the length of the student dictionary
5. Get the value of skills and check the data type, it should be a list
6. Modify the skills values by adding one or two skills
7. Get the dictionary keys as a list
8. Get the dictionary values as a list
9. Change the dictionary to a list of tuples using *items()* method
10. Delete one of the items in the dictionary
11. Delete one of the dictionaries

🎉 CONGRATULATIONS ! 🎉

DAY-9 CONDITIONALS

Conditionals

By default, statements in Python script are executed sequentially from top to bottom. If the processing logic require so, the sequential flow of execution can be altered in two way:

- Conditional execution: a block of one or more statements will be executed if a certain expression is true
- Repetitive execution: a block of one or more statements will be repetitively executed as long as a certain expression is true. In this section, we will cover *if*, *else*, *elif* statements. The comparison and logical operators we learned in previous sections will be useful here.

If Condition

In python and other programming languages the key word *if* is used to check if a condition is true and to execute the block code. Remember the indentation after the colon.

```
# syntax
if condition:
    this part of code runs for truthy conditions
```

Example: 1

```
a = 3
if a > 0:
    print('A is a positive number')
# A is a positive number
```

As you can see in the example above, 3 is greater than 0. The condition was true and the block code was executed. However, if the condition is false, we do not see the result. In order to see the result of the falsy condition, we should have another block, which is going to be *else*.

If Else

If condition is true the first block will be executed, if not the else condition will run.

```
# syntax
if condition:
    this part of code runs for truthy conditions
else:
     this part of code runs for false conditions
```

Example:

```
a = 3
if a < 0:
    print('A is a negative number')
else:
    print('A is a positive number')
```

The condition above proves false, therefore the else block was executed. How about if our condition is more than two? We could use *elif*.

If Elif Else

In our daily life, we make decisions on daily basis. We make decisions not by checking one or two conditions but multiple conditions. As similar to life, programming is also full of conditions. We use *elif* when we have multiple conditions.

```
# syntax
if condition:
    code
elif condition:
    code
else:
    code
```

Example:

```
a = 0
if a > 0:
    print('A is a positive number')
elif a < 0:
    print('A is a negative number')
else:
    print('A is zero')
```

Short Hand

```
# syntax
code if condition else code
```

Example:

```
a = 3
print('A is positive') if a > 0 else print('A is negative') #
first condition met, 'A is positive' will be printed
```

Nested Conditions

Conditions can be nested

```
# syntax
if condition:
    code
    if condition:
    code
```

Example:

```
a = 0
if a > 0:
    if a % 2 == 0:
        print('A is a positive and even integer')
    else:
        print('A is a positive number')
elif a == 0:
    print('A is zero')
else:
    print('A is a negative number')
```

We can avoid writing nested condition by using logical operator *and*.

If Condition and Logical Operators

```
# syntax
if condition and condition:
    code
```

Example:

```
a = 0
if a > 0 and a % 2 == 0:
        print('A is an even and positive integer')
elif a > 0 and a % 2 !=  0:
     print('A is a positive integer')
elif a == 0:
    print('A is zero')
else:
    print('A is negative')
```

If and Or Logical Operators

```
# syntax
if condition or condition:
    code
```

Example:

```
user = 'James'
access_level = 3
```

```
if user == 'admin' or access_level >= 4:
        print('Access granted!')
else:
    print('Access denied!')
```

⚽ You are doing great.Never give up because great things take time. You have just completed day 9 challenges and you are 9 steps a head in to your way to greatness. Now do some exercises for your brain and muscles.

💻 Exercises: Day 9

Exercises: Level 1

1. Get user input using input(“Enter your age: ”). If user is 18 or older, give feedback: You are old enough to drive. If below 18 give feedback to wait for the missing amount of years. Output:

```
Enter your age: 30
You are old enough to learn to drive.
Output:
Enter your age: 15
You need 3 more years to learn to drive.
```

2. Compare the values of my_age and your_age using if … else. Who is older (me or you)? Use input(“Enter your age: ”) to get the age as input. You can use a nested condition to print 'year' for 1 year difference in age, 'years' for bigger differences, and a custom text if my_age = your_age. Output:

```
Enter your age: 30
You are 5 years older than me.
```

3. Get two numbers from the user using input prompt. If a is greater than b return a is greater than b, if a is less b return a is smaller than b, else a is equal to b. Output:

```
Enter number one: 4
Enter number two: 3
4 is greater than 3
```

Exercises: Level 2

1. Write a code which gives grade to students according to theirs scores:

```
80-100, A
70-89, B
60-69, C
50-59, D
0-49, F
```

2. Check if the season is Autumn, Winter, Spring or Summer. If the user input is: September, October or November, the season is Autumn. December, January or February, the season is Winter. March, April or May, the season is Spring June, July or August, the season is Summer

3. The following list contains some fruits:

```
```sh
fruits = ['banana', 'orange', 'mango', 'lemon']
```
```

```
If a fruit doesn't exist in the list add the fruit to the list and print the modified list. If the fruit exists print('That fruit already exist in the list')
```

Exercises: Level 3

1. Here we have a person dictionary. Feel free to modify it!

```
        person={
    'first_name': 'Asabeneh',
    'last_name': 'Yetayeh',
    'age': 250,
    'country': 'Finland',
    'is_marred': True,
    'skills': ['JavaScript', 'React', 'Node', 'MongoDB', 'Python'],
    'address': {
        'street': 'Space street',
        'zipcode': '02210'
    }
    }
```

```
 * Check if the person dictionary has skills key, if so print out the middle skill in the skills list.
 * Check if the person dictionary has skills key, if so check if the person has 'Python' skill and print out the result.
 * If a person skills has only JavaScript and React, print('He is a front end developer'), if the person skills has Node, Python, MongoDB, print('He is a backend developer'), if the person skills has React, Node and MongoDB, Print('He is a fullstack developer'), else print('unknown title') - for more accurate results more conditions can be nested!
 * If the person is married and if he lives in Finland, print the information in the following format:

    Asabeneh Yetayeh lives in Finland. He is married.
```

🎉 CONGRATULATIONS ! 🎉

DAY-10 LOOPS

Loops

Life is full of routines. In programming we also do lots of repetitive tasks. In order to handle repetitive task programming languages use loops. Python programming language also provides the following types of two loops:

1. while loop
2. for loop

While Loop

We use the reserved word *while* to make a while loop. It is used to execute a block of statements repeatedly until a given condition is satisfied. When the condition becomes false, the lines of code after the loop will be continued to be executed.

```
  # syntax
while condition:
    code goes here
```

Example:

```
count = 0
while count < 5:
    print(count)
    count = count + 1
#prints from 0 to 4
```

In the above while loop, the condition becomes false when count is 5. That is when the loop stops. If we are interested to run block of code once the condition is no longer true, we can use *else*.

```
  # syntax
while condition:
    code goes here
else:
    code goes here
```

Example:

```
count = 0
while count < 5:
    print(count)
    count = count + 1
else:
    print(count)
```

The above loop condition will be false when count is 5 and the loop stops, and execution starts the else statement. As a result 5 will be printed.

Break and Continue - Part 1

- Break: We use break when we like to get out of or stop the loop.

```
# syntax
while condition:
    code goes here
    if another_condition:
        break
```

Example:

```
count = 0
while count < 5:
    print(count)
    count = count + 1
    if count == 3:
        break
```

The above while loop only prints 0, 1, 2, but when it reaches 3 it stops.

- Continue: With the continue statement we can skip the current iteration, and continue with the next:

```
  # syntax
while condition:
    code goes here
    if another_condition:
        continue
```

Example:

```
count = 0
while count < 5:
    if count == 3:
        count = count + 1
        continue
    print(count)
    count = count + 1
```

The above while loop only prints 0, 1, 2 and 4 (skips 3).

For Loop

A *for* keyword is used to make a for loop, similar with other programming languages, but with some syntax differences. Loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

- For loop with list

```
# syntax
for iterator in lst:
    code goes here
```

Example:

```
numbers = [0, 1, 2, 3, 4, 5]
for number in numbers: # number is temporary name to refer to
the list's items, valid only inside this loop
    print(number)       # the numbers will be printed line by
line, from 0 to 5
```

- For loop with string

```
# syntax
for iterator in string:
    code goes here
```

Example:

```
language = 'Python'
for letter in language:
    print(letter)


for i in range(len(language)):
    print(language[i])
```

- For loop with tuple

syntax

```
for iterator in tpl:
    code goes here
```

Example:

```
numbers = (0, 1, 2, 3, 4, 5)
for number in numbers:
    print(number)
```

- For loop with dictionary Looping through a dictionary gives you the key of the dictionary.

```
  # syntax
for iterator in dct:
    code goes here
```

Example:

```
person = {
    'first_name':'Asabeneh',
    'last_name':'Yetayeh',
    'age':250,
```

```
    'country':'Finland',
    'is_marred':True,
    'skills':['JavaScript', 'React', 'Node', 'MongoDB',
'Python'],
    'address':{
        'street':'Space street',
        'zipcode':'02210'
    }
}
for key in person:
    print(key)

for key, value in person.items():
    print(key, value) # this way we get both keys and values
printed out
```

- Loops in set

```
# syntax
for iterator in st:
    code goes here
```

Example:

```
it_companies = {'Facebook', 'Google', 'Microsoft', 'Apple',
'IBM', 'Oracle', 'Amazon'}
for company in it_companies:
    print(company)
```

Break and Continue - Part 2

Short reminder: *Break*: We use break when we like to stop our loop before it is completed.

```
# syntax
for iterator in sequence:
    code goes here
    if condition:
        break
```

Example:

```
numbers = (0,1,2,3,4,5)
for number in numbers:
    print(number)
    if number == 3:
        break
```

In the above example, the loop stops when it reaches 3.

Continue: We use continue when we like to skip some of the steps in the iteration of the loop.

```
  # syntax
for iterator in sequence:
    code goes here
    if condition:
        continue
```

Example:

```
numbers = (0,1,2,3,4,5)
for number in numbers:
    print(number)
    if number == 3:
        continue
    print('Next number should be ', number + 1) if number != 5
else print("loop's end") # for short hand conditions need both
if and else statements
print('outside the loop')
```

In the example above, if the number equals 3, the step *after* the condition (but inside the loop) is skipped and the execution of the loop continues if there are any iterations left.

The Range Function

The *range()* function is used list of numbers. The *range(start, end, step)* takes three parameters: starting, ending and increment. By default it starts from 0 and the increment is 1. The range sequence needs at least 1 argument (end). Creating sequences using range

```
lst = list(range(11))
print(lst) # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
st = set(range(1, 11))    # 2 arguments indicate start and end
of the sequence, step set to default 1
print(st) # {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

lst = list(range(0,11,2))
print(lst) # [0, 2, 4, 6, 8, 10]
st = set(range(0,11,2))
print(st) #  {0, 2, 4, 6, 8, 10}

# syntax
for iterator in range(start, end, step):
```

Example:

```
for number in range(11):
    print(number)   # prints 0 to 10, not including 11
```

Nested For Loop

We can write loops inside a loop.

```
# syntax
for x in y:
    for t in x:
        print(t)
```

Example:

```
person = {
    'first_name': 'Asabeneh',
    'last_name': 'Yetayeh',
    'age': 250,
    'country': 'Finland',
    'is_marred': True,
    'skills': ['JavaScript', 'React', 'Node', 'MongoDB',
'Python'],
    'address': {
        'street': 'Space street',
        'zipcode': '02210'
    }
}
for key in person:
    if key == 'skills':
        for skill in person['skills']:
            print(skill)
```

For Else

If we want to execute some message when the loop ends, we use else.

```
# syntax
for iterator in range(start, end, step):
    do something
else:
    print('The loop ended')
```

Example:

```
for number in range(11):
    print(number)   # prints 0 to 10, not including 11
else:
    print('The loop stops at', number)
```

Pass

In python when statement is required (after semicolon), but we don't like to execute any code there, we can write the word *pass* to avoid errors. Also we can use it as a placeholder, for future statements.

Example:

```
for number in range(6):
    pass
```

🎯 You established a big milestone, you are unstoppable. Keep going! You have just completed day 10 challenges and you are 10 steps a head in to your way to greatness. Now do some exercises for your brain and muscles.

💻 Exercises: Day 10

Exercises: Level 1

1. Iterate 0 to 10 using for loop, do the same using while loop.
2. Iterate 10 to 0 using for loop, do the same using while loop.
3. Write a loop that makes seven calls to print(), so we get on the output the following triangle:

```
  #
  ##
  ###
  ####
  #####
  ######
  #######
```

4. Use nested loops to create the following:

```
# # # # # # # #
# # # # # # # #
# # # # # # # #
# # # # # # # #
# # # # # # # #
# # # # # # # #
# # # # # # # #
# # # # # # # #
```

5. Print the following pattern:

```
0 x 0 = 0
1 x 1 = 1
2 x 2 = 4
3 x 3 = 9
4 x 4 = 16
5 x 5 = 25
6 x 6 = 36
7 x 7 = 49
8 x 8 = 64
9 x 9 = 81
10 x 10 = 100
```

6. Iterate through the list, ['Python', 'Numpy','Pandas','Django', 'Flask'] using a for loop and print out the items.
7. Use for loop to iterate from 0 to 100 and print only even numbers
8. Use for loop to iterate from 0 to 100 and print only odd numbers

Exercises: Level 2

1. Use for loop to iterate from 0 to 100 and print the sum of all numbers.

```
The sum of all numbers is 5050.
```

1. Use for loop to iterate from 0 to 100 and print the sum of all evens and the sum of all odds.

```
The sum of all evens is 2550. And the sum of all odds is 2500.
```

Exercises: Level 3

1. Go to the data folder and use the countries.py file. Loop through the countries and extract all the countries containing the word *land*.
2. This is a fruit list, ['banana', 'orange', 'mango', 'lemon'] reverse the order using loop.
3. Go to the data folder and use the countries_data.py file.
    i. What are the total number of languages in the data
    ii. Find the ten most spoken languages from the data
    iii. Find the 10 most populated countries in the world

🎉 CONGRATULATIONS ! 🎉

DAY-11 FUNCTIONS

Functions

So far we have seen many built-in Python functions. In this section, we will focus on custom functions. What is a function? Before we start making functions, let us learn what a function is and why we need them?

Defining a Function

A function is a reusable block of code or programming statements designed to perform a certain task. To define or declare a function, Python provides the *def* keyword. The following is the syntax for defining a function. The function block of code is executed only if the function is called or invoked.

Declaring and Calling a Function

When we make a function, we call it declaring a function. When we start using the it, we call it *calling* or *invoking* a function. Function can be declared with or without parameters.

```
# syntax
# Declaring a function
def function_name():
    codes
    codes
# Calling a function
function_name()
```

Function without Parameters

Function can be declared without parameters.

Example:

```
def generate_full_name ():
    first_name = 'Asabeneh'
    last_name = 'Yetayeh'
    space = ' '
    full_name = first_name + space + last_name
    print(full_name)
generate_full_name () # calling a function

def add_two_numbers ():
    num_one = 2
    num_two = 3
    total = num_one + num_two
    print(total)
add_two_numbers()
```

Function Returning a Value - Part 1

Function can also return values, if a function does not have a return statement, the value of the function is None. Let us rewrite the above functions using return. From now on, we get a value from a function when we call the function and print it.

```
def generate_full_name ():
    first_name = 'Asabeneh'
    last_name = 'Yetayeh'
    space = ' '
    full_name = first_name + space + last_name
    return full_name
print(generate_full_name())

def add_two_numbers ():
    num_one = 2
    num_two = 3
    total = num_one + num_two
    return total
print(add_two_numbers())
```

Function with Parameters

In a function we can pass different data types(number, string, boolean, list, tuple, dictionary or set) as a parameter

- Single Parameter: If our function takes a parameter we should call our function with an argument

```
  # syntax
  # Declaring a function
  def function_name(parameter):
    codes
    codes
  # Calling function
  print(function_name(argument))
```

Example:

```
def greetings (name):
```

```
    message = name + ', welcome to Python for Everyone!'
    return message

print(greetings('Asabeneh'))

def add_ten(num):
    ten = 10
    return num + ten
print(add_ten(90))

def square_number(x):
    return x * x
print(square_number(2))

def area_of_circle (r):
    PI = 3.14
    area = PI * r ** 2
    return area
print(area_of_circle(10))

def sum_of_numbers(n):
    total = 0
    for i in range(n+1):
        total+=i
    print(total)
print(sum_of_numbers(10)) # 55
print(sum_of_numbers(100)) # 5050
```

- Two Parameter: A function may or may not have a parameter or parameters. A function may also have two or more parameters. If our function takes parameters we should call it with arguments. Let us check a function with two parameters:

```
  # syntax
  # Declaring a function
  def function_name(para1, para2):
    codes
    codes
  # Calling function
  print(function_name(arg1, arg2))
```

Example:

```
def generate_full_name (first_name, last_name):
    space = ' '
      full_name = first_name + space + last_name
      return full_name
print('Full Name: ', generate_full_name('Asabeneh','Yetayeh'))

def sum_two_numbers (num_one, num_two):
    sum = num_one + num_two
    return sum
```

```
print('Sum of two numbers: ', sum_two_numbers(1, 9))

def calculate_age (current_year, birth_year):
    age = current_year - birth_year
    return age;

print('Age: ', calculate_age(2021, 1819))

def weight_of_object (mass, gravity):
    weight = str(mass * gravity)+ ' N' # the value has to be
changed to a string first
    return weight
print('Weight of an object in Newtons: ',
weight_of_object(100, 9.81))
```

Passing Arguments with Key and Value

If we pass the arguments with key and value, the order of the arguments does not matter.

```
# syntax
# Declaring a function
def function_name(para1, para2):
    codes
    codes
# Calling function
print(function_name(para1 = 'John', para2 = 'Doe')) # the
order of arguments does not matter here
```

Example:

```
def print_fullname(firstname, lastname):
    space = ' '
    full_name = firstname  + space + lastname
    print(full_name)
print(print_fullname(firstname = 'Asabeneh', lastname =
'Yetayeh'))

def add_two_numbers (num1, num2):
    total = num1 + num2
    print(total)
print(add_two_numbers(num2 = 3, num1 = 2)) # Order does not
matter
```

Function Returning a Value - Part 2

If we do not return a value with a function, then our function is returning *None* by default. To return a value with a function we use the keyword *return* followed by the variable we are returning. We can return any kind of data types from a function.

- Returning a string: **Example:**

```
def print_name(firstname):
    return firstname
print_name('Asabeneh') # Asabeneh

def print_full_name(firstname, lastname):
    space = ' '
    full_name = firstname  + space + lastname
    return full_name
print_full_name(firstname='Asabeneh', lastname='Yetayeh')
```

- Returning a number:

Example:

```
def add_two_numbers (num1, num2):
    total = num1 + num2
    return total
print(add_two_numbers(2, 3))

def calculate_age (current_year, birth_year):
    age = current_year - birth_year
    return age;
print('Age: ', calculate_age(2019, 1819))
```

- Returning a boolean: **Example:**

```
def is_even (n):
    if n % 2 == 0:
        print('even')
        return True    # return stops further execution of the
function, similar to break
    return False
print(is_even(10)) # True
print(is_even(7)) # False
```

- Returning a list: **Example:**

```
def find_even_numbers(n):
    evens = []
    for i in range(n + 1):
        if i % 2 == 0:
            evens.append(i)
    return evens
print(find_even_numbers(10))
```

Function with Default Parameters

Sometimes we pass default values to parameters, when we invoke the function. If we do not pass arguments when calling the function, their default values will be used.

```
# syntax
# Declaring a function
def function_name(param = value):
    codes
    codes
# Calling function
function_name()
function_name(arg)
```

Example:

```
def greetings (name = 'Peter'):
    message = name + ', welcome to Python for Everyone!'
    return message
print(greetings())
print(greetings('Asabeneh'))

def generate_full_name (first_name = 'Asabeneh', last_name =
'Yetayeh'):
    space = ' '
    full_name = first_name + space + last_name
    return full_name

print(generate_full_name())
print(generate_full_name('David','Smith'))

def calculate_age (birth_year,current_year = 2021):
    age = current_year - birth_year
    return age;
print('Age: ', calculate_age(1821))

def weight_of_object (mass, gravity = 9.81):
    weight = str(mass * gravity)+ ' N' # the value has to be
changed to string first
    return weight
print('Weight of an object in Newtons: ',
weight_of_object(100)) # 9.81 - average gravity on Earth's
surface
print('Weight of an object in Newtons: ',
weight_of_object(100, 1.62)) # gravity on the surface of the
Moon
```

Arbitrary Number of Arguments

If we do not know the number of arguments we pass to our function, we can create a function which can take arbitrary number of arguments by adding * before the parameter name.

```
# syntax
# Declaring a function
def function_name(*args):
    codes
    codes
# Calling function
function_name(param1, param2, param3,..)
```

Example:

```
def sum_all_nums(*nums):
    total = 0
    for num in nums:
        total += num     # same as total = total + num
    return total
print(sum_all_nums(2, 3, 5)) # 10
```

Default and Arbitrary Number of Parameters in Functions

```
def generate_groups (team,*args):
    print(team)
    for i in args:
        print(i)
print(generate_groups('Team-
1','Asabeneh','Brook','David','Eyob'))
```

Function as a Parameter of Another Function

```
#You can pass functions around as parameters
def square_number (n):
    return n * n
def do_something(f, x):
    return f(x)
print(do_something(square_number, 3)) # 27
```

⚽ You achieved quite a lot so far. Keep going! You have just completed day 11 challenges and you are 11 steps a head in to your way to greatness. Now do some exercises for your brain and muscles.

Testimony

Now it is time to express your thoughts about the Author and 30DaysOfPython. You can leave your testimonial on this link

💻 Exercises: Day 11

Exercises: Level 1

1. Declare a function *add_two_numbers*. It takes two parameters and it returns a sum.
2. Area of a circle is calculated as follows: area = π x r x r. Write a function that calculates *area_of_circle*.
3. Write a function called add_all_nums which takes arbitrary number of arguments and sums all the arguments. Check if all the list items are number types. If not do give a reasonable feedback.
4. Temperature in °C can be converted to °F using this formula: °F = (°C x 9/5) + 32. Write a function which converts °C to °F, *convert_celsius_to-fahrenheit*.
5. Write a function called check-season, it takes a month parameter and returns the season: Autumn, Winter, Spring or Summer.
6. Write a function called calculate_slope which return the slope of a linear equation
7. Quadratic equation is calculated as follows: $ax^2 + bx + c = 0$. Write a function which calculates solution set of a quadratic equation, *solve_quadratic_eqn*.
8. Declare a function named print_list. It takes a list as a parameter and it prints out each element of the list.
9. Declare a function named reverse_list. It takes an array as a parameter and it returns the reverse of the array (use loops).

```
print(reverse_list([1, 2, 3, 4, 5]))
# [5, 4, 3, 2, 1]
print(reverse_list1(["A", "B", "C"]))
# ["C", "B", "A"]
```

10. Declare a function named capitalize_list_items. It takes a list as a parameter and it returns a capitalized list of items
11. Declare a function named add_item. It takes a list and an item parameters. It returns a list with the item added at the end.

```
food_staff = ['Potato', 'Tomato', 'Mango', 'Milk']
print(add_item(food_staff, 'Meat'))     # ['Potato', 'Tomato',
'Mango', 'Milk','Meat']
numbers = [2, 3, 7, 9]
print(add_item(numbers, 5))       [2, 3, 7, 9, 5]
```

12. Declare a function named remove_item. It takes a list and an item parameters. It returns a list with the item removed from it.

```
food_staff = ['Potato', 'Tomato', 'Mango', 'Milk']
print(remove_item(food_staff, 'Mango'))  # ['Potato',
'Tomato', 'Milk'];
numbers = [2, 3, 7, 9]
print(remove_item(numbers, 3))  # [2, 7, 9]
```

13. Declare a function named sum_of_numbers. It takes a number parameter and it adds all the numbers in that range.

```
print(sum_of_numbers(5))  # 15
print(sum_of_numbers(10)) # 55
print(sum_of_numbers(100)) # 5050
```

14. Declare a function named sum_of_odds. It takes a number parameter and it adds all the odd numbers in that range.
15. Declare a function named sum_of_even. It takes a number parameter and it adds all the even numbers in that - range.

Exercises: Level 2

1. Declare a function named evens_and_odds . It takes a positive integer as parameter and it counts number of evens and odds in the number.

```
print(evens_and_odds(100))
# The number of odds are 50.
# The number of evens are 51.
```

1. Call your function factorial, it takes a whole number as a parameter and it return a factorial of the number
2. Call your function *is_empty*, it takes a parameter and it checks if it is empty or not
3. Write different functions which take lists. They should calculate_mean, calculate_median, calculate_mode, calculate_range, calculate_variance, calculate_std (standard deviation).

Exercises: Level 3

1. Write a function called is_prime, which checks if a number is prime.
2. Write a functions which checks if all items are unique in the list.

3. Write a function which checks if all the items of the list are of the same data type.
4. Write a function which check if provided variable is a valid python variable
5. Go to the data folder and access the countries-data.py file.

- Create a function called the most_spoken_languages in the world. It should return 10 or 20 most spoken languages in the world in descending order
- Create a function called the most_populated_countries. It should return 10 or 20 most populated countries in descending order.

🎉 CONGRATULATIONS !

DAY-12 MODULES

What is a Module

A module is a file containing a set of codes or a set of functions which can be included to an application. A module could be a file containing a single variable, a function or a big code base.

Creating a Module

To create a module we write our codes in a python script and we save it as a .py file. Create a file named mymodule.py inside your project folder. Let us write some code in this file.

```
# mymodule.py file
def generate_full_name(firstname, lastname):
    return firstname + ' ' + lastname
```

Create main.py file in your project directory and import the mymodule.py file.

Importing a Module

To import the file we use the *import* keyword and the name of the file only.

```
# main.py file
import mymodule
print(mymodule.generate_full_name('Asabeneh', 'Yetayeh')) #
Asabeneh Yetayeh
```

Import Functions from a Module

We can have many functions in a file and we can import all the functions differently.

```
# main.py file
from mymodule import generate_full_name, sum_two_nums, person,
gravity
print(generate_full_name('Asabneh','Yetayeh'))
print(sum_two_nums(1,9))
mass = 100;
weight = mass * gravity
print(weight)
print(person['firstname'])
```

Import Functions from a Module and Renaming

During importing we can rename the name of the module.

```
# main.py file
from mymodule import generate_full_name as fullname,
sum_two_nums as total, person as p, gravity as g
print(fullname('Asabneh','Yetayeh'))
print(total(1, 9))
mass = 100;
weight = mass * g
print(weight)
print(p)
print(p['firstname'])
```

Import Built-in Modules

Like other programming languages we can also import modules by importing the file/function using the key word *import*. Let's import the common module we will use most of the time. Some of the common built-in modules: *math*, *datetime*, *os*,*sys*, *random*, *statistics*, *collections*, *json*,*re*

OS Module

Using python *os* module it is possible to automatically perform many operating system tasks. The OS module in Python provides functions for creating, changing current working directory, and removing a directory (folder), fetching its contents, changing and identifying the current directory.

```
# import the module
import os
# Creating a directory
os.mkdir('directory_name')
# Changing the current directory
os.chdir('path')
# Getting current working directory
os.getcwd()
# Removing directory
os.rmdir()
```

Sys Module

The sys module provides functions and variables used to manipulate different parts of the Python runtime environment. Function sys.argv returns a list of command line arguments passed to a Python script. The item at index 0 in this list is always the name of the script, at index 1 is the argument passed from the command line.

Example of a script.py file:

```
import sys
#print(sys.argv[0], argv[1],sys.argv[2])  # this line would
print out: filename argument1 argument2
```

```
print('Welcome {}. Enjoy  {} challenge!'.format(sys.argv[1], sys.argv[2]))
```

Now to check how this script works I wrote in command line:

```
python script.py Asabeneh 30DaysOfPython
```

The result:

```
Welcome Asabeneh. Enjoy  30DayOfPython challenge!
```

Some useful sys commands:

```
# to exit sys
sys.exit()
# To know the largest integer variable it takes
sys.maxsize
# To know environment path
sys.path
# To know the version of python you are using
sys.version
```

Statistics Module

The statistics module provides functions for mathematical statistics of numeric data. The popular statistical functions which are defined in this module: *mean*, *median*, *mode*, *stdev* etc.

```
from statistics import * # importing all the statistics modules
ages = [20, 20, 4, 24, 25, 22, 26, 20, 23, 22, 26]
print(mean(ages))       # ~22.9
print(median(ages))     # 23
print(mode(ages))       # 20
print(stdev(ages))      # ~2.3
```

Math Module

Module containing many mathematical operations and constants.

```
import math
print(math.pi)           # 3.141592653589793, pi constant
print(math.sqrt(2))      # 1.4142135623730951, square root
print(math.pow(2, 3))    # 8.0, exponential function
print(math.floor(9.81))  # 9, rounding to the lowest
print(math.ceil(9.81))   # 10, rounding to the highest
print(math.log10(100))   # 2, logarithm with 10 as base
```

Now, we have imported the *math* module which contains lots of function which can help us to perform mathematical calculations. To check what functions the module has got, we can use *help(math)*, or *dir(math)*. This will display the available functions in the module. If we want to import only a specific function from the module we import it as follows:

```
from math import pi
print(pi)
```

It is also possible to import multiple functions at once

```
from math import pi, sqrt, pow, floor, ceil, log10
print(pi)                 # 3.141592653589793
print(sqrt(2))            # 1.4142135623730951
print(pow(2, 3))          # 8.0
print(floor(9.81))        # 9
print(ceil(9.81))         # 10
print(math.log10(100))    # 2
```

But if we want to import all the function in math module we can use * .

```
from math import *
print(pi)                  # 3.141592653589793, pi constant
print(sqrt(2))             # 1.4142135623730951, square root
print(pow(2, 3))           # 8.0, exponential
print(floor(9.81))         # 9, rounding to the lowest
print(ceil(9.81))          # 10, rounding to the highest
print(math.log10(100))     # 2
```

When we import we can also rename the name of the function.

```
from math import pi as  PI
print(PI) # 3.141592653589793
```

String Module

A string module is a useful module for many purposes. The example below shows some use of the string module.

```
import string
print(string.ascii_letters) #
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
print(string.digits)        # 0123456789
print(string.punctuation)   # !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
```

Random Module

By now you are familiar with importing modules. Let us do one more import to get very familiar with it. Let us import *random* module which gives us a random number between 0 and 0.9999.... The *random* module has lots of functions but in this section we will only use *random* and *randint*.

```
from random import random, randint
print(random())   # it doesn't take any arguments; it returns
a value between 0 and 0.9999
print(randint(5, 20)) # it returns a random integer number
between [5, 20] inclusive
```

🛞 You are going far. Keep going! You have just completed day 12 challenges and you are 12 steps a head in to your way to greatness. Now do some exercises for your brain and muscles.

💻 Exercises: Day 12

Exercises: Level 1

1. Writ a function which generates a six digit/character random_user_id.

```
  print(random_user_id());
  '1ee33d'
```

2. Modify the previous task. Declare a function named user_id_gen_by_user. It doesn't take any parameters but it takes two inputs using input(). One of the inputs is the number of characters and the second input is the number of IDs which are supposed to be generated.

```
print(user_id_gen_by_user()) # user input: 5 5
#output:
#kcsy2
#SMFYb
#bWmeq
#ZXOYh
#2Rgxf

print(user_id_gen_by_user()) # 16 5
#1GCSgPLMaBAVQZ26
#YD7eFwNQKNs7qXaT
#ycArC5yrRupyG00S
#UbGxOFI7UXSWAyKN
#dIV0SSUTgAdKwStr
```

3. Write a function named rgb_color_gen. It will generate rgb colors (3 values ranging from 0 to 255 each).

```
print(rgb_color_gen())
# rgb(125,244,255) - the output should be in this form
```

Exercises: Level 2

1. Write a function list_of_hexa_colors which returns any number of hexadecimal colors in an array (six hexadecimal numbers written after #. Hexadecimal numeral system is made out of 16 symbols, 0-9 and first 6 letters of the alphabet, a-f. Check the task 6 for output examples).
2. Write a function list_of_rgb_colors which returns any number of RGB colors in an array.
3. Write a function generate_colors which can generate any number of hexa or rgb colors.

```
   generate_colors('hexa', 3) #
['#a3e12f','#03ed55','#eb3d2b']
   generate_colors('hexa', 1) # ['#b334ef']
```

```
   generate_colors('rgb', 3)  # ['rgb(5, 55, 175','rgb(50,
105, 100','rgb(15, 26, 80']
   generate_colors('rgb', 1)  # ['rgb(33,79, 176)']
```

Exercises: Level 3

1. Call your function shuffle_list, it takes a list as a parameter and it returns a shuffled list
2. Write a function which returns an array of seven random numbers in a range of 0-9. All the numbers must be unique.

🎉 CONGRATULATIONS ! 🎉

DAY-13 List Comprehension

List Comprehension

List comprehension in Python is a compact way of creating a list from a sequence. It is a short way to create a new list. List comprehension is considerably faster than processing a list using the *for* loop.

```
# syntax
[i for i in iterable if expression]
```

Example:1

For instance if you want to change a string to a list of characters. You can use a couple of methods. Let's see some of them:

```
# One way
language = 'Python'
lst = list(language) # changing the string to list
print(type(lst))     # list
print(lst)           # ['P', 'y', 't', 'h', 'o', 'n']

# Second way: list comprehension
lst = [i for i in language]
print(type(lst)) # list
print(lst)       # ['P', 'y', 't', 'h', 'o', 'n']
```

Example:2

For instance if you want to generate a list of numbers

```
# Generating numbers
numbers = [i for i in range(11)]  # to generate numbers from 0
to 10
print(numbers)                    # [0, 1, 2, 3, 4, 5, 6, 7,
8, 9, 10]

# It is possible to do mathematical operations during
iteration
squares = [i * i for i in range(11)]
print(squares)                    # [0, 1, 4, 9, 16, 25, 36,
49, 64, 81, 100]

# It is also possible to make a list of tuples
numbers = [(i, i * i) for i in range(11)]
print(numbers)                              # [(0, 0), (1, 1),
(2, 4), (3, 9), (4, 16), (5, 25)]
```

Example:2

List comprehension can be combined with if expression

```
# Generating even numbers
even_numbers = [i for i in range(21) if i % 2 == 0]  # to
generate even numbers list in range 0 to 21
print(even_numbers)                    # [0, 2, 4, 6, 8, 10,
12, 14, 16, 18, 20]

# Generating odd numbers
odd_numbers = [i for i in range(21) if i % 2 != 0]  # to
generate odd numbers in range 0 to 21
print(odd_numbers)                      # [1, 3, 5, 7, 9, 11,
13, 15, 17, 19]
# Filter numbers: let's filter out positive even numbers from
the list below
numbers = [-8, -7, -3, -1, 0, 1, 3, 4, 5, 7, 6, 8, 10]
positive_even_numbers = [i for i in numbers if i % 2 == 0 and
i > 0]
print(positive_even_numbers)                    # [2, 4, 6, 8,
10, 12, 14, 16, 18, 20]

# Flattening a three dimensional array
list_of_lists = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
flattened_list = [ number for row in list_of_lists for number
in row]
print(flattened_list)    # [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Lambda Function

Lambda function is a small anonymous function without a name. It can take any number of arguments, but can only have one expression. Lambda function is similar to anonymous functions in JavaScript. We need it when we want to write an anonymous function inside another function.

Creating a Lambda Function

To create a lambda function we use *lambda* keyword followed by a parameter(s), followed by an expression. See the syntax and the example below. Lambda function does not use return but it explicitly returns the expression.

```
# syntax
x = lambda param1, param2, param3: param1 + param2 + param2
print(x(arg1, arg2, arg3))
```

Example:

```
# Named function
def add_two_nums(a, b):
    return a + b

print(add_two_nums(2, 3))     # 5
# Lets change the above function to a lambda function
add_two_nums = lambda a, b: a + b
print(add_two_nums(2,3))    # 5

# Self invoking lambda function
(lambda a, b: a + b)(2,3) # 5 - need to encapsulate it in
print() to see the result in the console

square = lambda x : x ** 2
print(square(3))    # 9
cube = lambda x : x ** 3
print(cube(3))    # 27

# Multiple variables
multiple_variable = lambda a, b, c: a ** 2 - 3 * b + 4 * c
print(multiple_variable(5, 5, 3)) # 22
```

Lambda Function Inside Another Function

Using a lambda function inside another function.

```
def power(x):
    return lambda n : x ** n

cube = power(2)(3)   # function power now need 2 arguments to
run, in separate rounded brackets
print(cube)          # 8
two_power_of_five = power(2)(5)
print(two_power_of_five)  # 32
```

🎉 Keep up the good work. Keep the momentum going, the sky is the limit! You have just completed day 13 challenges and you are 13 steps a head in to your way to greatness. Now do some exercises for your brain and muscles.

💻 Exercises: Day 13

1. Filter only negative and zero in the list using list comprehension

```
numbers = [-4, -3, -2, -1, 0, 2, 4, 6]
```

2. Flatten the following list of lists of lists to a one dimensional list :

```
list_of_lists =[[[1, 2, 3]], [[4, 5, 6]], [[7, 8, 9]]]

output
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

3. Using list comprehension create the following list of tuples:

```
[(0, 1, 0, 0, 0, 0, 0),
(1, 1, 1, 1, 1, 1, 1),
(2, 1, 2, 4, 8, 16, 32),
(3, 1, 3, 9, 27, 81, 243),
(4, 1, 4, 16, 64, 256, 1024),
(5, 1, 5, 25, 125, 625, 3125),
(6, 1, 6, 36, 216, 1296, 7776),
(7, 1, 7, 49, 343, 2401, 16807),
(8, 1, 8, 64, 512, 4096, 32768),
(9, 1, 9, 81, 729, 6561, 59049),
(10, 1, 10, 100, 1000, 10000, 100000)]
```

4. Flatten the following list to a new list:

```
countries = [[('Finland', 'Helsinki')], [('Sweden',
'Stockholm')], [('Norway', 'Oslo')]]
output:
[['FINLAND','FIN', 'HELSINKI'], ['SWEDEN', 'SWE',
'STOCKHOLM'], ['NORWAY', 'NOR', 'OSLO']]
```

5. Change the following list to a list of dictionaries:

```
countries = [[('Finland', 'Helsinki')], [('Sweden',
'Stockholm')], [('Norway', 'Oslo')]]
output:
[{'country': 'FINLAND', 'city': 'HELSINKI'},
{'country': 'SWEDEN', 'city': 'STOCKHOLM'},
{'country': 'NORWAY', 'city': 'OSLO'}]
```

6. Change the following list of lists to a list of concatenated strings:

```
names = [[('Asabeneh', 'Yetayeh')], [('David', 'Smith')],
[('Donald', 'Trump')], [('Bill', 'Gates')]]
output
['Asabeneh Yetaeyeh', 'David Smith', 'Donald Trump', 'Bill
Gates']
```

7. Write a lambda function which can solve a slope or y-intercept of linear functions.

🎋 CONGRATULATIONS ! 🎋

DAY-14 HIGHER ORDER FUNCTIONS

Higher Order Functions

In Python functions are treated as first class citizens, allowing you to perform the following operations on functions:

- A function can take one or more functions as parameters
- A function can be returned as a result of another function
- A function can be modified
- A function can be assigned to a variable

In this section, we will cover:

1. Handling functions as parameters
2. Returning functions as return value from another functions
3. Using Python closures and decorators

Function as a Parameter

```
def sum_numbers(nums):  # normal function
    return sum(nums)    # a sad function abusing the built-in
sum function :<

def higher_order_function(f, lst):  # function as a parameter
    summation = f(lst)
    return summation
result = higher_order_function(sum_numbers, [1, 2, 3, 4, 5])
print(result)       # 15
```

Function as a Return Value

```
def square(x):          # a square function
    return x ** 2

def cube(x):            # a cube function
    return x ** 3

def absolute(x):        # an absolute value function
    if x >= 0:
        return x
    else:
        return -(x)
```

```
def higher_order_function(type): # a higher order function
returning a function
    if type == 'square':
        return square
    elif type == 'cube':
        return cube
    elif type == 'absolute':
        return absolute

result = higher_order_function('square')
print(result(3))       # 9
result = higher_order_function('cube')
print(result(3))       # 27
result = higher_order_function('absolute')
print(result(-3))      # 3
```

You can see from the above example that the higher order function is returning different functions depending on the passed parameter

Python Closures

Python allows a nested function to access the outer scope of the enclosing function. This is is known as a Closure. Let us have a look at how closures work in Python. In Python, closure is created by nesting a function inside another encapsulating function and then returning the inner function. See the example below.

Example:

```
def add_ten():
    ten = 10
    def add(num):
        return num + ten
    return add

closure_result = add_ten()
print(closure_result(5))  # 15
print(closure_result(10))  # 20
```

Python Decorators

A decorator is a design pattern in Python that allows a user to add new functionality to an existing object without modifying its structure. Decorators are usually called before the definition of a function you want to decorate.

Creating Decorators

To create a decorator function, we need an outer function with an inner wrapper function.

Example:

```
# Normal function
def greeting():
    return 'Welcome to Python'
def uppercase_decorator(function):
    def wrapper():
        func = function()
        make_uppercase = func.upper()
        return make_uppercase
    return wrapper
g = uppercase_decorator(greeting)
print(g())          # WELCOME TO PYTHON

## Let us implement the example above with a decorator

'''This decorator function is a higher order function
that takes a function as a parameter'''
def uppercase_decorator(function):
    def wrapper():
        func = function()
        make_uppercase = func.upper()
        return make_uppercase
    return wrapper
@uppercase_decorator
def greeting():
    return 'Welcome to Python'
print(greeting())   # WELCOME TO PYTHON
```

Applying Multiple Decorators to a Single Function

```
'''These decorator functions are higher order functions
that take functions as parameters'''

# First Decorator
def uppercase_decorator(function):
    def wrapper():
        func = function()
        make_uppercase = func.upper()
        return make_uppercase
    return wrapper

# Second decorator
def split_string_decorator(function):
    def wrapper():
        func = function()
```

```
        splitted_string = func.split()
        return splitted_string

    return wrapper

@split_string_decorator
@uppercase_decorator     # order with decorators is important
in this case - .upper() function does not work with lists
def greeting():
    return 'Welcome to Python'
print(greeting())   # WELCOME TO PYTHON
```

Accepting Parameters in Decorator Functions

Most of the time we need our functions to take parameters, so we might need to define a decorator that accepts parameters.

```
def decorator_with_parameters(function):
    def wrapper_accepting_parameters(para1, para2, para3):
        function(para1, para2, para3)
        print("I live in {}".format(para3))
    return wrapper_accepting_parameters

@decorator_with_parameters
def print_full_name(first_name, last_name, country):
    print("I am {} {}. I love to teach.".format(
        first_name, last_name, country))

print_full_name("Asabeneh", "Yetayeh",'Finland')
```

Built-in Higher Order Functions

Some of the built-in higher order functions that we cover in this part are *map()*, *filter*, and *reduce*. Lambda function can be passed as a parameter and the best use case of lambda functions is in functions like map, filter and reduce.

Python - Map Function

The map() function is a built-in function that takes a function and iterable as parameters.

```
    # syntax
    map(function, iterable)
```

Example:1

```
numbers = [1, 2, 3, 4, 5] # iterable
def square(x):
```

```
    return x ** 2
numbers_squared = map(square, numbers)
print(list(numbers_squared))    # [1, 4, 9, 16, 25]
# Lets apply it with a lambda function
numbers_squared = map(lambda x : x ** 2, numbers)
print(list(numbers_squared))    # [1, 4, 9, 16, 25]
```

Example:2

```
numbers_str = ['1', '2', '3', '4', '5']  # iterable
numbers_int = map(int, numbers_str)
print(list(numbers_int))    # [1, 2, 3, 4, 5]
```

Example:3

```
names = ['Asabeneh', 'Lidiya', 'Ermias', 'Abraham']  #
iterable

def change_to_upper(name):
    return name.upper()

names_upper_cased = map(change_to_upper, names)
print(list(names_upper_cased))    # ['ASABENEH', 'LIDIYA',
'ERMIAS', 'ABRAHAM']

# Let us apply it with a lambda function
names_upper_cased = map(lambda name: name.upper(), names)
print(list(names_upper_cased))    # ['ASABENEH', 'LIDIYA',
'ERMIAS', 'ABRAHAM']
```

What actually map does is iterating over a list. For instance, it changes the names to upper case and returns a new list.

Python - Filter Function

The filter() function calls the specified function which returns boolean for each item of the specified iterable (list). It filters the items that satisfy the filtering criteria.

```
    # syntax
    filter(function, iterable)
```

Example:1

```
# Lets filter only even nubers
numbers = [1, 2, 3, 4, 5]  # iterable
```

```
def is_even(num):
    if num % 2 == 0:
        return True
    return False

even_numbers = filter(is_even, numbers)
print(list(even_numbers))        # [2, 4]
```

Example:2

```
numbers = [1, 2, 3, 4, 5]  # iterable

def is_odd(num):
    if num % 2 != 0:
        return True
    return False

odd_numbers = filter(is_odd, numbers)
print(list(odd_numbers))        # [1, 3, 5]
```

```
# Filter long name
names = ['Asabeneh', 'Lidiya', 'Ermias', 'Abraham']  #
iterable
def is_name_long(name):
    if len(name) > 7:
        return True
    return False

long_names = filter(is_name_long, names)
print(list(long_names))         # ['Asabeneh']
```

Python - Reduce Function

The *reduce()* function is defined in the functools module and we should import it from this module. Like map and filter it takes two parameters, a function and an iterable. However, it does not return another iterable, instead it returns a single value. **Example:1**

```
numbers_str = ['1', '2', '3', '4', '5']  # iterable
def add_two_nums(x, y):
    return int(x) + int(y)

total = reduce(add_two_nums, numbers_str)
print(total)    # 15
```

💻 Exercises: Day 14

```
countries = ['Estonia', 'Finland', 'Sweden', 'Denmark',
'Norway', 'Iceland']
```

```
names = ['Asabeneh', 'Lidiya', 'Ermias', 'Abraham']
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Exercises: Level 1

1. Explain the difference between map, filter, and reduce.
2. Explain the difference between higher order function, closure and decorator
3. Define a call function before map, filter or reduce, see examples.
4. Use for loop to print each country in the countries list.
5. Use for to print each name in the names list.
6. Use for to print each number in the numbers list.

Exercises: Level 2

1. Use map to create a new list by changing each country to uppercase in the countries list
2. Use map to create a new list by changing each number to its square in the numbers list
3. Use map to change each name to uppercase in the names list
4. Use filter to filter out countries containing 'land'.
5. Use filter to filter out countries having exactly six characters.
6. Use filter to filter out countries containing six letters and more in the country list.
7. Use filter to filter out countries starting with an 'E'
8. Chain two or more list iterators (eg. arr.map(callback).filter(callback).reduce(callback))
9. Declare a function called get_string_lists which takes a list as a parameter and then returns a list containing only string items.
10. Use reduce to sum all the numbers in the numbers list.
11. Use reduce to concatenate all the countries and to produce this sentence: Estonia, Finland, Sweden, Denmark, Norway, and Iceland are north European countries
12. Declare a function called categorize_countries that returns a list of countries with some common pattern (you can find the countries list in this repository as countries.js(eg 'land', 'ia', 'island', 'stan')).

13. Create a function returning a dictionary, where keys stand for starting letters of countries and values are the number of country names starting with that letter.
14. Declare a get_first_ten_countries function - it returns a list of first ten countries from the countries.js list in the data folder.
15. Declare a get_last_ten_countries function that returns the last ten countries in the countries list.

Exercises: Level 3

1. Use the countries_data.py (https://github.com/Asabeneh/30-Days-Of-Python/blob/master/data/countries-data.py) file and follow the tasks below:
    - Sort countries by name, by capital, by population
    - Sort out the ten most spoken languages by location.
    - Sort out the ten most populated countries.

🎉 CONGRATULATIONS ! 🎉

DAY-15 PYTHON TYPE ERRORS

Python Error Types

When we write code it is common that we make a typo or some other common error. If our code fails to run, the Python interpreter will display a message, containing feedback with information on where the problem occurs and the type of an error. It will also sometimes gives us suggestions on a possible fix. Understanding different types of errors in programming languages will help us to debug our code quickly and also it makes us better at what we do.

Let us see the most common error types one by one. First let us open our Python interactive shell. Go to your you computer terminal and write 'python'. The python interactive shell will be opened.

SyntaxError

Example 1: SyntaxError

```
asabeneh@Asabeneh:~$ python
Python 3.9.6 (default, Jun 28 2021, 15:26:21)
[Clang 11.0.0 (clang-1100.0.33.8)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> print 'hello world'
  File "<stdin>", line 1
    print 'hello world'
                      ^
SyntaxError: Missing parentheses in call to 'print'. Did you
mean print('hello world')?
>>>
```

As you can see we made a syntax error because we forgot to enclose the string with parenthesis and Python already suggests the solution. Let us fix it.

```
asabeneh@Asabeneh:~$ python
Python 3.9.6 (default, Jun 28 2021, 15:26:21)
[Clang 11.0.0 (clang-1100.0.33.8)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> print 'hello world'
  File "<stdin>", line 1
    print 'hello world'
                      ^
SyntaxError: Missing parentheses in call to 'print'. Did you
mean print('hello world')?
>>> print('hello world')
```

```
hello world
>>>
```

The error was a *SyntaxError*. After the fix our code was executed without a hitch. Let see more error types.

NameError

Example 1: NameError

```
asabeneh@Asabeneh:~$ python
Python 3.9.6 (default, Jun 28 2021, 15:26:21)
[Clang 11.0.0 (clang-1100.0.33.8)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> print(age)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'age' is not defined
>>>
```

As you can see from the message above, name age is not defined. Yes, it is true that we did not define an age variable but we were trying to print it out as if we had had declared it. Now, lets fix this by declaring it and assigning with a value.

```
asabeneh@Asabeneh:~$ python
Python 3.9.6 (default, Jun 28 2021, 15:26:21)
[Clang 11.0.0 (clang-1100.0.33.8)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> print(age)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'age' is not defined
>>> age = 25
>>> print(age)
25
>>>
```

The type of error was a *NameError*. We debugged the error by defining the variable name.

IndexError

Example 1: IndexError

```
asabeneh@Asabeneh:~$ python
Python 3.9.6 (default, Jun 28 2021, 15:26:21)
[Clang 11.0.0 (clang-1100.0.33.8)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> numbers = [1, 2, 3, 4, 5]
>>> numbers[5]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>>
```

In the example above, Python raised an *IndexError*, because the list has only indexes from 0 to 4 , so it was out of range.

ModuleNotFoundError

Example 1: ModuleNotFoundError

```
asabeneh@Asabeneh:~$ python
Python 3.9.6 (default, Jun 28 2021, 15:26:21)
[Clang 11.0.0 (clang-1100.0.33.8)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> import maths
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'maths'
>>>
```

In the example above, I added an extra s to math deliberately
and *ModuleNotFoundError* was raised. Lets fix it by removing the extra s from math.

```
asabeneh@Asabeneh:~$ python
Python 3.9.6 (default, Jun 28 2021, 15:26:21)
[Clang 11.0.0 (clang-1100.0.33.8)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> import maths
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'maths'
>>> import math
```

>>>

We fixed it, so let's use some of the functions from the math module.

AttributeError

Example 1: AttributeError

```
asabeneh@Asabeneh:~$ python
Python 3.9.6 (default, Jun 28 2021, 15:26:21)
[Clang 11.0.0 (clang-1100.0.33.8)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> import maths
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'maths'
>>> import math
>>> math.PI
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: module 'math' has no attribute 'PI'
>>>
```

As you can see, I made a mistake again! Instead of pi, I tried to call a PI function from maths module. It raised an attribute error, it means, that the function does not exist in the module. Lets fix it by changing from PI to pi.

```
asabeneh@Asabeneh:~$ python
Python 3.9.6 (default, Jun 28 2021, 15:26:21)
[Clang 11.0.0 (clang-1100.0.33.8)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> import maths
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'maths'
>>> import math
>>> math.PI
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: module 'math' has no attribute 'PI'
>>> math.pi
3.141592653589793
>>>
```

Now, when we call pi from the math module we got the result.

KeyError

Example 1: KeyError

```
asabeneh@Asabeneh:~$ python
Python 3.9.6 (default, Jun 28 2021, 15:26:21)
[Clang 11.0.0 (clang-1100.0.33.8)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> users = {'name':'Asab', 'age':250, 'country':'Finland'}
>>> users['name']
'Asab'
>>> users['county']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'county'
>>>
```

As you can see, there was a typo in the key used to get the dictionary value. so, this is a key error and the fix is quite straight forward. Let's do this!

```
asabeneh@Asabeneh:~$ python
Python 3.9.6 (default, Jun 28 2021, 15:26:21)
[Clang 11.0.0 (clang-1100.0.33.8)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> user = {'name':'Asab', 'age':250, 'country':'Finland'}
>>> user['name']
'Asab'
>>> user['county']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'county'
>>> user['country']
'Finland'
>>>
```

We debugged the error, our code ran and we got the value.

TypeError

Example 1: TypeError

```
asabeneh@Asabeneh:~$ python
Python 3.9.6 (default, Jun 28 2021, 15:26:21)
[Clang 11.0.0 (clang-1100.0.33.8)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> 4 + '3'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>>
```

In the example above, a TypeError is raised because we cannot add a number to a string. First solution would be to convert the string to int or float. Another solution

would be converting the number to a string (the result then would be '43'). Let us follow the first fix.

```
asabeneh@Asabeneh:~$ python
Python 3.9.6 (default, Jun 28 2021, 15:26:21)
[Clang 11.0.0 (clang-1100.0.33.8)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> 4 + '3'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> 4 + int('3')
7
>>> 4 + float('3')
7.0
>>>
```

Error removed and we got the result we expected.

ImportError

Example 1: TypeError

```
asabeneh@Asabeneh:~$ python
Python 3.9.6 (default, Jun 28 2021, 15:26:21)
[Clang 11.0.0 (clang-1100.0.33.8)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> from math import power
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: cannot import name 'power' from 'math'
>>>
```

There is no function called power in the math module, it goes with a different name: *pow*. Let's correct it:

```
asabeneh@Asabeneh:~$ python
Python 3.9.6 (default, Jun 28 2021, 15:26:21)
[Clang 11.0.0 (clang-1100.0.33.8)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> from math import power
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: cannot import name 'power' from 'math'
>>> from math import pow
>>> pow(2,3)
8.0
>>>
```

ValueError

```
asabeneh@Asabeneh:~$ python
Python 3.9.6 (default, Jun 28 2021, 15:26:21)
[Clang 11.0.0 (clang-1100.0.33.8)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> int('12a')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '12a'
>>>
```

In this case we cannot change the given string to a number, because of the 'a' letter in it.

ZeroDivisionError

```
asabeneh@Asabeneh:~$ python
Python 3.9.6 (default, Jun 28 2021, 15:26:21)
[Clang 11.0.0 (clang-1100.0.33.8)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> 1/0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>>
```

We cannot divide a number by zero.

We have covered some of the python error types, if you want to check more about it check the python documentation about python error types. If you are good at reading the error types then you will be able to fix your bugs fast and you will also become a better programmer.

🎉 You are excelling. You made it to half way to your way to greatness. Now do some exercises for your brain and for your muscle.

💻 Exercises: Day 15

1. Open you python interactive shell and try all the examples covered in this section.

🥕 CONGRATULATIONS ! 🥕

DAY-16 PYTHON DATE TIME

Python *datetime*

Python has got *datetime* module to handle date and time.

import datetime

print(dir(datetime))

```
['MAXYEAR', 'MINYEAR', '__builtins__', '__cached__',
'__doc__', '__file__', '__loader__', '__name__',
'__package__', '__spec__', 'date', 'datetime',
'datetime_CAPI', 'sys', 'time', 'timedelta', 'timezone',
'tzinfo']
```

With dir or help built-in commands it is possible to know the available functions in a certain module. As you can see, in the datetime module there are many functions, but we will focus on *date*, *datetime*, *time* and *timedelta*. Let se see them one by one.

Getting *datetime* Information

```
from datetime import datetime
now = datetime.now()
print(now)                          # 2021-07-08 07:34:46.549883
day = now.day                       # 8
month = now.month                   # 7
year = now.year                     # 2021
hour = now.hour                     # 7
minute = now.minute                 # 38
second = now.second
timestamp = now.timestamp()
print(day, month, year, hour, minute)
print('timestamp', timestamp)
print(f'{day}/{month}/{year}, {hour}:{minute}')  # 8/7/2021,
7:38
```

Timestamp or Unix timestamp is the number of seconds elapsed from 1st of January 1970 UTC.

Formatting Date Output Using *strftime*

```
from datetime import datetime
new_year = datetime(2020, 1, 1)
print(new_year)       # 2020-01-01 00:00:00
day = new_year.day
month = new_year.month
year = new_year.year
hour = new_year.hour
minute = new_year.minute
second = new_year.second
```

```
print(day, month, year, hour, minute) #1 1 2020 0 0
print(f'{day}/{month}/{year}, {hour}:{minute}')  # 1/1/2020,
0:0
```

Formatting date time using *strftime* method and the documentation can be found here.

```
from datetime import datetime
# current date and time
now = datetime.now()
t = now.strftime("%H:%M:%S")
print("time:", t)
time_one = now.strftime("%m/%d/%Y, %H:%M:%S")
# mm/dd/YY H:M:S format
print("time one:", time_one)
time_two = now.strftime("%d/%m/%Y, %H:%M:%S")
# dd/mm/YY H:M:S format
print("time two:", time_two)
```

```
time: 01:05:01
time one: 12/05/2019, 01:05:01
time two: 05/12/2019, 01:05:01
```

Here are all the *strftime* symbols we use to format time. An example of all the formats for this module.

Directive	Description	Example
%a	Weekday, short version	Wed
%A	Weekday, full version	Wednesday
%w	Weekday as a number 0-6, 0 is Sunday	3
%d	Day of month 01-31	31
%b	Month name, short version	Dec
%B	Month name, full version	December
%m	Month as a number 01-12	12
%y	Year, short version, without century	18
%Y	Year, full version	2018
%H	Hour 00-23	17
%I	Hour 00-12	05
%p	AM/PM	PM
%M	Minute 00-59	41
%S	Second 00-59	08
%f	Microsecond 000000-999999	548513
%z	UTC offset	+0100
%Z	Timezone	CST
%j	Day number of year 001-366	365
%U	Week number of year, Sunday as the first day of week, 00-53	52
%W	Week number of year, Monday as the first day of week, 00-53	52
%c	Local version of date and time	Mon Dec 31 17:41:00 2018
%x	Local version of date	12/31/18
%X	Local version of time	17:41:00
%%	A % character	%

String to Time Using *strptime*

Here is a documentation hat helps to understand the format.

```
from datetime import datetime
date_string = "5 December, 2019"
print("date_string =", date_string)
date_object = datetime.strptime(date_string, "%d %B, %Y")
print("date_object =", date_object)
```

```
date_string = 5 December, 2019
date_object = 2019-12-05 00:00:00
```

Using *date* from *datetime*

```
from datetime import date
d = date(2020, 1, 1)
print(d)
print('Current date:', d.today())    # 2019-12-05
# date object of today's date
today = date.today()
print("Current year:", today.year)   # 2019
print("Current month:", today.month) # 12
print("Current day:", today.day)     # 5
```

Time Objects to Represent Time

```
from datetime import time
# time(hour = 0, minute = 0, second = 0)
a = time()
print("a =", a)
# time(hour, minute and second)
b = time(10, 30, 50)
print("b =", b)
# time(hour, minute and second)
c = time(hour=10, minute=30, second=50)
print("c =", c)
# time(hour, minute, second, microsecond)
d = time(10, 30, 50, 200555)
print("d =", d)
```

output
a = 00:00:00
b = 10:30:50
c = 10:30:50
d = 10:30:50.200555

Difference Between Two Points in Time Using

```
today = date(year=2019, month=12, day=5)
new_year = date(year=2020, month=1, day=1)
time_left_for_newyear = new_year - today
# Time left for new year:  27 days, 0:00:00
print('Time left for new year: ', time_left_for_newyear)

t1 = datetime(year = 2019, month = 12, day = 5, hour = 0,
minute = 59, second = 0)
t2 = datetime(year = 2020, month = 1, day = 1, hour = 0,
minute = 0, second = 0)
diff = t2 - t1
print('Time left for new year:', diff) # Time left for new
year: 26 days, 23: 01: 00
```

Difference Between Two Points in Time Using *timedelta*

```
from datetime import timedelta
t1 = timedelta(weeks=12, days=10, hours=4, seconds=20)
```

```
t2 = timedelta(days=7, hours=5, minutes=3, seconds=30)
t3 = t1 - t2
print("t3 =", t3)
```

```
    date_string = 5 December, 2019
    date_object = 2019-12-05 00:00:00
    t3 = 86 days, 22:56:50
```

🍪 You are an extraordinary. You are 16 steps a head to your way to greatness. Now do some exercises for your brain and muscles.

💻 Exercises: Day 16

1. Get the current day, month, year, hour, minute and timestamp from datetime module
2. Format the current date using this format: "%m/%d/%Y, %H:%M:%S")
3. Today is 5 December, 2019. Change this time string to time.
4. Calculate the time difference between now and new year.
5. Calculate the time difference between 1 January 1970 and now.
6. Think, what can you use the datetime module for? Examples:
    - Time series analysis
    - To get a timestamp of any activities in an application
    - Adding posts on a blog

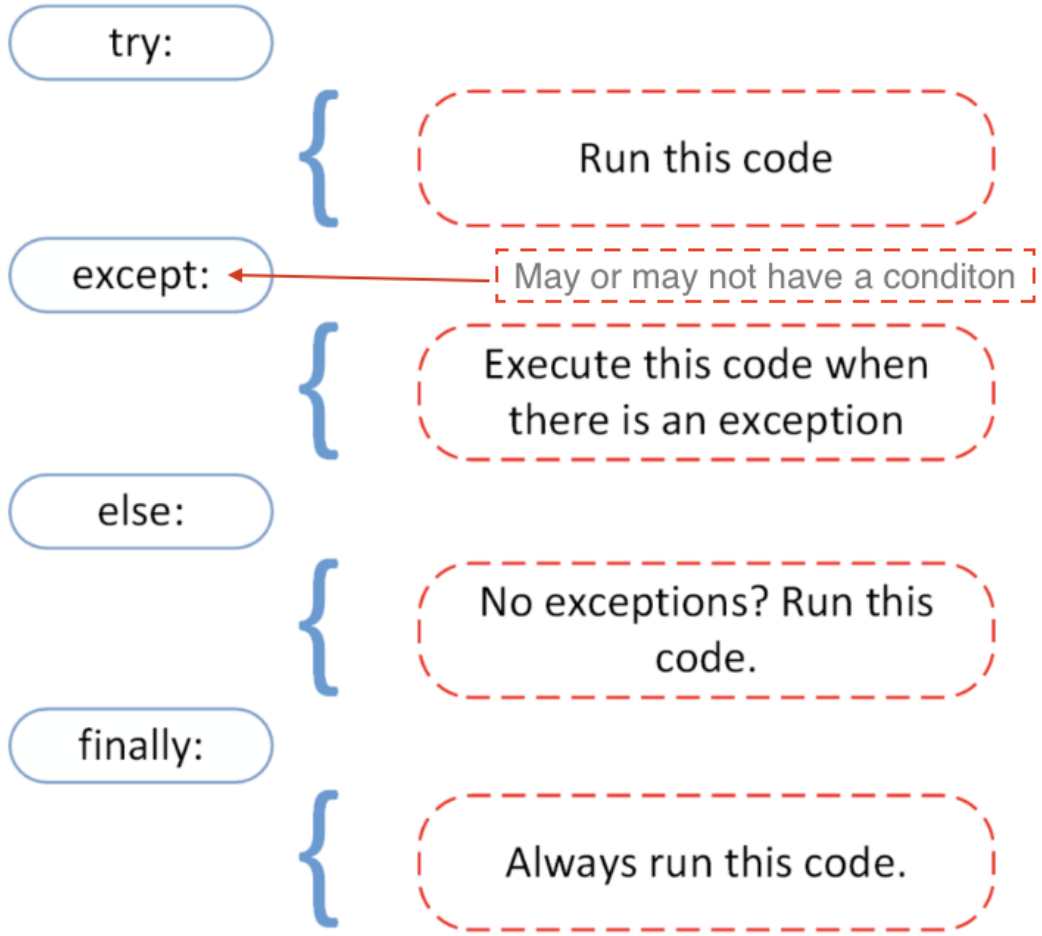
🥕 CONGRATULATIONS ! 🥕

DAY-17 EXCEPTION HANDLING

Exception Handling

Python uses *try* and *except* to handle errors gracefully. A graceful exit (or graceful handling) of errors is a simple programming idiom - a program detects a serious error condition and "exits gracefully", in a controlled manner as a result. Often the program prints a descriptive error message to a terminal or log as part of the graceful exit, this makes our application more robust. The cause of an exception is often external to the program itself. An example of exceptions could be an incorrect input, wrong file name, unable to find a file, a malfunctioning IO device. Graceful handling of errors prevents our applications from crashing.

We have covered the different Python *error* types in the previous section. If we use *try* and *except* in our program, then it will not raise errors in those blocks.




```
try:
    code in this block if things go well
except:
    code in this block run if things go wrong
```

Example:

```
try:
    print(10 + '5')
except:
    print('Something went wrong')
```

In the example above the second operand is a string. We could change it to float or int to add it with the number to make it work. But without any changes, the second block, *except*, will be executed.

Example:

```
try:
    name = input('Enter your name:')
    year_born = input('Year you were born:')
    age = 2019 - year_born
    print(f'You are {name}. And your age is {age}.')
except:
    print('Something went wrong')
```

```
Something went wrong
```

In the above example, the exception block will run and we do not know exactly the problem. To analyze the problem, we can use the different error types with except.

In the following example, it will handle the error and will also tell us the kind of error raised.

```
try:
    name = input('Enter your name:')
    year_born = input('Year you were born:')
    age = 2019 - year_born
    print(f'You are {name}. And your age is {age}.')
except TypeError:
    print('Type error occured')
except ValueError:
    print('Value error occured')
except ZeroDivisionError:
    print('zero division error occured')

Enter your name:Asabeneh
Year you born:1920
Type error occured
```

In the code above the output is going to be *TypeError*. Now, let's add an additional block:

```
try:
    name = input('Enter your name:')
    year_born = input('Year you born:')
    age = 2019 - int(year_born)
    print(f'You are {name}. And your age is {age}.')
except TypeError:
    print('Type error occur')
except ValueError:
    print('Value error occur')
except ZeroDivisionError:
    print('zero division error occur')
else:
    print('I usually run with the try block')
finally:
    print('I alway run.')
```

```
Enter your name:Asabeneh
Year you born:1920
You are Asabeneh. And your age is 99.
I usually run with the try block
I alway run.
```

It is also shorten the above code as follows:

```
try:
    name = input('Enter your name:')
    year_born = input('Year you born:')
    age = 2019 - int(year_born)
    print(f'You are {name}. And your age is {age}.')
except Exception as e:
    print(e)
```

Packing and Unpacking Arguments in Python

We use two operators:

- * for tuples
- ** for dictionaries

Let us take as an example below. It takes only arguments but we have list. We can unpack the list and changes to argument.

Unpacking

Unpacking Lists

```
def sum_of_five_nums(a, b, c, d, e):
```

```
    return a + b + c + d + e

lst = [1, 2, 3, 4, 5]
print(sum_of_five_nums(lst)) # TypeError: sum_of_five_nums()
missing 4 required positional arguments: 'b', 'c', 'd', and
'e'
```

When we run the this code, it raises an error, because this function takes numbers (not a list) as arguments. Let us unpack/destructure the list.

```
def sum_of_five_nums(a, b, c, d, e):
    return a + b + c + d + e

lst = [1, 2, 3, 4, 5]
print(sum_of_five_nums(*lst))  # 15
```

We can also use unpacking in the range built-in function that expects a start and an end.

```
numbers = range(2, 7)  # normal call with separate arguments
print(list(numbers)) # [2, 3, 4, 5, 6]
args = [2, 7]
numbers = range(*args)  # call with arguments unpacked from a
list
print(numbers)      # [2, 3, 4, 5,6]
```

A list or a tuple can also be unpacked like this:

```
countries = ['Finland', 'Sweden', 'Norway', 'Denmark',
'Iceland']
fin, sw, nor, *rest = countries
print(fin, sw, nor, rest)   # Finland Sweden Norway
['Denmark', 'Iceland']
numbers = [1, 2, 3, 4, 5, 6, 7]
one, *middle, last = numbers
print(one, middle, last)      #  1 [2, 3, 4, 5, 6] 7
```

Unpacking Dictionaries

```
def unpacking_person_info(name, country, city, age):
    return f'{name} lives in {country}, {city}. He is {age}
year old.'
dct = {'name':'Asabeneh', 'country':'Finland',
'city':'Helsinki', 'age':250}
print(unpacking_person_info(**dct)) # Asabeneh lives in
Finland, Helsinki. He is 250 years old.
```

Packing

Sometimes we never know how many arguments need to be passed to a python function. We can use the packing method to allow our function to take unlimited number or arbitrary number of arguments.

Packing Lists

```
def sum_all(*args):
    s = 0
    for i in args:
        s += i
    return s
print(sum_all(1, 2, 3))              # 6
print(sum_all(1, 2, 3, 4, 5, 6, 7)) # 28
```

Packing Dictionaries

```
def packing_person_info(**kwargs):
    # check the type of kwargs and it is a dict type
    # print(type(kwargs))
    # Printing dictionary items
    for key in kwargs:
        print(f"{key} = {kwargs[key]}")
    return kwargs

print(packing_person_info(name="Asabeneh",
      country="Finland", city="Helsinki", age=250))

name = Asabeneh
country = Finland
city = Helsinki
age = 250
{'name': 'Asabeneh', 'country': 'Finland', 'city': 'Helsinki',
'age': 250}
```

Spreading in Python

Like in JavaScript, spreading is possible in Python. Let us check it in an example below:

```
lst_one = [1, 2, 3]
lst_two = [4, 5, 6, 7]
lst = [0, *lst_one, *lst_two]
print(lst)          # [0, 1, 2, 3, 4, 5, 6, 7]
country_lst_one = ['Finland', 'Sweden', 'Norway']
country_lst_two = ['Denmark', 'Iceland']
nordic_countries = [*country_lst_one, *country_lst_two]
print(nordic_countries)  # ['Finland', 'Sweden', 'Norway',
'Denmark', 'Iceland']
```

Enumerate

If we are interested in an index of a list, we use *enumerate* built-in function to get the index of each item in the list.

```
for index, item in enumerate([20, 30, 40]):
    print(index, item)

for index, i in enumerate(countries):
    print('hi')
    if i == 'Finland':
        print('The country {i} has been found at index
{index}')

The country Finland has been found at index 1.
```

Zip

Sometimes we would like to combine lists when looping through them. See the example below:

```
fruits = ['banana', 'orange', 'mango', 'lemon', 'lime']
vegetables = ['Tomato', 'Potato', 'Cabbage','Onion', 'Carrot']
fruits_and_veges = []
for f, v in zip(fruits, vegetables):
    fruits_and_veges.append({'fruit':f, 'veg':v})

print(fruits_and_veges)
```

```
[{'fruit': 'banana', 'veg': 'Tomato'}, {'fruit': 'orange',
'veg': 'Potato'}, {'fruit': 'mango', 'veg': 'Cabbage'},
{'fruit': 'lemon', 'veg': 'Onion'}, {'fruit': 'lime', 'veg':
'Carrot'}]
```

😳 You are determined. You are 17 steps a head to your way to greatness. Now do some exercises for your brain and muscles.

Exercises: Day 17

1. names = ['Finland', 'Sweden', 'Norway','Denmark','Iceland', 'Estonia','Russia']. Unpack the first five countries and store them in a variable nordic_countries, store Estonia and Russia in es, and ru respectively.

🎉 CONGRATULATIONS ! 🎉

DAY 18 REGULAR EXPRESSIONS

Regular Expressions

A regular expression or RegEx is a special text string that helps to find patterns in data. A RegEx can be used to check if some pattern exists in a different data type. To use RegEx in python first we should import the RegEx module which is called *re*.

The *re* Module

After importing the module we can use it to detect or find patterns.

```
import re
```

Methods in *re* Module

To find a pattern we use different set of *re* character sets that allows to search for a match in a string.

- *re.match()*: searches only in the beginning of the first line of the string and returns matched objects if found, else returns None.
- *re.search*: Returns a match object if there is one anywhere in the string, including multiline strings.
- *re.findall*: Returns a list containing all matches
- *re.split*: Takes a string, splits it at the match points, returns a list
- *re.sub*: Replaces one or many matches within a string

Match

```
# syntac
re.match(substring, string, re.I)
# substring is a string or a pattern, string is the text we
look for a pattern , re.I is case ignore
```

```
import re

txt = 'I love to teach python and javaScript'
# It returns an object with span, and match
match = re.match('I love to teach', txt, re.I)
print(match)  # <re.Match object; span=(0, 15), match='I love
to teach'>
# We can get the starting and ending position of the match as
tuple using span
span = match.span()
print(span)     # (0, 15)
# Lets find the start and stop position from the span
start, end = span
print(start, end)  # 0, 15
substring = txt[start:end]
```

```
print(substring)        # I love to teach
```

As you can see from the example above, the pattern we are looking for (or the substring we are looking for) is *I love to teach*. The match function returns an object **only** if the text starts with the pattern.

```
import re

txt = 'I love to teach python and javaScript'
match = re.match('I like to teach', txt, re.I)
print(match)  # None
```

The string does not string with *I like to teach*, therefore there was no match and the match method returned None.

Search

```
# syntax
re.match(substring, string, re.I)
# substring is a pattern, string is the text we look for a
pattern , re.I is case ignore flag
```

```
import re

txt = '''Python is the most beautiful language that a human
being has ever created.
I recommend python for a first programming language'''

# It returns an object with span and match
match = re.search('first', txt, re.I)
print(match)  # <re.Match object; span=(100, 105),
match='first'>
# We can get the starting and ending position of the match as
tuple using span
span = match.span()
print(span)     # (100, 105)
# Lets find the start and stop position from the span
start, end = span
print(start, end)  # 100 105
substring = txt[start:end]
print(substring)        # first
```

As you can see, search is much better than match because it can look for the pattern throughout the text. Search returns a match object with a first match that was found, otherwise it returns *None*. A much better *re* function is *findall*. This function checks for the pattern through the whole string and returns all the matches as a list.

Searching for All Matches Using *findall*

findall() returns all the matches as a list

```
txt = '''Python is the most beautiful language that a human
being has ever created.
I recommend python for a first programming language'''

# It return a list
matches = re.findall('language', txt, re.I)
print(matches)  # ['language', 'language']
```

As you can see, the word *language* was found two times in the string. Let us practice some more. Now we will look for both Python and python words in the string:

```
txt = '''Python is the most beautiful language that a human
being has ever created.
I recommend python for a first programming language'''

# It returns list
matches = re.findall('python', txt, re.I)
print(matches)  # ['Python', 'python']
```

Since we are using *re.I* both lowercase and uppercase letters are included. If we do not have the re.I flag, then we will have to write our pattern differently. Let us check it out:

```
txt = '''Python is the most beautiful language that a human
being has ever created.
I recommend python for a first programming language'''

matches = re.findall('Python|python', txt)
print(matches)  # ['Python', 'python']

#
matches = re.findall('[Pp]ython', txt)
print(matches)  # ['Python', 'python']
```

Replacing a Substring

```
txt = '''Python is the most beautiful language that a human
being has ever created.
I recommend python for a first programming language'''

match_replaced = re.sub('Python|python', 'JavaScript', txt,
re.I)
print(match_replaced)  # JavaScript is the most beautiful
language that a human being has ever created.
# OR
match_replaced = re.sub('[Pp]ython', 'JavaScript', txt, re.I)
print(match_replaced)  # JavaScript is the most beautiful
language that a human being has ever created.
```

Let us add one more example. The following string is really hard to read unless we remove the % symbol. Replacing the % with an empty string will clean the text.

```
txt = '''%I a%m te%%a%%che%r% a%n%d %% I l%o%ve te%ach%ing.
T%he%re i%s n%o%th%ing as r%ewarding a%s e%duc%at%i%ng a%n%d
e%m%p%ow%er%ing p%e%o%ple.
I fo%und te%a%ching m%ore i%n%t%er%%es%ting t%h%an any other
%jobs.
D%o%es thi%s m%ot%iv%a%te %y%o%u to b%e a t%e%a%cher?'''

matches = re.sub('%', '', txt)
print(matches)
```

```
I am teacher and I love teaching.
There is nothing as rewarding as educating and empowering
people.
I found teaching more interesting than any other jobs. Does
this motivate you to be a teacher?
```

Splitting Text Using RegEx Split

```
txt = '''I am teacher and  I love teaching.
There is nothing as rewarding as educating and empowering
people.
I found teaching more interesting than any other jobs.
Does this motivate you to be a teacher?'''
print(re.split('\n', txt)) # splitting using \n - end of line
symbol
```

```
['I am teacher and  I love teaching.', 'There is nothing as
rewarding as educating and empowering people.', 'I found
teaching more interesting than any other jobs.', 'Does this
motivate you to be a teacher?']
```

Writing RegEx Patterns

To declare a string variable we use a single or double quote. To declare RegEx variable *r''*. The following pattern only identifies apple with lowercase, to make it case insensitive either we should rewrite our pattern or we should add a flag.

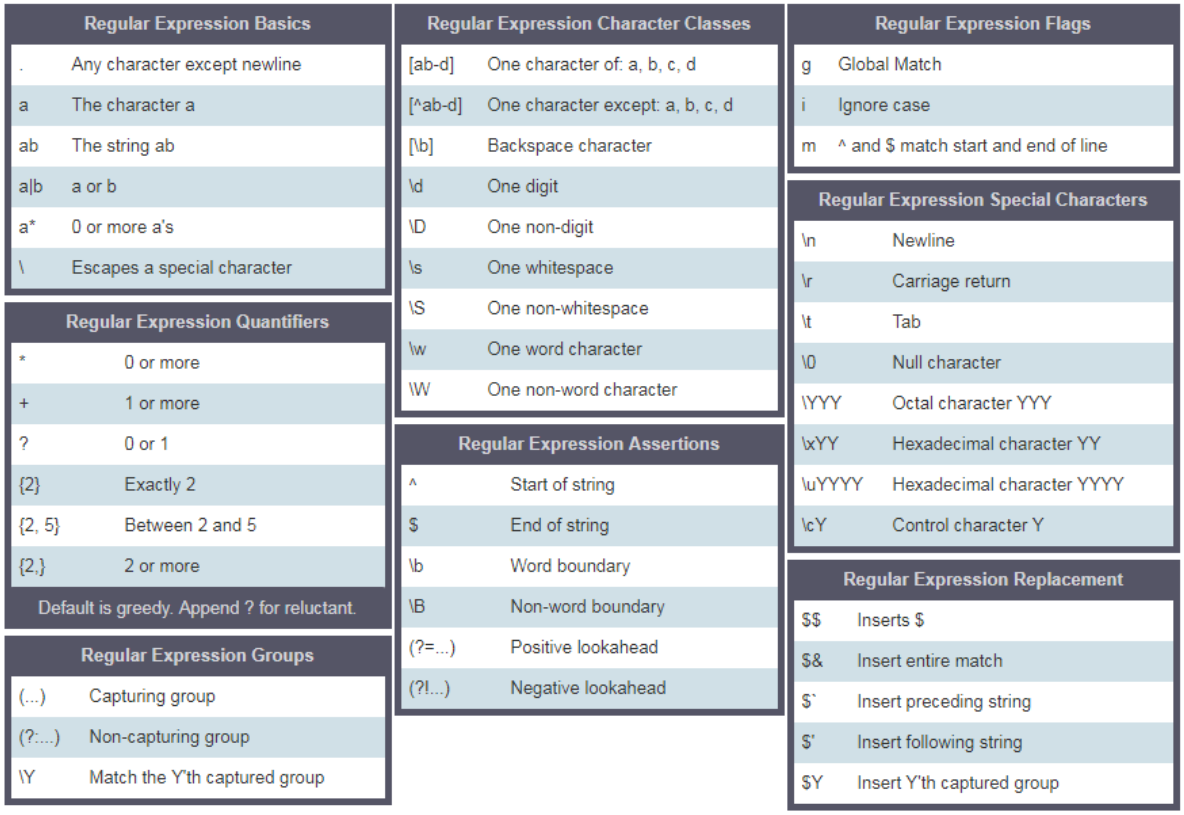
```
import re

regex_pattern = r'apple'
txt = 'Apple and banana are fruits. An old cliche says an
apple a day a doctor way has been replaced by a banana a day
keeps the doctor far far away. '
matches = re.findall(regex_pattern, txt)
print(matches)  # ['apple']
```

```
# To make case insensitive adding flag '
matches = re.findall(regex_pattern, txt, re.I)
print(matches)  # ['Apple', 'apple']
# or we can use a set of characters method
regex_pattern = r'[Aa]pple'  # this mean the first letter
could be Apple or apple
matches = re.findall(regex_pattern, txt)
print(matches)  # ['Apple', 'apple']
```

- []: A set of characters
    - [a-c] means, a or b or c
    - [a-z] means, any letter from a to z
    - [A-Z] means, any character from A to Z
    - [0-3] means, 0 or 1 or 2 or 3
    - [0-9] means any number from 0 to 9
    - [A-Za-z0-9] any single character, that is a to z, A to Z or 0 to 9
- \: uses to escape special characters
    - \d means: match where the string contains digits (numbers from 0-9)
    - \D means: match where the string does not contain digits
- . : any character except new line character(\n)
- ^: starts with
    - r'^substring' eg r'^love', a sentence that starts with a word love
    - r'[^abc] means not a, not b, not c.
- $: ends with
    - r'substring$' eg r'love$', sentence that ends with a word love
- *: zero or more times
    - r'[a]*' means a optional or it can occur many times.
- +: one or more times
    - r'[a]+' means at least once (or more)
- ?: zero or one time
    - r'[a]?' means zero times or once
- {3}: Exactly 3 characters
- {3,}: At least 3 characters

- {3,8}: 3 to 8 characters
- |: Either or
    - r'apple|banana' means either apple or a banana
- (): Capture and group



Regular Expression Basics	
.	Any character except newline
a	The character a
ab	The string ab
a|b	a or b
a*	0 or more a's
\	Escapes a special character

Regular Expression Quantifiers	
*	0 or more
+	1 or more
?	0 or 1
{2}	Exactly 2
{2, 5}	Between 2 and 5
{2,}	2 or more

Default is greedy. Append ? for reluctant.

Regular Expression Groups	
(...)	Capturing group
(?:...)	Non-capturing group
\Y	Match the Y'th captured group

Regular Expression Character Classes	
[ab-d]	One character of: a, b, c, d
[^ab-d]	One character except: a, b, c, d
[\b]	Backspace character
\d	One digit
\D	One non-digit
\s	One whitespace
\S	One non-whitespace
\w	One word character
\W	One non-word character

Regular Expression Assertions	
^	Start of string
$	End of string
\b	Word boundary
\B	Non-word boundary
(?=...)	Positive lookahead
(?!...)	Negative lookahead

Regular Expression Flags	
g	Global Match
i	Ignore case
m	^ and $ match start and end of line

Regular Expression Special Characters	
\n	Newline
\r	Carriage return
\t	Tab
\0	Null character
\YYY	Octal character YYY
\xYY	Hexadecimal character YY
\uYYYY	Hexadecimal character YYYY
\cY	Control character Y

Regular Expression Replacement	
$$	Inserts $
$&	Insert entire match
$`	Insert preceding string
$'	Insert following string
$Y	Insert Y'th captured group

Let us use examples to clarify the meta characters above

Square Bracket

Let us use square bracket to include lower and upper case

```
regex_pattern = r'[Aa]pple' # this square bracket mean either
A or a
txt = 'Apple and banana are fruits. An old cliche says an
apple a day a doctor way has been replaced by a banana a day
keeps the doctor far far away.'
matches = re.findall(regex_pattern, txt)
print(matches)  # ['Apple', 'apple']
```

If we want to look for the banana, we write the pattern as follows:

```
regex_pattern = r'[Aa]pple|[Bb]anana' # this square bracket
means either A or a
txt = 'Apple and banana are fruits. An old cliche says an
apple a day a doctor way has been replaced by a banana a day
keeps the doctor far far away.'
matches = re.findall(regex_pattern, txt)
```

```
print(matches)  # ['Apple', 'banana', 'apple', 'banana']
```

Using the square bracket and or operator , we manage to extract Apple, apple, Banana and banana.

Escape character(\) in RegEx

```
regex_pattern = r'\d'  # d is a special character which means digits
txt = 'This regular expression example was made on December 6, 2019 and revised on July 8, 2021'
matches = re.findall(regex_pattern, txt)
print(matches)  # ['6', '2', '0', '1', '9', '8', '2', '0', '2', '1'], this is not what we want
```

One or more times(+)

```
regex_pattern = r'\d+'  # d is a special character which means digits, + mean one or more times
txt = 'This regular expression example was made on December 6, 2019 and revised on July 8, 2021'
matches = re.findall(regex_pattern, txt)
print(matches)  # ['6', '2019', '8', '2021'] - now, this is better!
```

Period(.)

```
regex_pattern = r'[a].'  # this square bracket means a and . means any character except new line
txt = '''Apple and banana are fruits'''
matches = re.findall(regex_pattern, txt)
print(matches)  # ['an', 'an', 'an', 'a ', 'ar']

regex_pattern = r'[a].+'  # . any character, + any character one or more times
matches = re.findall(regex_pattern, txt)
print(matches)  # ['and banana are fruits']
```

Zero or more times(*)

Zero or many times. The pattern could may not occur or it can occur many times.

```
regex_pattern = r'[a].*'  # . any character, * any character zero or more times
txt = '''Apple and banana are fruits'''
```

```
matches = re.findall(regex_pattern, txt)
print(matches)  # ['and banana are fruits']
```

Zero or one time(?)

Zero or one time. The pattern may not occur or it may occur once.

```
txt = '''I am not sure if there is a convention how to write
the word e-mail.
Some people write it as email others may write it as Email or
E-mail.'''
regex_pattern = r'[Ee]-?mail'  # ? means here that '-' is
optional
matches = re.findall(regex_pattern, txt)
print(matches)  # ['e-mail', 'email', 'Email', 'E-mail']
```

Quantifier in RegEx

We can specify the length of the substring we are looking for in a text, using a curly bracket. Let us imagine, we are interested in a substring with a length of 4 characters:

```
txt = 'This regular expression example was made on December 6,
2019 and revised on July 8, 2021'
regex_pattern = r'\d{4}'  # exactly four times
matches = re.findall(regex_pattern, txt)
print(matches)  # ['2019', '2021']

txt = 'This regular expression example was made on December 6,
2019 and revised on July 8, 2021'
regex_pattern = r'\d{1, 4}'   # 1 to 4
matches = re.findall(regex_pattern, txt)
print(matches)  # ['6', '2019', '8', '2021']
```

Cart ^

- Starts with

```
txt = 'This regular expression example was made on December 6,
2019 and revised on July 8, 2021'
```

```
regex_pattern = r'^This'  # ^ means starts with
matches = re.findall(regex_pattern, txt)
print(matches)  # ['This']
```

- Negation

```
txt = 'This regular expression example was made on December 6,
2019 and revised on July 8, 2021'
regex_pattern = r'[^A-Za-z ]+'  # ^ in set character means
negation, not A to Z, not a to z, no space
matches = re.findall(regex_pattern, txt)
print(matches)  # ['6,', '2019', '8', '2021']
```

💻 Exercises: Day 18

Exercises: Level 1

1. What is the most frequent word in the following paragraph?

```
    paragraph = 'I love teaching. If you do not love teaching
what else can you love. I love Python if you do not love
something which can give you all the capabilities to develop
an application what else can you love.

  [
    (6, 'love'),
    (5, 'you'),
    (3, 'can'),
    (2, 'what'),
    (2, 'teaching'),
    (2, 'not'),
    (2, 'else'),
    (2, 'do'),
    (2, 'I'),
    (1, 'which'),
    (1, 'to'),
    (1, 'the'),
    (1, 'something'),
    (1, 'if'),
    (1, 'give'),
    (1, 'develop'),
    (1, 'capabilities'),
    (1, 'application'),
    (1, 'an'),
    (1, 'all'),
    (1, 'Python'),
    (1, 'If')
    ]
```

2. The position of some particles on the horizontal x-axis are -12, -4, -3 and -1 in the negative direction, 0 at origin, 4 and 8 in the positive direction. Extract these numbers from this whole text and find the distance between the two furthest particles.

```
points = ['-12', '-4', '-3', '-1', '0', '4', '8']
sorted_points =  [-12, -4, -3, -1, -1, 0, 2, 4, 8]
distance = 8 -(-12) # 20
```

Exercises: Level 2

1. Write a pattern which identifies if a string is a valid python variable

```
is_valid_variable('first_name') # True
is_valid_variable('first-name') # False
is_valid_variable('1first_name') # False
is_valid_variable('firstname') # True
```

Exercises: Level 3

1. Clean the following text. After cleaning, count three most frequent words in the string.

```
sentence = '''%I $am@% a %tea@cher%, &and& I lo%#ve
%tea@ching%;. There $is nothing; &as& mo@re rewarding as
educa@ting &and& @emp%o@wering peo@ple. ;I found tea@ching
m%o@re interesting tha@n any other %jo@bs. %Do@es thi%s
mo@tivate yo@u to be a tea@cher!?'''

print(clean_text(sentence));
I am a teacher and I love teaching There is nothing as more
rewarding as educating and empowering people I found teaching
more interesting than any other jobs Does this motivate you to
be a teacher

print(most_frequent_words(cleaned_text)) # [(3, 'I'), (2,
'teaching'), (2, 'teacher')]
```

🎉 CONGRATULATIONS ! 🎉

DAY-19 FILE HANDLING

File Handling

So far we have seen different Python data types. We usually store our data in different file formats. In addition to handling files, we will also see different file formats(.txt, .json, .xml, .csv, .tsv, .excel) in this section. First, let us get familiar with handling files with common file format(.txt).

File handling is an import part of programming which allows us to create, read, update and delete files. In Python to handle data we use *open()* built-in function.

```
# Syntax
open('filename', mode) # mode(r, a, w, x, t,b)  could be to
read, write, update
```

- "r" - Read - Default value. Opens a file for reading, it returns an error if the file does not exist
- "a" - Append - Opens a file for appending, creates the file if it does not exist
- "w" - Write - Opens a file for writing, creates the file if it does not exist
- "x" - Create - Creates the specified file, returns an error if the file exists
- "t" - Text - Default value. Text mode
- "b" - Binary - Binary mode (e.g. images)

Opening Files for Reading

The default mode of *open* is reading, so we do not have to specify 'r' or 'rt'. I have created and saved a file named reading_file_example.txt in the files directory. Let us see how it is done:

```
f = open('./files/reading_file_example.txt')
print(f) # <_io.TextIOWrapper
name='./files/reading_file_example.txt' mode='r'
encoding='UTF-8'>
```

As you can see in the example above, I printed the opened file and it gave some information about it. Opened file has different reading methods: *read()*, *readline*, *readlines*. An opened file has to be closed with *close()* method.

- *read()*: read the whole text as string. If we want to limit the number of characters we want to read, we can limit it by passing int value to the *read(number)* method.

```
f = open('./files/reading_file_example.txt')
txt = f.read()
```

```
print(type(txt))
print(txt)
f.close()

# output
<class 'str'>
This is an example to show how to open a file and read.
This is the second line of the text.
```

Instead of printing all the text, let us print the first 10 characters of the text file.

```
f = open('./files/reading_file_example.txt')
txt = f.read(10)
print(type(txt))
print(txt)
f.close()
```

```
# output
<class 'str'>
This is an
```

- *readline()*: read only the first line

```
f = open('./files/reading_file_example.txt')
line = f.readline()
print(type(line))
print(line)
f.close()

# output
<class 'str'>
This is an example to show how to open a file and read.
```

- *readlines()*: read all the text line by line and returns a list of lines

```
f = open('./files/reading_file_example.txt')
lines = f.readlines()
print(type(lines))
print(lines)
f.close()

# output
<class 'list'>
['This is an example to show how to open a file and read.\n',
'This is the second line of the text.']
```

Another way to get all the lines as a list is using *splitlines()*:

```
f = open('./files/reading_file_example.txt')
lines = f.read().splitlines()
print(type(lines))
print(lines)
f.close()
```

```
# output
<class 'list'>
['This is an example to show how to open a file and read.',
'This is the second line of the text.']
```

After we open a file, we should close it. There is a high tendency of forgetting to close them. There is a new way of opening files using *with* - closes the files by itself. Let us rewrite the the previous example with the *with* method:

```
with open('./files/reading_file_example.txt') as f:
    lines = f.read().splitlines()
    print(type(lines))
    print(lines)

# output
<class 'list'>
['This is an example to show how to open a file and read.',
'This is the second line of the text.']
```

Opening Files for Writing and Updating

To write to an existing file, we must add a mode as parameter to the *open()* function:

- "a" - append - will append to the end of the file, if the file does not it creates a new file.
- "w" - write - will overwrite any existing content, if the file does not exist it creates.

Let us append some text to the file we have been reading:

```
with open('./files/reading_file_example.txt','a') as f:
    f.write('This text has to be appended at the end')
```

The method below creates a new file, if the file does not exist:

```
with open('./files/writing_file_example.txt','w') as f:
    f.write('This text will be written in a newly created
file')
```

Deleting Files

We have seen in previous section, how to make and remove a directory using *os* module. Again now, if we want to remove a file we use *os* module.

```
import os
os.remove('./files/example.txt')
```

If the file does not exist, the remove method will raise an error, so it is good to use a condition like this:

```
import os
if os.path.exists('./files/example.txt'):
    os.remove('./files/example.txt')
else:
    print('The file does not exist')
```

File Types

File with txt Extension

File with *txt* extension is a very common form of data and we have covered it in the previous section. Let us move to the JSON file

File with json Extension

JSON stands for JavaScript Object Notation. Actually, it is a stringified JavaScript object or Python dictionary.

Example:

```
# dictionary
person_dct= {
    "name":"Asabeneh",
    "country":"Finland",
    "city":"Helsinki",
    "skills":["JavaScrip", "React","Python"]
}
# JSON: A string form a dictionary
person_json = "{'name': 'Asabeneh', 'country': 'Finland',
'city': 'Helsinki', 'skills': ['JavaScrip', 'React',
'Python']}"

# we use three quotes and make it multiple line to make it
more readable
person_json = '''{
    "name":"Asabeneh",
    "country":"Finland",
    "city":"Helsinki",
    "skills":["JavaScrip", "React","Python"]
}'''
```

Changing JSON to Dictionary

To change a JSON to a dictionary, first we import the json module and then we use *loads* method.

```
import json
# JSON
person_json = '''{
    "name": "Asabeneh",
    "country": "Finland",
    "city": "Helsinki",
    "skills": ["JavaScrip", "React", "Python"]
}'''
# let's change JSON to dictionary
person_dct = json.loads(person_json)
print(type(person_dct))
print(person_dct)
print(person_dct['name'])

# output
<class 'dict'>
{'name': 'Asabeneh', 'country': 'Finland', 'city': 'Helsinki',
'skills': ['JavaScrip', 'React', 'Python']}
Asabeneh
```

Changing Dictionary to JSON

To change a dictionary to a JSON we use *dumps* method from the json module.

```
import json
# python dictionary
person = {
    "name": "Asabeneh",
    "country": "Finland",
    "city": "Helsinki",
    "skills": ["JavaScrip", "React", "Python"]
}
# let's convert it to  json
person_json = json.dumps(person, indent=4) # indent could be
2, 4, 8. It beautifies the json
print(type(person_json))
print(person_json)
```

```
# output
# when you print it, it does not have the quote, but actually
it is a string
# JSON does not have type, it is a string type.
<class 'str'>
{
    "name": "Asabeneh",
```

```
    "country": "Finland",
    "city": "Helsinki",
    "skills": [
        "JavaScrip",
        "React",
        "Python"
    ]
}
```

Saving as JSON File

We can also save our data as a json file. Let us save it as a json file using the following steps. For writing a json file, we use the json.dump() method, it can take dictionary, output file, ensure_ascii and indent.

```
import json
# python dictionary
person = {
    "name": "Asabeneh",
    "country": "Finland",
    "city": "Helsinki",
    "skills": ["JavaScrip", "React", "Python"]
}
with open('./files/json_example.json', 'w', encoding='utf-8')
as f:
    json.dump(person, f, ensure_ascii=False, indent=4)
```

In the code above, we use encoding and indentation. Indentation makes the json file easy to read.

File with csv Extension

CSV stands for comma separated values. CSV is a simple file format used to store tabular data, such as a spreadsheet or database. CSV is a very common data format in data science.

Example:

```
"name","country","city","skills"
"Asabeneh","Finland","Helsinki","JavaScript"
```

Example:

```
import csv
with open('./files/csv_example.csv') as f:
```

```
    csv_reader = csv.reader(f, delimiter=',') # w use, reader
method to read csv
    line_count = 0
    for row in csv_reader:
        if line_count == 0:
            print(f'Column names are :{", ".join(row)}')
            line_count += 1
        else:
            print(
                f'\t{row[0]} is a teachers. He lives in
{row[1]}, {row[2]}.')
            line_count += 1
    print(f'Number of lines:  {line_count}')

# output:
Column names are :name, country, city, skills
        Asabeneh is a teacher. He lives in Finland, Helsinki.
Number of lines:  2
```

File with xlsx Extension

To read excel files we need to install *xlrd* package. We will cover this after we cover package installing using pip.

```
import xlrd
excel_book = xlrd.open_workbook('sample.xls)
print(excel_book.nsheets)
print(excel_book.sheet_names)
```

File with xml Extension

XML is another structured data format which looks like HTML. In XML the tags are not predefined. The first line is an XML declaration. The person tag is the root of the XML. The person has a gender attribute. **Example:XML**

```
<?xml version="1.0"?>
<person gender="female">
  <name>Asabeneh</name>
  <country>Finland</country>
  <city>Helsinki</city>
  <skills>
    <skill>JavaScrip</skill>
    <skill>React</skill>
    <skill>Python</skill>
  </skills>
</person>
```

For more information on how to read an XML file check the documentation

```
import xml.etree.ElementTree as ET
tree = ET.parse('./files/xml_example.xml')
root = tree.getroot()
print('Root tag:', root.tag)
print('Attribute:', root.attrib)
for child in root:
    print('field: ', child.tag)
```

```
# output
Root tag: person
Attribute: {'gender': 'male'}
field: name
field: country
field: city
field: skills
```

You are making a big progress. Maintain your momentum, keep the good work. Now do some exercises for your brain and muscles.

💻 Exercises: Day 19

Exercises: Level 1

1. Write a function which count number of lines and number of words in a text. All the files are in the data the folder: a) Read obama_speech.txt file and count number of lines and words b) Read michelle_obama_speech.txt file and count number of lines and words c) Read donald_speech.txt file and count number of lines and words d) Read melina_trump_speech.txt file and count number of lines and words
2. Read the countries_data.json data file in data directory, create a function that finds the ten most spoken languages

```
# Your output should look like this
print(most_spoken_languages(filename='./data/countries_data.json', 10))
[(91, 'English'),
(45, 'French'),
(25, 'Arabic'),
(24, 'Spanish'),
(9, 'Russian'),
(9, 'Portuguese'),
(8, 'Dutch'),
(7, 'German'),
(5, 'Chinese'),
(4, 'Swahili'),
(4, 'Serbian')]

# Your output should look like this
print(most_spoken_languages(filename='./data/countries_data.json', 3))
[(91, 'English'),
(45, 'French'),
(25, 'Arabic')]
```

3. Read the countries_data.json data file in data directory, create a function that creates a list of the ten most populated countries

```
# Your output should look like this
print(most_populated_countries(filename='./data/countries_data.json', 10))

[
{'country': 'China', 'population': 1377422166},
{'country': 'India', 'population': 1295210000},
{'country': 'United States of America', 'population': 323947000},
{'country': 'Indonesia', 'population': 258705000},
```

```
{'country': 'Brazil', 'population': 206135893},
{'country': 'Pakistan', 'population': 194125062},
{'country': 'Nigeria', 'population': 186988000},
{'country': 'Bangladesh', 'population': 161006790},
{'country': 'Russian Federation', 'population': 146599183},
{'country': 'Japan', 'population': 126960000}
]

# Your output should look like this

print(most_populated_countries(filename='./data/countries_data
.json', 3))
[
{'country': 'China', 'population': 1377422166},
{'country': 'India', 'population': 1295210000},
{'country': 'United States of America', 'population':
323947000}
]
```

Exercises: Level 2

4. Extract all incoming email addresses as a list from the email_exchange_big.txt file.
5. Find the most common words in the English language. Call the name of your function find_most_common_words, it will take two parameters - a string or a file and a positive integer, indicating the number of words. Your function will return an array of tuples in descending order. Check the output

```
    # Your output should look like this
    print(find_most_common_words('sample.txt', 10))
    [(10, 'the'),
    (8, 'be'),
    (6, 'to'),
    (6, 'of'),
    (5, 'and'),
    (4, 'a'),
    (4, 'in'),
    (3, 'that'),
    (2, 'have'),
    (2, 'I')]

    # Your output should look like this
    print(find_most_common_words('sample.txt', 5))

    [(10, 'the'),
    (8, 'be'),
    (6, 'to'),
    (6, 'of'),
```

```
    (5, 'and')]
```

🥕 CONGRATULATIONS ! 🥕

DAY-20 PIP

Python PIP - Python Package Manager

What is PIP ?

PIP stands for Preferred installer program. We use *pip* to install different Python packages. Package is a Python module that can contain one or more modules or other packages. A module or modules that we can install to our application is a package. In programming, we do not have to write every utility program, instead we install packages and import them to our applications.

Installing PIP

If you did not install pip, let us install it now. Go to your terminal or command prompt and copy and paste this:

```
asabeneh@Asabeneh:~$ pip install pip
```

Check if pip is installed by writing

```
pip –version
```

```
asabeneh@Asabeneh:~$ pip --version
pip 21.1.3 from /usr/local/lib/python3.7/site-packages/pip
(python 3.9.6)
```

As you can see, I am using pip version 21.1.3, if you see some number a bit below or above that, means you have pip installed.

Let us check some of the packages used in the Python community for different purposes. Just to let you know that there are lots of packages available for use with different applications.

Installing packages using pip

Let us try to install *numpy*, called numeric python. It is one of the most popular packages in machine learning and data science community.

- NumPy is the fundamental package for scientific computing with Python. It contains among other things:
    - a powerful N-dimensional array object
    - sophisticated (broadcasting) functions
    - tools for integrating C/C++ and Fortran code
    - useful linear algebra, Fourier transform, and random number capabilities

```
asabeneh@Asabeneh:~$ pip install numpy
```

Let us start using numpy. Open your python interactive shell, write python and then import numpy as follows:

```
asabeneh@Asabeneh:~$ python
Python 3.9.6 (default, Jun 28 2021, 15:26:21)
[Clang 11.0.0 (clang-1100.0.33.8)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> import numpy
>>> numpy.version.version
'1.20.1'
>>> lst = [1, 2, 3,4, 5]
>>> np_arr = numpy.array(lst)
>>> np_arr
array([1, 2, 3, 4, 5])
>>> len(np_arr)
5
>>> np_arr * 2
array([ 2,  4,  6,  8, 10])
>>> np_arr  + 2
array([3, 4, 5, 6, 7])
>>>
```

Pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language. Let us install the big brother of numpy, *pandas*:

```
asabeneh@Asabeneh:~$ pip install pandas
```

```
asabeneh@Asabeneh:~$ python
Python 3.9.6 (default, Jun 28 2021, 15:26:21)
[Clang 11.0.0 (clang-1100.0.33.8)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> import pandas
```

This section is not about numpy nor pandas, here we are trying to learn how to install packages and how to import them. If it is needed, we will talk about different packages in other sections.

Let us import a web browser module, which can help us to open any website. We do not need to install this module, it is already installed by default with Python 3. For instance if you like to open any number of websites at any time or if you like to schedule something, this *webbrowser* module can be used.

```
import webbrowser # web browser module to open websites

# list of urls: python
url_lists = [
    'http://www.python.org',
```

```
    'https://www.linkedin.com/in/asabeneh/',
    'https://github.com/Asabeneh',
    'https://twitter.com/Asabeneh',
]

# opens the above list of websites in a different tab
for url in url_lists:
    webbrowser.open_new_tab(url)
```

Uninstalling Packages

If you do not like to keep the installed packages, you can remove them using the following command.

```
pip uninstall packagename
```

List of Packages

To see the installed packages on our machine. We can use pip followed by list.

```
pip list
```

Show Package

To show information about a package

```
pip show packagename
```

```
asabeneh@Asabeneh:~$ pip show pandas
Name: pandas
Version: 1.2.3
Summary: Powerful data structures for data analysis, time
series, and statistics
Home-page: http://pandas.pydata.org
Author: None
Author-email: None
License: BSD
Location: /usr/local/lib/python3.7/site-packages
Requires: python-dateutil, pytz, numpy
Required-by:
```

If we want even more details, just add --verbose

```
asabeneh@Asabeneh:~$ pip show --verbose pandas
Name: pandas
Version: 1.2.3
Summary: Powerful data structures for data analysis, time
series, and statistics
Home-page: http://pandas.pydata.org
Author: None
Author-email: None
License: BSD
Location: /usr/local/lib/python3.7/site-packages
Requires: numpy, pytz, python-dateutil
Required-by:
Metadata-Version: 2.1
Installer: pip
Classifiers:
  Development Status :: 5 - Production/Stable
  Environment :: Console
  Operating System :: OS Independent
  Intended Audience :: Science/Research
  Programming Language :: Python
  Programming Language :: Python :: 3
  Programming Language :: Python :: 3.5
  Programming Language :: Python :: 3.6
  Programming Language :: Python :: 3.7
  Programming Language :: Python :: 3.8
  Programming Language :: Cython
  Topic :: Scientific/Engineering
Entry-points:
  [pandas_plotting_backends]
  matplotlib = pandas:plotting._matplotlib
```

PIP Freeze

Generate installed Python packages with their version and the output is suitable to use it in a requirements file. A requirements.txt file is a file that should contain all the installed Python packages in a Python project.

```
asabeneh@Asabeneh:~$ pip freeze
docutils==0.11
Jinja2==2.7.2
MarkupSafe==0.19
Pygments==1.6
Sphinx==1.2.2
```

The pip freeze gave us the packages used, installed and their version. We use it with requirements.txt file for deployment.

Reading from URL

By now you are familiar with how to read or write on a file located on you local machine. Sometimes, we would like to read from a website using url or from an API. API stands for Application Program Interface. It is a means to exchange structured data between servers primarily as json data. To open a network connection, we need a package called *requests* - it allows to open a network connection and to implement CRUD(create, read, update and delete) operations. In this section, we will cover only reading ore getting part of a CRUD.

Let us install *requests*:

```
asabeneh@Asabeneh:~$ pip install requests
```

We will see *get*, *status_code*, *headers*, *text* and *json* methods in *requests* module:

- *get()*: to open a network and fetch data from url - it returns a response object
- *status_code*: After we fetched data, we can check the status of the operation (success, error, etc)
- *headers*: To check the header types
- *text*: to extract the text from the fetched response object
- *json*: to extract json data Let's read a txt file from this website, https://www.w3.org/TR/PNG/iso_8859-1.txt.

```
import requests # importing the request module

url = 'https://www.w3.org/TR/PNG/iso_8859-1.txt' # text from a
website

response = requests.get(url) # opening a network and fetching
a data
print(response)
print(response.status_code) # status code, success:200
print(response.headers)     # headers information
print(response.text) # gives all the text from the page
```

```
<Response [200]>
200
{'date': 'Sun, 08 Dec 2019 18:00:31 GMT', 'last-modified':
'Fri, 07 Nov 2003 05:51:11 GMT', 'etag': '"17e9-
3cb82080711c0;50c0b26855880-gzip"', 'accept-ranges': 'bytes',
'cache-control': 'max-age=31536000', 'expires': 'Mon, 07 Dec
2020 18:00:31 GMT', 'vary': 'Accept-Encoding', 'content-
encoding': 'gzip', 'access-control-allow-origin': '*',
'content-length': '1616', 'content-type': 'text/plain',
'strict-transport-security': 'max-age=15552000;
includeSubdomains; preload', 'content-security-policy':
'upgrade-insecure-requests'}
```

- Let us read from an API. API stands for Application Program Interface. It is a means to exchange structure data between servers primarily a json data. An example of an API:https://restcountries.eu/rest/v2/all. Let us read this API using *requests* module.

```
import requests
url = 'https://restcountries.eu/rest/v2/all'  # countries api
response = requests.get(url)  # opening a network and fetching
a data
print(response) # response object
print(response.status_code)  # status code, success:200
countries = response.json()
print(countries[:1])  # we sliced only the first country,
remove the slicing to see all countries
```

```
<Response [200]>
200
[{'alpha2Code': 'AF',
  'alpha3Code': 'AFG',
  'altSpellings': ['AF', 'Afġānistān'],
  'area': 652230.0,
  'borders': ['IRN', 'PAK', 'TKM', 'UZB', 'TJK', 'CHN'],
  'callingCodes': ['93'],
  'capital': 'Kabul',
  'cioc': 'AFG',
  'currencies': [{'code': 'AFN', 'name': 'Afghan afghani',
'symbol': '؋'}],
  'demonym': 'Afghan',
  'flag': 'https://restcountries.eu/data/afg.svg',
  'gini': 27.8,
  'languages': [{'iso639_1': 'ps',
                 'iso639_2': 'pus',
                 'name': 'Pashto',
                 'nativeName': 'پښتو'},
                {'iso639_1': 'uz',
                 'iso639_2': 'uzb',
                 'name': 'Uzbek',
                 'nativeName': 'Oʻzbek'},
                {'iso639_1': 'tk',
                 'iso639_2': 'tuk',
                 'name': 'Turkmen',
                 'nativeName': 'Türkmen'}],
  'latlng': [33.0, 65.0],
  'name': 'Afghanistan',
  'nativeName': 'افغانستان',
  'numericCode': '004',
  'population': 27657145,
  'region': 'Asia',
  'regionalBlocs': [{'acronym': 'SAARC',
```

```
                      'name': 'South Asian Association for
Regional Cooperation',
                      'otherAcronyms': [],
                      'otherNames': []}],
  'subregion': 'Southern Asia',
  'timezones': ['UTC+04:30'],
  'topLevelDomain': ['.af'],
  'translations': {'br': 'Afeganistão',
                   'de': 'Afghanistan',
                   'es': 'Afganistán',
                   'fa': 'افغانستان',
                   'fr': 'Afghanistan',
                   'hr': 'Afganistan',
                   'it': 'Afghanistan',
                   'ja': 'アフガニスタン',
                   'nl': 'Afghanistan',
                   'pt': 'Afeganistão'}}]
```

We use *json()* method from response object, if the we are fetching JSON data. For txt, html, xml and other file formats we can use *text*.

Creating a Package

We organize a large number of files in different folders and sub-folders based on some criteria, so that we can find and manage them easily. As you know, a module can contain multiple objects, such as classes, functions, etc. A package can contain one or more relevant modules. A package is actually a folder containing one or more module files. Let us create a package named mypackage, using the following steps:

Create a new folder named mypacakge inside 30DaysOfPython folder Create an empty **init**.py file in the mypackage folder. Create modules arithmetic.py and greet.py with following code:

```
# mypackage/arithmetics.py
# arithmetics.py
def add_numbers(*args):
    total = 0
    for num in args:
        total += num
    return total


def subtract(a, b):
    return (a - b)


def multiple(a, b):
```

```
    return a * b


def division(a, b):
    return a / b


def remainder(a, b):
    return a % b


def power(a, b):
    return a ** b
```

```
# mypackage/greet.py
# greet.py
def greet_person(firstname, lastname):
    return f'{firstname} {lastname}, welcome to 30DaysOfPython
Challenge!'
```

The folder structure of your package should look like this:

```
— mypackage
    ├── __init__.py
    ├── arithmetic.py
    └── greet.py
```

Now let's open the python interactive shell and try the package we have created:

```
asabeneh@Asabeneh:~/Desktop/30DaysOfPython$ python
Python 3.9.6 (default, Jun 28 2021, 15:26:21)
[Clang 11.0.0 (clang-1100.0.33.8)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> from mypackage import arithmetics
>>> arithmetics.add_numbers(1, 2, 3, 5)
11
>>> arithmetics.subtract(5, 3)
2
>>> arithmetics.multiple(5, 3)
15
>>> arithmetics.division(5, 3)
1.6666666666666667
>>> arithmetics.remainder(5, 3)
2
>>> arithmetics.power(5, 3)
125
>>> from mypackage import greet
>>> greet.greet_person('Asabeneh', 'Yetayeh')
'Asabeneh Yetayeh, welcome to 30DaysOfPython Challenge!'
>>>
```

As you can see our package works perfectly. The package folder contains a special file called **init**.py - it stores the package's content. If we put **init**.py in the package folder, python start recognizes it as a package. The **init**.py exposes specified resources from its modules to be imported to other python files. An empty **init**.py file makes all functions available when a package is imported. The **init**.py is essential for the folder to be recognized by Python as a package.

Further Information About Packages

- Database
    - SQLAlchemy or SQLObject - Object oriented access to several different database systems
        - *pip install SQLAlchemy*
- Web Development
    - Django - High-level web framework.
        - *pip install django*
    - Flask - micro framework for Python based on Werkzeug, Jinja 2. (It's BSD licensed)
        - *pip install flask*
- HTML Parser
    - Beautiful Soup - HTML/XML parser designed for quick turnaround projects like screen-scraping, will accept bad markup.
        - *pip install beautifulsoup4*
    - PyQuery - implements jQuery in Python; faster than BeautifulSoup, apparently.
- XML Processing
    - ElementTree - The Element type is a simple but flexible container object, designed to store hierarchical data structures, such as simplified XML infosets, in memory. --Note: Python 2.5 and up has ElementTree in the Standard Library
- GUI
    - PyQt - Bindings for the cross-platform Qt framework.
    - TkInter - The traditional Python user interface toolkit.
- Data Analysis, Data Science and Machine learning

    - Numpy: Numpy(numeric python) is known as one of the most popular machine learning library in Python.
    - Pandas: is a data analysis, data science and a machine learning library in Python that provides data structures of high-level and a wide variety of tools for analysis.
    - SciPy: SciPy is a machine learning library for application developers and engineers. SciPy library contains modules for optimization, linear algebra, integration, image processing, and statistics.
    - Scikit-Learn: It is NumPy and SciPy. It is considered as one of the best libraries for working with complex data.
    - TensorFlow: is a machine learning library built by Google.
    - Keras: is considered as one of the coolest machine learning libraries in Python. It provides an easier mechanism to express neural networks. Keras also provides some of the best utilities for compiling models, processing data-sets, visualization of graphs, and much more.
- Network:
    - requests: is a package which we can use to send requests to a server(GET, POST, DELETE, PUT)
        - *pip install requests*

⚽ You are always progressing and you are a head of 20 steps to your way to greatness. Now do some exercises for your brain and muscles.

Exercises: Day 20

1. Read this url and find the 10 most frequent words. romeo_and_juliet = 'http://www.gutenberg.org/files/1112/1112.txt'
2. Read the cats API and cats_api = 'https://api.thecatapi.com/v1/breeds' and find :
    i. the min, max, mean, median, standard deviation of cats' weight in metric units.
    ii. the min, max, mean, median, standard deviation of cats' lifespan in years.
    iii. Create a frequency table of country and breed of cats
3. Read the countries API and find
    i. the 10 largest countries
    ii. the 10 most spoken languages
    iii. the total number of languages in the countries API
4. UCI is one of the most common places to get data sets for data science and machine learning. Read the content of UCL (https://archive.ics.uci.edu/ml/datasets.php). Without additional libraries it will be difficult, so you may try it with BeautifulSoup4

🎉 CONGRATULATIONS ! 🎉

DAY-21 CLASSES AND OBJECTS

Classes and Objects

Python is an object oriented programming language. Everything in Python is an object, with its properties and methods. A number, string, list, dictionary, tuple, set etc. used in a program is an object of a corresponding built-in class. We create class to create an object. A class is like an object constructor, or a "blueprint" for creating objects. We instantiate a class to create an object. The class defines attributes and the behavior of the object, while the object, on the other hand, represents the class.

We have been working with classes and objects right from the beginning of this challenge unknowingly. Every element in a Python program is an object of a class. Let us check if everything in python is a class:

```
asabeneh@Asabeneh:~$ python
Python 3.9.6 (default, Jun 28 2021, 15:26:21)
[Clang 11.0.0 (clang-1100.0.33.8)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> num = 10
>>> type(num)
<class 'int'>
>>> string = 'string'
>>> type(string)
<class 'str'>
>>> boolean = True
>>> type(boolean)
<class 'bool'>
>>> lst = []
>>> type(lst)
<class 'list'>
>>> tpl = ()
>>> type(tpl)
<class 'tuple'>
>>> set1 = set()
>>> type(set1)
<class 'set'>
>>> dct = {}
>>> type(dct)
<class 'dict'>
```

Creating a Class

To create a class we need the key word **class** followed by the name and colon. Class name should be **CamelCase**.

```
# syntax
class ClassName:
  code goes here
```

Example:

```
class Person:
  pass
print(Person)
```

```
<__main__.Person object at 0x10804e510>
```

Creating an Object

We can create an object by calling the class.

```
p = Person()
print(p)
```

Class Constructor

In the examples above, we have created an object from the Person class. However, a class without a constructor is not really useful in real applications. Let us use constructor function to make our class more useful. Like the constructor function in Java or JavaScript, Python has also a built-in **init**() constructor function.
The **init** constructor function has self parameter which is a reference to the current instance of the class **Examples:**

```
class Person:
      def __init__ (self, name):
        # self allows to attach parameter to the class
          self.name =name

p = Person('Asabeneh')
print(p.name)
print(p)
```

```
# output
Asabeneh
<__main__.Person object at 0x2abf46907e80>
```

Let us add more parameters to the constructor function.

```
class Person:
```

```
      def __init__(self, firstname, lastname, age, country,
city):
          self.firstname = firstname
          self.lastname = lastname
          self.age = age
          self.country = country
          self.city = city


p = Person('Asabeneh', 'Yetayeh', 250, 'Finland', 'Helsinki')
print(p.firstname)
print(p.lastname)
print(p.age)
print(p.country)
print(p.city)
```

```
# output
Asabeneh
Yetayeh
250
Finland
Helsinki
```

Object Methods

Objects can have methods. The methods are functions which belong to the object.

Example:

```
class Person:
      def __init__(self, firstname, lastname, age, country,
city):
          self.firstname = firstname
          self.lastname = lastname
          self.age = age
          self.country = country
          self.city = city
      def person_info(self):
        return f'{self.firstname} {self.lastname} is
{self.age} years old. He lives in {self.city}, {self.country}'

p = Person('Asabeneh', 'Yetayeh', 250, 'Finland', 'Helsinki')
print(p.person_info())
```

output

```
Asabeneh Yetayeh is 250 years old. He lives in Helsinki,
Finland
```

Object Default Methods

Sometimes, you may want to have a default values for your object methods. If we give default values for the parameters in the constructor, we can avoid errors when we call or instantiate our class without parameters. Let's see how it looks:

Example:

```
class Person:
      def __init__(self, firstname='Asabeneh',
lastname='Yetayeh', age=250, country='Finland',
city='Helsinki'):
          self.firstname = firstname
          self.lastname = lastname
          self.age = age
          self.country = country
          self.city = city

      def person_info(self):
        return f'{self.firstname} {self.lastname} is
{self.age} years old. He lives in {self.city},
{self.country}.'

p1 = Person()
print(p1.person_info())
p2 = Person('John', 'Doe', 30, 'Nomanland', 'Noman city')
print(p2.person_info())
```

```
# output
Asabeneh Yetayeh is 250 years old. He lives in Helsinki,
Finland.
John Doe is 30 years old. He lives in Noman city, Nomanland.
```

Method to Modify Class Default Values

In the example below, the person class, all the constructor parameters have default values. In addition to that, we have skills parameter, which we can access using a method. Let us create add_skill method to add skills to the skills list.

```
class Person:
```

```
      def __init__(self, firstname='Asabeneh',
lastname='Yetayeh', age=250, country='Finland',
city='Helsinki'):
          self.firstname = firstname
          self.lastname = lastname
          self.age = age
          self.country = country
          self.city = city
          self.skills = []

      def person_info(self):
        return f'{self.firstname} {self.lastname} is
{self.age} years old. He lives in {self.city},
{self.country}.'
      def add_skill(self, skill):
          self.skills.append(skill)

p1 = Person()
print(p1.person_info())
p1.add_skill('HTML')
p1.add_skill('CSS')
p1.add_skill('JavaScript')
p2 = Person('John', 'Doe', 30, 'Nomanland', 'Noman city')
print(p2.person_info())
print(p1.skills)
print(p2.skills)
```

```
# output
Asabeneh Yetayeh is 250 years old. He lives in Helsinki,
Finland.
John Doe is 30 years old. He lives in Noman city, Nomanland.
['HTML', 'CSS', 'JavaScript']
[]
```

Inheritance

Using inheritance we can reuse parent class code. Inheritance allows us to define a class that inherits all the methods and properties from parent class. The parent class or super or base class is the class which gives all the methods and properties. Child class is the class that inherits from another or parent class. Let us create a student class by inheriting from person class.

```
class Student(Person):
    pass


s1 = Student('Eyob', 'Yetayeh', 30, 'Finland', 'Helsinki')
s2 = Student('Lidiya', 'Teklemariam', 28, 'Finland', 'Espoo')
```

```
print(s1.person_info())
s1.add_skill('JavaScript')
s1.add_skill('React')
s1.add_skill('Python')
print(s1.skills)

print(s2.person_info())
s2.add_skill('Organizing')
s2.add_skill('Marketing')
s2.add_skill('Digital Marketing')
print(s2.skills)
```

```
output
Eyob Yetayeh is 30 years old. He lives in Helsinki, Finland.
['JavaScript', 'React', 'Python']
Lidiya Teklemariam is 28 years old. He lives in Espoo,
Finland.
['Organizing', 'Marketing', 'Digital Marketing']
```

We did not call the **init**() constructor in the child class. If we didn't call it then we can still access all the properties from the parent. But if we do call the constructor we can access the parent properties by calling *super*.
We can add a new method to the child or we can override the parent class methods by creating the same method name in the child class. When we add the **init**() function, the child class will no longer inherit the parent's **init**() function.

Overriding parent method

```
class Student(Person):
    def __init__ (self, firstname='Asabeneh',
lastname='Yetayeh',age=250, country='Finland',
city='Helsinki', gender='male'):
        self.gender = gender
        super().__init__(firstname, lastname,age, country,
city)
    def person_info(self):
        gender = 'He' if self.gender =='male' else 'She'
        return f'{self.firstname} {self.lastname} is
{self.age} years old. {gender} lives in {self.city},
{self.country}.'

s1 = Student('Eyob', 'Yetayeh', 30, 'Finland',
'Helsinki','male')
s2 = Student('Lidiya', 'Teklemariam', 28, 'Finland', 'Espoo',
'female')
print(s1.person_info())
s1.add_skill('JavaScript')
```

```
s1.add_skill('React')
s1.add_skill('Python')
print(s1.skills)

print(s2.person_info())
s2.add_skill('Organizing')
s2.add_skill('Marketing')
s2.add_skill('Digital Marketing')
print(s2.skills)

Eyob Yetayeh is 30 years old. He lives in Helsinki, Finland.
['JavaScript', 'React', 'Python']
Lidiya Teklemariam is 28 years old. She lives in Espoo,
Finland.
['Organizing', 'Marketing', 'Digital Marketing']
```

We can use super() built-in function or the parent name Person to automatically inherit the methods and properties from its parent. In the example above we override the parent method. The child method has a different feature, it can identify, if the gender is male or female and assign the proper pronoun(He/She).

⚽ Now, you are fully charged with a super power of programming. Now do some exercises for your brain and muscles.

💻 Exercises: Day 21

Exercises: Level 1

1. Python has the module called *statistics* and we can use this module to do all the statistical calculations. However, to learn how to make function and reuse function let us try to develop a program, which calculates the measure of central tendency of a sample (mean, median, mode) and measure of variability (range, variance, standard deviation). In addition to those measures, find the min, max, count, percentile, and frequency distribution of the sample. You can create a class called Statistics and create all the functions that do statistical calculations as methods for the Statistics class. Check the output below.

```
ages = [31, 26, 34, 37, 27, 26, 32, 32, 26, 27, 27, 24, 32,
33, 27, 25, 26, 38, 37, 31, 34, 24, 33, 29, 26]

print('Count:', data.count()) # 25
print('Sum: ', data.sum()) # 744
print('Min: ', data.min()) # 24
print('Max: ', data.max()) # 38
print('Range: ', data.range() # 14
print('Mean: ', data.mean()) # 30
print('Median: ', data.median()) # 29
print('Mode: ', data.mode()) # {'mode': 26, 'count': 5}
print('Standard Deviation: ', data.std()) # 4.2
print('Variance: ', data.var()) # 17.5
print('Frequency Distribution: ', data.freq_dist()) # [(20.0,
26), (16.0, 27), (12.0, 32), (8.0, 37), (8.0, 34), (8.0, 33),
(8.0, 31), (8.0, 24), (4.0, 38), (4.0, 29), (4.0, 25)]
```

```
# you output should look like this
print(data.describe())
Count: 25
Sum:  744
Min:  24
Max:  38
Range:  14
Mean:  30
Median:  29
Mode:  (26, 5)
Variance:  17.5
Standard Deviation:  4.2
Frequency Distribution: [(20.0, 26), (16.0, 27), (12.0, 32),
(8.0, 37), (8.0, 34), (8.0, 33), (8.0, 31), (8.0, 24), (4.0,
38), (4.0, 29), (4.0, 25)]
```

Exercises: Level 2

1. Create a class called PersonAccount. It has firstname, lastname, incomes, expenses properties and it has total_income, total_expense, account_info, add_income, add_expense and account_balance methods. Incomes is a set of incomes and its description. The same goes for expenses.

DAY-22 PYTHON WEB SCRAPING

What is Web Scrapping

The internet is full of huge amount of data which can be used for different purposes. To collect this data we need to know how to scrape data from a website.

Web scraping is the process of extracting and collecting data from websites and storing it on a local machine or in a database.

In this section, we will use beautifulsoup and requests package to scrape data. The package version we are using is beautifulsoup 4.

To start scraping websites you need *requests*, *beautifoulSoup4* and a *website*.

```
pip install requests
pip install beautifulsoup4
```

To scrape data from websites, basic understanding of HTML tags and CSS selectors is needed. We target content from a website using HTML tags, classes or/and ids. Let us import the requests and BeautifulSoup module

```
import requests
from bs4 import BeautifulSoup
```

Let us declare url variable for the website which we are going to scrape.

```
import requests
from bs4 import BeautifulSoup
url = 'https://archive.ics.uci.edu/ml/datasets.php'

# Lets use the requests get method to fetch the data from url

response = requests.get(url)
# lets check the status
status = response.status_code
print(status) # 200 means the fetching was successful
```

```
200
```

Using beautifulSoup to parse content from the page

```
import requests
from bs4 import BeautifulSoup
url = 'https://archive.ics.uci.edu/ml/datasets.php'

response = requests.get(url)
content = response.content # we get all the content from the
website
soup = BeautifulSoup(content, 'html.parser') # beautiful soup
will give a chance to parse
print(soup.title) # <title>UCI Machine Learning Repository:
Data Sets</title>
print(soup.title.get_text()) # UCI Machine Learning
Repository: Data Sets
print(soup.body) # gives the whole page on the website
print(response.status_code)

tables = soup.find_all('table', {'cellpadding':'3'})
# We are targeting the table with cellpadding attribute with
the value of 3
# We can select using id, class or HTML tag , for more
information check the beautifulsoup doc
table = tables[0] # the result is a list, we are taking out
data from it
for td in table.find('tr').find_all('td'):
    print(td.text)
```

If you run this code, you can see that the extraction is half done. You can continue doing it because it is part of exercise 1. For reference check the [beautifulsoup documentation]

🍪 You are so special, you are progressing everyday. You are left with only eight days to your way to greatness. Now do some exercises for your brain and muscles.

💻 Exercises: Day 22

1. Scrape the following website and store the data as json file(url = 'http://www.bu.edu/president/boston-university-facts-stats/').
2. Extract the table in this url (https://archive.ics.uci.edu/ml/datasets.php) and change it to a json file
3. Scrape the presidents table and store the data as json(https://en.wikipedia.org/wiki/List_of_presidents_of_the_United_States). The table is not very structured and the scrapping may take very long time.

🎉 CONGRATULATIONS ! 🎉

DAY-23 VIRTUAL ENVIRONMENT

Setting up Virtual Environments

To start with project, it would be better to have a virtual environment. Virtual environment can help us to create an isolated or separate environment. This will help us to avoid conflicts in dependencies across projects. If you write pip freeze on your terminal you will see all the installed packages on your computer. If we use virtualenv, we will access only packages which are specific for that project. Open your terminal and install virtualenv

```
asabeneh@Asabeneh:~$ pip install virtualenv
```

Inside the 30DaysOfPython folder create a flask_project folder.

After installing the virtualenv package go to your project folder and create a virtual env by writing:

For Mac/Linux:

```
asabeneh@Asabeneh:~/Desktop/30DaysOfPython/flask_project\$
virtualenv venv
```

For Windows:

```
C:\Users\User\Documents\30DaysOfPython\flask_project>python -m
venv venv
```

I prefer to call the new project venv, but feel free to name it differently. Let us check if the the venv was created by using ls (or dir for windows command prompt) command.

```
asabeneh@Asabeneh:~/Desktop/30DaysOfPython/flask_project$ ls
venv/
```

Let us activate the virtual environment by writing the following command at our project folder.

For Mac/Linux:

```
asabeneh@Asabeneh:~/Desktop/30DaysOfPython/flask_project$
source venv/bin/activate
```

Activation of the virtual environment in Windows may very on Windows Power shell and git bash.

For Windows Power Shell:

```
C:\Users\User\Documents\30DaysOfPython\flask_project>
venv\Scripts\activate
```

For Windows Git bash:

```
C:\Users\User\Documents\30DaysOfPython\flask_project>
venv\Scripts\. activate
```

After you write the activation command, your project directory will start with venv. See the example below.

```
(venv)
asabeneh@Asabeneh:~/Desktop/30DaysOfPython/flask_project$
```

Now, lets check the available packages in this project by writing pip freeze. You will not see any packages.

We are going to do a small flask project so let us install flask package to this project.

```
(venv)
asabeneh@Asabeneh:~/Desktop/30DaysOfPython/flask_project$ pip
install Flask
```

Now, let us write pip freeze to see a list of installed packages in the project:

```
(venv)
asabeneh@Asabeneh:~/Desktop/30DaysOfPython/flask_project$ pip
freeze
Click==7.0
Flask==1.1.1
itsdangerous==1.1.0
Jinja2==2.10.3
MarkupSafe==1.1.1
Werkzeug==0.16.0
```

When you finish you should dactivate active project using *deactivate*.

```
(venv) asabeneh@Asabeneh:~/Desktop/30DaysOfPython$ deactivate
```

The necessary modules to work with flask are installed. Now, your project directory is ready for a flask project. You should include the venv to your .gitignore file not to push it to github.

💻 Exercises: Day 23

1. Create a project directory with a virtual environment based on the example given above.

🎉 CONGRATULATIONS ! 🎉

DAY-24 STATISTICS

Python for Statistical Analysis

Statistics

Statistics is the discipline that studies
the *collection*, *organization*, *displaying*, *analysing*, *interpretation* and *presentation* of data. Statistics is a branch of Mathematics that is recommended to be a prerequisite for data science and machine learning. Statistics is a very broad field but we will focus in this section only on the most relevant part. After completing this challenge, you may go onto the web development, data analysis, machine learning and data science path. Whatever path you may follow, at some point in your career you will get data which you may work on. Having some statistical knowledge will help you to make decisions based on data, *data tells as they say*.

Data

What is data? Data is any set of characters that is gathered and translated for some purpose, usually analysis. It can be any character, including text and numbers, pictures, sound, or video. If data is not put in a context, it doesn't make any sense to a human or computer. To make sense from data we need to work on the data using different tools.

The work flow of data analysis, data science or machine learning starts from data. Data can be provided from some data source or it can be created. There are structured and unstructured data.

Data can be found in small or big format. Most of the data types we will get have been covered in the file handling section.

Statistics Module

The Python *statistics* module provides functions for calculating mathematical statistics of numerical data. The module is not intended to be a competitor to third-party libraries such as NumPy, SciPy, or proprietary full-featured statistics packages aimed at professional statisticians such as Minitab, SAS and Matlab. It is aimed at the level of graphing and scientific calculators.

NumPy

In the first section we defined Python as a great general-purpose programming language on its own, but with the help of other popular libraries as(numpy, scipy, matplotlib, pandas etc) it becomes a powerful environment for scientific computing.

NumPy is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with arrays.

So far, we have been using vscode but from now on I would recommend using Jupyter Notebook. To access jupyter notebook let's install anaconda. If you are using anaconda most of the common packages are included and you don't have install packages if you installed anaconda.

```
asabeneh@Asabeneh:~/Desktop/30DaysOfPython$ pip install numpy
```

Importing NumPy

Jupyter notebook is available if your are in favor of jupyter notebook

```
    # How to import numpy
    import numpy as np
    # How to check the version of the numpy package
    print('numpy:', np.__version__)
    # Checking the available methods
    print(dir(np))
```

Creating numpy array using

Creating int numpy arrays

```
    # Creating python List
    python_list = [1,2,3,4,5]

    # Checking data types
    print('Type:', type (python_list)) # <class 'list'>
    #
    print(python_list) # [1, 2, 3, 4, 5]

    two_dimensional_list = [[0,1,2], [3,4,5], [6,7,8]]

    print(two_dimensional_list)  # [[0, 1, 2], [3, 4, 5], [6,
7, 8]]

    # Creating Numpy(Numerical Python) array from python list

    numpy_array_from_list = np.array(python_list)
    print(type (numpy_array_from_list))   # <class
'numpy.ndarray'>
    print(numpy_array_from_list) # array([1, 2, 3, 4, 5])
```

Creating float numpy arrays

Creating a float numpy array from list with a float data type parameter

```
    # Python list
    python_list = [1,2,3,4,5]

    numy_array_from_list2 = np.array(python_list, dtype=float)
    print(numy_array_from_list2) # array([1., 2., 3., 4., 5.])
```

Creating boolean numpy arrays

Creating a boolean a numpy array from list

```
    numpy_bool_array = np.array([0, 1, -1, 0, 0], dtype=bool)
    print(numpy_bool_array) # array([False,  True,  True,
False, False])
```

Creating multidimensional array using numpy

A numpy array may have one or multiple rows and columns

```
    two_dimensional_list = [[0,1,2], [3,4,5], [6,7,8]]
    numpy_two_dimensional_list =
np.array(two_dimensional_list)
    print(type (numpy_two_dimensional_list))
    print(numpy_two_dimensional_list)
```

```
 <class 'numpy.ndarray'>
    [[0 1 2]
     [3 4 5]
     [6 7 8]]
```

Converting numpy array to list

```
# We can always convert an array back to a python list using
tolist().
np_to_list = numpy_array_from_list.tolist()
print(type (np_to_list))
print('one dimensional array:', np_to_list)
print('two dimensional array: ',
numpy_two_dimensional_list.tolist())
```

```
 <class 'list'>
    one dimensional array: [1, 2, 3, 4, 5]
    two dimensional array:  [[0, 1, 2], [3, 4, 5], [6, 7, 8]]
```

```
Creating numpy array from tuple
# Numpy array from tuple
# Creating tuple in Python
```

```
python_tuple = (1,2,3,4,5)
print(type (python_tuple)) # <class 'tuple'>
print('python_tuple: ', python_tuple) # python_tuple:  (1, 2,
3, 4, 5)

numpy_array_from_tuple = np.array(python_tuple)
print(type (numpy_array_from_tuple)) # <class 'numpy.ndarray'>
print('numpy_array_from_tuple: ', numpy_array_from_tuple) #
numpy_array_from_tuple:  [1 2 3 4 5]
```

Shape of numpy array

The shape method provide the shape of the array as a tuple. The first is the row and the second is the column. If the array is just one dimensional it returns the size of the array.

```
    nums = np.array([1, 2, 3, 4, 5])
    print(nums)
    print('shape of nums: ', nums.shape)
    print(numpy_two_dimensional_list)
    print('shape of numpy_two_dimensional_list: ',
numpy_two_dimensional_list.shape)
    three_by_four_array = np.array([[0, 1, 2, 3],
        [4,5,6,7],
        [8,9,10, 11]])
    print(three_by_four_array.shape)
```

```
    [1 2 3 4 5]
    shape of nums:  (5,)
    [[0 1 2]
     [3 4 5]
     [6 7 8]]
    shape of numpy_two_dimensional_list:  (3, 3)
    (3, 4)
```

Data type of numpy array

Type of data types: str, int, float, complex, bool, list, None

```
int_lists = [-3, -2, -1, 0, 1, 2,3]
int_array = np.array(int_lists)
float_array = np.array(int_lists, dtype=float)

print(int_array)
print(int_array.dtype)
print(float_array)
print(float_array.dtype)
```

```
    [-3 -2 -1  0  1  2  3]
    int64
    [-3. -2. -1.  0.  1.  2.  3.]
    float64
```

Size of a numpy array

In numpy to know the number of items in a numpy array list we use size

```
numpy_array_from_list = np.array([1, 2, 3, 4, 5])
two_dimensional_list = np.array([[0, 1, 2],
                                 [3, 4, 5],
                                 [6, 7, 8]])

print('The size:', numpy_array_from_list.size) # 5
print('The size:', two_dimensional_list.size)  # 3
```

```
    The size: 5
    The size: 9
```

Mathematical Operation using numpy

NumPy array is not like exactly like python list. To do mathematical operation in Python list we have to loop through the items but numpy can allow to do any mathematical operation without looping. Mathematical Operation:

- Addition (+)
- Subtraction (-)
- Multiplication (*)
- Division (/)
- Modules (%)
- Floor Division(//)
- Exponential(**)

Addition

Mathematical Operation

```
# Addition
```

```
numpy_array_from_list = np.array([1, 2, 3, 4, 5])
print('original array: ', numpy_array_from_list)
ten_plus_original = numpy_array_from_list  + 10
print(ten_plus_original)
```

```
    original array:  [1 2 3 4 5]
    [11 12 13 14 15]
```

Subtraction

```
# Subtraction
numpy_array_from_list = np.array([1, 2, 3, 4, 5])
print('original array: ', numpy_array_from_list)
ten_minus_original = numpy_array_from_list  - 10
print(ten_minus_original)
```

```
  original array:  [1 2 3 4 5]
    [-9 -8 -7 -6 -5]
```

Multiplication

```
# Multiplication
numpy_array_from_list = np.array([1, 2, 3, 4, 5])
print('original array: ', numpy_array_from_list)
ten_times_original = numpy_array_from_list * 10
print(ten_times_original)
```

```
   original array:  [1 2 3 4 5]
    [10 20 30 40 50]
```

Division

```
# Division
numpy_array_from_list = np.array([1, 2, 3, 4, 5])
print('original array: ', numpy_array_from_list)
ten_times_original = numpy_array_from_list / 10
print(ten_times_original)
```

```
    original array:  [1 2 3 4 5]
    [0.1 0.2 0.3 0.4 0.5]
```

Modulus

```
# Modulus; Finding the remainder
numpy_array_from_list = np.array([1, 2, 3, 4, 5])
```

```
print('original array: ', numpy_array_from_list)
ten_times_original = numpy_array_from_list % 3
print(ten_times_original)
```

```
   original array:  [1 2 3 4 5]
    [1 2 0 1 2]
```

Floor Division

```
# Floor division: the division result without the remainder
numpy_array_from_list = np.array([1, 2, 3, 4, 5])
print('original array: ', numpy_array_from_list)
ten_times_original = numpy_array_from_list // 10
print(ten_times_original)
```

Exponential

```
# Exponential is finding some number the power of another:
numpy_array_from_list = np.array([1, 2, 3, 4, 5])
print('original array: ', numpy_array_from_list)
ten_times_original = numpy_array_from_list  ** 2
print(ten_times_original)
```

```
    original array:  [1 2 3 4 5]
    [ 1  4  9 16 25]
```

Checking data types

```
#Int,  Float numbers
numpy_int_arr = np.array([1,2,3,4])
numpy_float_arr = np.array([1.1, 2.0,3.2])
numpy_bool_arr = np.array([-3, -2, 0, 1,2,3], dtype='bool')

print(numpy_int_arr.dtype)
print(numpy_float_arr.dtype)
print(numpy_bool_arr.dtype)
```

```
    int64
    float64
    bool
```

Converting types

We can convert the data types of numpy array

1. Int to Float

```
numpy_int_arr = np.array([1,2,3,4], dtype = 'float')
numpy_int_arr
```

```
array([1., 2., 3., 4.])
```

2. Float to Int

```
numpy_int_arr = np.array([1., 2., 3., 4.], dtype = 'int')
numpy_int_arr
```

```
   array([1, 2, 3, 4])
```

3. Int ot boolean

```
np.array([-3, -2, 0, 1,2,3], dtype='bool')
```

```
    array([ True,  True, False,  True,  True,  True])
```

4. Int to str

```
numpy_float_list.astype('int').astype('str')
```

```
    array(['1', '2', '3'], dtype='<U21')
```

Multi-dimensional Arrays

```
# 2 Dimension Array
two_dimension_array = np.array([(1,2,3),(4,5,6), (7,8,9)])
print(type (two_dimension_array))
print(two_dimension_array)
print('Shape: ', two_dimension_array.shape)
print('Size:', two_dimension_array.size)
print('Data type:', two_dimension_array.dtype)
```

```
    <class 'numpy.ndarray'>
    [[1 2 3]
     [4 5 6]
     [7 8 9]]
    Shape:  (3, 3)
    Size: 9
    Data type: int64
```

Getting items from a numpy array

```
# 2 Dimension Array
two_dimension_array = np.array([[1,2,3],[4,5,6], [7,8,9]])
first_row = two_dimension_array[0]
```

```
second_row = two_dimension_array[1]
third_row = two_dimension_array[2]
print('First row:', first_row)
print('Second row:', second_row)
print('Third row: ', third_row)
```

```
    First row: [1 2 3]
    Second row: [4 5 6]
    Third row:  [7 8 9]
```

```
first_column= two_dimension_array[:,0]
second_column = two_dimension_array[:,1]
third_column = two_dimension_array[:,2]
print('First column:', first_column)
print('Second column:', second_column)
print('Third column: ', third_column)
print(two_dimension_array)
```

```
    First column: [1 4 7]
    Second column: [2 5 8]
    Third column:  [3 6 9]
    [[1 2 3]
     [4 5 6]
     [7 8 9]]
```

Slicing Numpy array

Slicing in numpy is similar to slicing in python list

```
two_dimension_array = np.array([[1,2,3],[4,5,6], [7,8,9]])
first_two_rows_and_columns = two_dimension_array[0:2, 0:2]
print(first_two_rows_and_columns)
```

```
   [[1 2]
     [4 5]]
```

How to reverse the rows and the whole array?

```
two_dimension_array[::]
```

```
    array([[1, 2, 3],
           [4, 5, 6],
           [7, 8, 9]])
```

Reverse the row and column positions

```
    two_dimension_array = np.array([[1,2,3],[4,5,6], [7,8,9]])
    two_dimension_array[::-1,::-1]
```

```
    array([[9, 8, 7],
           [6, 5, 4],
           [3, 2, 1]])
```

How to represent missing values ?

```
   print(two_dimension_array)
    two_dimension_array[1,1] = 55
    two_dimension_array[1,2] =44
    print(two_dimension_array)
```

```
   [[1 2 3]
     [4 5 6]
     [7 8 9]]
    [[ 1  2  3]
     [ 4 55 44]
     [ 7  8  9]]
    # Numpy Zeroes
    # numpy.zeros(shape, dtype=float, order='C')
    numpy_zeroes = np.zeros((3,3),dtype=int,order='C')
    numpy_zeroes
```

```
 array([[0, 0, 0],
           [0, 0, 0],
           [0, 0, 0]])
```

```
# Numpy Zeroes
numpy_ones = np.ones((3,3),dtype=int,order='C')
print(numpy_ones)
```

```
    [[1 1 1]
     [1 1 1]
     [1 1 1]]
```

```
twoes = numpy_ones * 2
```

```
# Reshape
# numpy.reshape(), numpy.flatten()
first_shape  = np.array([(1,2,3), (4,5,6)])
print(first_shape)
reshaped = first_shape.reshape(3,2)
print(reshaped)
    [[1 2 3]
     [4 5 6]]
    [[1 2]
```

```
     [3 4]
     [5 6]]
```

```
flattened = reshaped.flatten()
flattened
```

```
    array([1, 2, 3, 4, 5, 6])
    ## Horitzontal Stack
    np_list_one = np.array([1,2,3])
    np_list_two = np.array([4,5,6])

    print(np_list_one + np_list_two)

    print('Horizontal Append:', np.hstack((np_list_one,
np_list_two)))
```

```
    [5 7 9]
    Horizontal Append: [1 2 3 4 5 6]
```

```
    ## Vertical Stack
    print('Vertical Append:', np.vstack((np_list_one,
np_list_two)))
```

```
   Vertical Append: [[1 2 3]
     [4 5 6]]
```

Generating Random Numbers

```
    # Generate a random float  number
    random_float = np.random.random()
    random_float
```

```
    0.018929887384753874
```

```
    # Generate a random float  number
    random_floats = np.random.random(5)
    random_floats
```

```
  array([0.26392192, 0.35842215, 0.87908478, 0.41902195,
0.78926418])
```

```
  # Generating a random integers between 0 and 10

    random_int = np.random.randint(0, 11)
    random_int
```

```
    4
```

```
   # Generating a random integers between 2 and 11, and
creating a one row array
    random_int = np.random.randint(2,10, size=4)
    random_int
```

```
    array([8, 8, 8, 2])
```

```
    # Generating a random integers between 0 and 10
    random_int = np.random.randint(2,10, size=(3,3))
    random_int
```

```
  array([[3, 5, 3],
         [7, 3, 6],
         [2, 3, 3]])
```

Generationg random numbers

```
    # np.random.normal(mu, sigma, size)
    normal_array = np.random.normal(79, 15, 80)
    normal_array
```

```
    array([ 89.49990595,  82.06056961, 107.21445842,
38.69307086,
            47.85259157,  93.07381061,  76.40724259,
78.55675184,
            72.17358173,  47.9888899 ,  65.10370622,
76.29696568,
            95.58234254,  68.14897213,  38.75862686,
122.5587927 ,
            67.0762565 ,  95.73990864,  81.97454563,
92.54264805,
            59.37035153,  77.76828101,  52.30752166,
64.43109931,
            62.63695351,  90.04616138,  75.70009094,
49.87586877,
            80.22002414,  68.56708848,  76.27791052,
67.24343975,
            81.86363935,  78.22703433, 102.85737041,
65.15700341,
            84.87033426,  76.7569997 ,  64.61321853,
67.37244562,
            74.4068773 ,  58.65119655,  71.66488727,
53.42458179,
            70.26872028,  60.96588544,  83.56129414,
72.14255326,
```

```
            81.00787609,  71.81264853,  72.64168853,
86.56608717,
            94.94667321,  82.32676973,  70.5165446 ,
85.43061003,
            72.45526212,  87.34681775,  87.69911217,
103.02831489,
            75.28598596,  67.17806893,  92.41274447,
101.06662611,
            87.70013935,  70.73980645,  46.40368207,
50.17947092,
            61.75618542,  90.26191397,  78.63968639,
70.84550744,
            88.91826581, 103.91474733,  66.3064638 ,
79.49726264,
            70.81087439,  83.90130623,  87.58555972,
59.95462521])
```

Numpy and Statistics

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
plt.hist(normal_array, color="grey", bins=50)
```

```
    (array([2., 0., 0., 0., 1., 2., 2., 0., 2., 0., 0., 1.,
2., 2., 1., 4., 3.,
            4., 2., 7., 2., 2., 5., 4., 2., 4., 3., 2., 1.,
5., 3., 0., 3., 2.,
            1., 0., 0., 1., 3., 0., 1., 0., 0., 0., 0., 0.,
0., 0., 0., 1.]),
     array([ 38.69307086,  40.37038529,  42.04769973,
43.72501417,
             45.4023286 ,  47.07964304,  48.75695748,
50.43427191,
             52.11158635,  53.78890079,  55.46621523,
57.14352966,
             58.8208441 ,  60.49815854,  62.17547297,
63.85278741,
             65.53010185,  67.20741628,  68.88473072,
70.56204516,
             72.23935959,  73.91667403,  75.59398847,
77.27130291,
             78.94861734,  80.62593178,  82.30324622,
83.98056065,
             85.65787509,  87.33518953,  89.01250396,
90.6898184 ,
             92.36713284,  94.04444727,  95.72176171,
97.39907615,
```

```
             99.07639058, 100.75370502, 102.43101946,
104.1083339 ,
            105.78564833, 107.46296277, 109.14027721,
110.81759164,
            112.49490608, 114.17222052, 115.84953495,
117.52684939,
            119.20416383, 120.88147826, 122.5587927 ]),
     <a list of 50 Patch objects>)
```

Matrix in numpy

```
four_by_four_matrix = np.matrix(np.ones((4,4), dtype=float))
```

```
four_by_four_matrix
```

```
matrix([[1., 1., 1., 1.],
            [1., 1., 1., 1.],
            [1., 1., 1., 1.],
            [1., 1., 1., 1.]])
```

```
np.asarray(four_by_four_matrix)[2] = 2
four_by_four_matrix
```

```
matrix([[1., 1., 1., 1.],
            [1., 1., 1., 1.],
            [2., 2., 2., 2.],
            [1., 1., 1., 1.]])
```

Numpy numpy.arange()

What is Arrange?

Sometimes, you want to create values that are evenly spaced within a defined interval. For instance, you want to create values from 1 to 10; you can use numpy.arange() function

```
# creating list using range(starting, stop, step)
lst = range(0, 11, 2)
lst
```

```
range(0, 11, 2)

for l in lst:
    print(l)
```

```
    2
```

```
    4
    6
    8
    10
```

```
# Similar to range arange numpy.arange(start, stop, step)
whole_numbers = np.arange(0, 20, 1)
whole_numbers
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13,
14, 15, 16, 17, 18, 19])
```

```
natural_numbers = np.arange(1, 20, 1)
natural_numbers
```

```
odd_numbers = np.arange(1, 20, 2)
odd_numbers
```

```
    array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19])
```

```
even_numbers = np.arange(2, 20, 2)
even_numbers
```

```
    array([ 2,  4,  6,  8, 10, 12, 14, 16, 18])
```

Creating sequence of numbers using linspace

```
# numpy.linspace()
# numpy.logspace() in Python with Example
# For instance, it can be used to create 10 values from 1 to 5
evenly spaced.
np.linspace(1.0, 5.0, num=10)
```

```
    array([1.        , 1.44444444, 1.88888889, 2.33333333,
2.77777778,
           3.22222222, 3.66666667, 4.11111111, 4.55555556, 5.
])
```

```
# not to include the last value in the interval
np.linspace(1.0, 5.0, num=5, endpoint=False)
```

```
array([1. , 1.8, 2.6, 3.4, 4.2])
```

```
# LogSpace
# LogSpace returns even spaced numbers on a log scale.
Logspace has the same parameters as np.linspace.

# Syntax:

# numpy.logspace(start, stop, num, endpoint)

np.logspace(2, 4.0, num=4)
```

```
array([  100.        ,   464.15888336,  2154.43469003, 10000.
])
```

```
# to check the size of an array
x = np.array([1,2,3], dtype=np.complex128)
```

```
x
```

```
    array([1.+0.j, 2.+0.j, 3.+0.j])
```

```
x.itemsize
```

```
16
```

```
# indexing and Slicing NumPy Arrays in Python
np_list = np.array([(1,2,3), (4,5,6)])
np_list
```

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

```
print('First row: ', np_list[0])
print('Second row: ', np_list[1])
```

```
    First row:  [1 2 3]
    Second row:  [4 5 6]
```

```
print('First column: ', np_list[:,0])
print('Second column: ', np_list[:,1])
print('Third column: ', np_list[:,2])
```

```
    First column:  [1 4]
    Second column:  [2 5]
    Third column:  [3 6]
```

NumPy Statistical Functions with Example

NumPy has quite useful statistical functions for finding minimum, maximum, mean, median, percentile,standard deviation and variance, etc from the given elements in the array. The functions are explained as follows − Statistical function Numpy is equipped with the robust statistical function as listed below

- Numpy Functions
    - Min np.min()
    - Max np.max()
    - Mean np.mean()
    - Median np.median()
    - Varience
    - Percentile
    - Standard deviation np.std()

```
np_normal_dis = np.random.normal(5, 0.5, 100)
np_normal_dis
## min, max, mean, median, sd
print('min: ', two_dimension_array.min())
print('max: ', two_dimension_array.max())
print('mean: ',two_dimension_array.mean())
# print('median: ', two_dimension_array.median())
print('sd: ', two_dimension_array.std())
```

```
min:  1
max:  55
mean:  14.777777777777779
sd:  18.913709183069525
```

```
min:  1
max:  55
mean:  14.777777777777779
sd:  18.913709183069525
```

```
print(two_dimension_array)
print('Column with minimum: ',
np.amin(two_dimension_array,axis=0))
print('Column with maximum: ',
np.amax(two_dimension_array,axis=0))
print('=== Row ==')
print('Row with minimum: ',
np.amin(two_dimension_array,axis=1))
print('Row with maximum: ',
np.amax(two_dimension_array,axis=1))
```

```
[[ 1  2  3]
 [ 4 55 44]
 [ 7  8  9]]
Column with minimum:  [1 2 3]
Column with maximum:  [ 7 55 44]
=== Row ==
Row with minimum:  [1 4 7]
Row with maximum:  [ 3 55  9]
```

How to create repeating sequences?

```
a = [1,2,3]

# Repeat whole of 'a' two times
print('Tile:   ', np.tile(a, 2))

# Repeat each element of 'a' two times
print('Repeat: ', np.repeat(a, 2))
```

```
Tile:    [1 2 3 1 2 3]
Repeat:  [1 1 2 2 3 3]
```

How to generate random numbers?

```
# One random number between [0,1)
one_random_num = np.random.random()
one_random_in = np.random
print(one_random_num)
```

```
0.6149403282678213
```

```
0.4763968133790438
```

```
0.4763968133790438
```

```
# Random numbers between [0,1) of shape 2,3
r = np.random.random(size=[2,3])
print(r)
```

```
[[0.13031737 0.4429537  0.1129527 ]
 [0.76811539 0.88256594 0.6754075 ]]
print(np.random.choice(['a', 'e', 'i', 'o', 'u'], size=10))
```

```
['u' 'o' 'o' 'i' 'e' 'e' 'u' 'o' 'u' 'a']
```

```
['i' 'u' 'e' 'o' 'a' 'i' 'e' 'u' 'o' 'i']
```

```
['iueoaieuoi']
```

```
## Random numbers between [0, 1] of shape 2, 2
rand = np.random.rand(2,2)
rand
```

```
array([[0.97992598, 0.79642484],
       [0.65263629, 0.55763145]])
```

```
rand2 = np.random.randn(2,2)
rand2
```

```
array([[ 1.65593322, -0.52326621],
       [ 0.39071179, -2.03649407]])
```

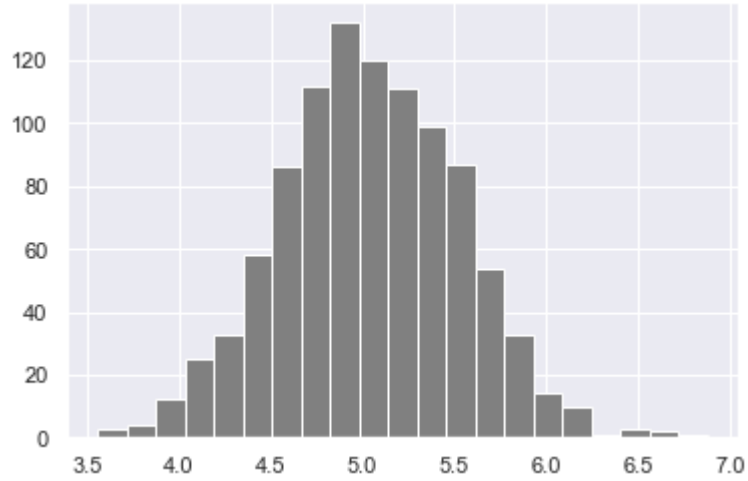
```
# Random integers between [0, 10) of shape 2,5
rand_int = np.random.randint(0, 10, size=[5,3])
rand_int
```

```
array([[0, 7, 5],
       [4, 1, 4],
       [3, 5, 3],
       [4, 3, 8],
       [4, 6, 7]])
```

```
from scipy import stats
np_normal_dis = np.random.normal(5, 0.5, 1000) # mean,
standard deviation, number of samples
np_normal_dis
## min, max, mean, median, sd
print('min: ', np.min(np_normal_dis))
print('max: ', np.max(np_normal_dis))
print('mean: ', np.mean(np_normal_dis))
print('median: ', np.median(np_normal_dis))
print('mode: ', stats.mode(np_normal_dis))
print('sd: ', np.std(np_normal_dis))
```

```
    min:  3.557811005458804
    max:  6.876317743643499
    mean:  5.035832048106663
    median:  5.020161980441937
    mode:  ModeResult(mode=array([3.55781101]),
count=array([1]))
    sd:  0.489682424165213
```

```
plt.hist(np_normal_dis, color="grey", bins=21)
plt.show()
```




```
# numpy.dot(): Dot Product in Python using Numpy
# Dot Product
# Numpy is powerful library for matrices computation. For
instance, you can compute the dot product with np.dot

# Syntax

# numpy.dot(x, y, out=None)
```

Linear Algebra

1. Dot Product

```
## Linear algebra
### Dot product: product of two arrays
f = np.array([1,2,3])
g = np.array([4,5,3])
### 1*4+2*5 + 3*6
np.dot(f, g)  # 23
```

NumPy Matrix Multiplication with np.matmul()

```
### Matmul: matruc product of two arrays
h = [[1,2],[3,4]]
i = [[5,6],[7,8]]
### 1*5+2*7 = 19
np.matmul(h, i)
    array([[19, 22],
           [43, 50]])
```

```
## Determinant 2*2 matrix
### 5*8-7*6np.linalg.det(i)
```

```
np.linalg.det(i)
```

```
-1.999999999999999
```

```
Z = np.zeros((8,8))
Z[1::2,::2] = 1
Z[::2,1::2] = 1
```

```
Z
```

```
array([[0., 1., 0., 1., 0., 1., 0., 1.],
       [1., 0., 1., 0., 1., 0., 1., 0.],
       [0., 1., 0., 1., 0., 1., 0., 1.],
       [1., 0., 1., 0., 1., 0., 1., 0.],
       [0., 1., 0., 1., 0., 1., 0., 1.],
       [1., 0., 1., 0., 1., 0., 1., 0.],
       [0., 1., 0., 1., 0., 1., 0., 1.],
       [1., 0., 1., 0., 1., 0., 1., 0.]])
```

```
new_list = [ x + 2 for x in range(0, 11)]
```

```
new_list
```

```
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

```
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

```
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

```
np_arr = np.array(range(0, 11))
np_arr + 2
```

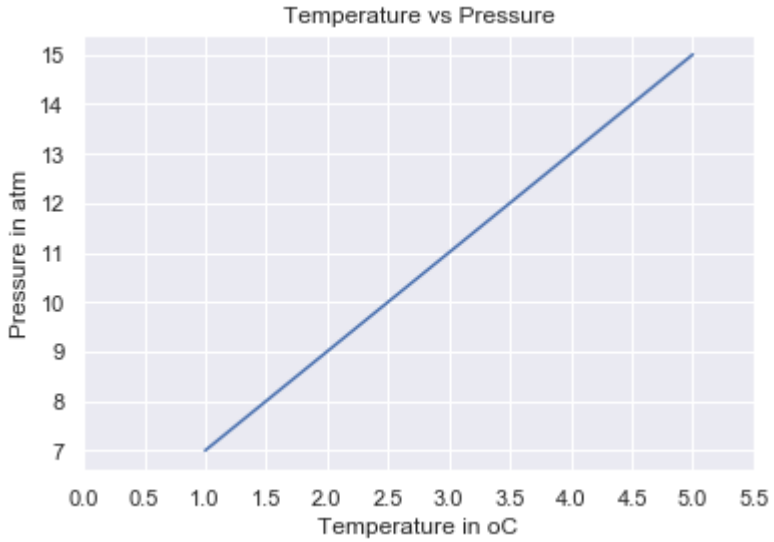
array([2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

We use linear equation for quantities which have linear relationship. Let's see the example below:

```
temp = np.array([1,2,3,4,5])
pressure = temp * 2 + 5
pressure
```

array([7, 9, 11, 13, 15])

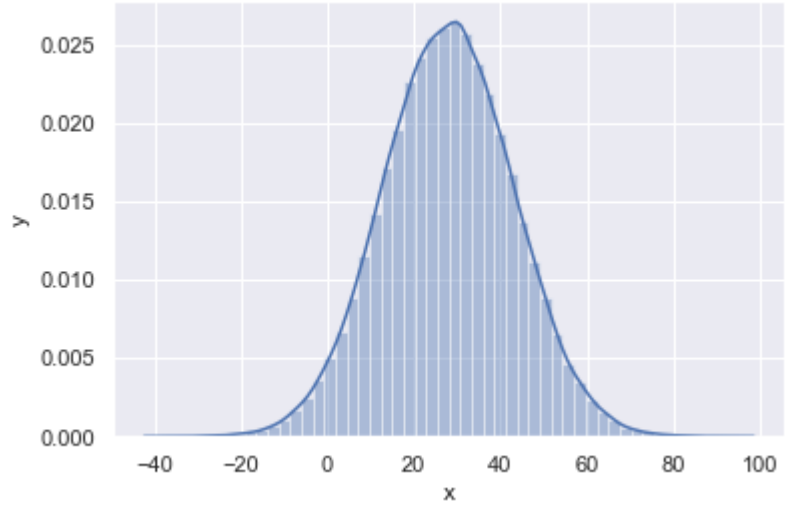
```
plt.plot(temp,pressure)
plt.xlabel('Temperature in oC')
plt.ylabel('Pressure in atm')
plt.title('Temperature vs Pressure')
plt.xticks(np.arange(0, 6, step=0.5))
plt.show()
```



To draw the Gaussian normal distribution using numpy. As you can see below, the numpy can generate random numbers. To create random sample, we need the mean(mu), sigma(standard deviation), mumber of data points.

```
mu = 28
sigma = 15
samples = 100000

x = np.random.normal(mu, sigma, samples)
ax = sns.distplot(x);
ax.set(xlabel="x", ylabel='y')
plt.show()
```



Summary

To summarize, the main differences with python lists are:

1. Arrays support vectorized operations, while lists don’t.

2. Once an array is created, you cannot change its size. You will have to create a new array or overwrite the existing one.
3. Every array has one and only one dtype. All items in it should be of that dtype.
4. An equivalent numpy array occupies much less space than a python list of lists.
5. numpy arrays support boolean indexing.

💻 Exercises: Day 24

1. Repeat all the examples

🎉 CONGRATULATIONS ! 🎉

DAY-25 PANDAS

Pandas

Pandas is an open source, high-performance, easy-to-use data structures and data analysis tools for the Python programming language. Pandas adds data structures and tools designed to work with table-like data which is *Series* and *Data Frames*. Pandas provides tools for data manipulation:

- reshaping
- merging
- sorting
- slicing
- aggregation
- imputation. If you are using anaconda, you do not have install pandas.

Installing Pandas

For Mac:

```
pip install conda
conda install pandas
For Windows:
pip install conda
pip install pandas
```

Pandas data structure is based on *Series* and *DataFrames*.

A *series* is a *column* and a DataFrame is a *multidimensional table* made up of collection of *series*. In order to create a pandas series we should use numpy to create a one dimensional arrays or a python list. Let us see an example of a series:

Names Pandas Series

	Name
0	Asabeneh
1	David
2	John

Countries Series

	Country
0	Finland
1	UK
2	Sweden

Cities Series

	City
0	Helsinki
1	London
2	Stockholm

As you can see, pandas series is just one column of data. If we want to have multiple columns we use data frames. The example below shows pandas DataFrames.

Let us see, an example of a pandas data frame:

	Name	Country	City
0	Asabeneh	Finland	Helsinki
1	David	UK	London
2	John	Sweden	Stockholm

Data frame is a collection of rows and columns. Look at the table below; it has many more columns than the example above:

	Name	Country	City	Weight	Height
0	Asabeneh	Finland	Helsinki	74	173
1	David	UK	London	78	175
2	John	Sweden	Stockholm	69	169

Next, we will see how to import pandas and how to create Series and DataFrames using pandas

Importing Pandas

```
import pandas as pd # importing pandas as pd
import numpy  as np # importing numpy as np
```

Creating Pandas Series with Default Index

```
nums = [1, 2, 3, 4,5]
s = pd.Series(nums)
print(s)
```

```
    0    1
    1    2
    2    3
    3    4
    4    5
    dtype: int64
```

Creating Pandas Series with custom index

```
nums = [1, 2, 3, 4, 5]
s = pd.Series(nums, index=[1, 2, 3, 4, 5])
print(s)
```

```
    1    1
    2    2
    3    3
    4    4
    5    5
    dtype: int64
```

```
fruits = ['Orange','Banana','Mango']
fruits = pd.Series(fruits, index=[1, 2, 3])
print(fruits)
```

```
    1    Orange
    2    Banana
    3    Mango
    dtype: object
```

Creating Pandas Series from a Dictionary

```
dct =
{'name':'Asabeneh','country':'Finland','city':'Helsinki'}
```

```
s = pd.Series(dct)
print(s)
```

```
    name       Asabeneh
    country     Finland
    city       Helsinki
    dtype: object
```

Creating a Constant Pandas Series

```
s = pd.Series(10, index = [1, 2, 3])
print(s)
```

```
   1    10
    2    10
    3    10
    dtype: int64
```

Creating a Pandas Series Using Linspace

```
s = pd.Series(np.linspace(5, 20, 10)) # linspace(starting,
end, items)
print(s)
```

```
    0     5.000000
    1     6.666667
    2     8.333333
    3    10.000000
    4    11.666667
    5    13.333333
    6    15.000000
    7    16.666667
    8    18.333333
    9    20.000000
    dtype: float64
```

DataFrames

Pandas data frames can be created in different ways.

Creating DataFrames from List of Lists

```
data = [
    ['Asabeneh', 'Finland', 'Helsink'],
    ['David', 'UK', 'London'],
    ['John', 'Sweden', 'Stockholm']
```

```
]
df = pd.DataFrame(data, columns=['Names','Country','City'])
print(df)
```

	Names	Country	City
0	Asabeneh	Finland	Helsink
1	David	UK	London
2	John	Sweden	Stockholm

Creating DataFrame Using Dictionary

```
data = {'Name': ['Asabeneh', 'David', 'John'], 'Country':[
    'Finland', 'UK', 'Sweden'], 'City': ['Helsiki', 'London',
'Stockholm']}
df = pd.DataFrame(data)
print(df)
```

	Name	Country	City
0	Asabeneh	Finland	Helsiki
1	David	UK	London
2	John	Sweden	Stockholm

Creating DataFrames from a List of Dictionaries

```
data = [
    {'Name': 'Asabeneh', 'Country': 'Finland', 'City':
'Helsinki'},
    {'Name': 'David', 'Country': 'UK', 'City': 'London'},
    {'Name': 'John', 'Country': 'Sweden', 'City':
'Stockholm'}]
df = pd.DataFrame(data)
print(df)
```

	Name	Country	City
0	Asabeneh	Finland	Helsinki
1	David	UK	London
2	John	Sweden	Stockholm

Reading CSV File Using Pandas

To download the CSV file, what is needed in this example, console/command line is enough:

```
curl -O https://raw.githubusercontent.com/Asabeneh/30-Days-Of-Python/master/data/weight-height.csv
```

Put the downloaded file in your working directory.

```
import pandas as pd

df = pd.read_csv('weight-height.csv')
print(df)
```

Data Exploration

Let us read only the first 5 rows using head()

```
print(df.head()) # give five rows we can increase the number of rows by passing argument to the head() method
```

	Gender	Height	Weight
0	Male	73.847017	241.893563
1	Male	68.781904	162.310473
2	Male	74.110105	212.740856
3	Male	71.730978	220.042470
4	Male	69.881796	206.349801

Let us also explore the last recordings of the dataframe using the tail() methods.

```
print(df.tail()) # tails give the last five rows, we can
increase the rows by passing argument to tail method
```

	Gender	Height	Weight
9995	Female	66.172652	136.777454
9996	Female	67.067155	170.867906
9997	Female	63.867992	128.475319
9998	Female	69.034243	163.852461
9999	Female	61.944246	113.649103

As you can see the csv file has three rows: Gender, Height and Weight. If the DataFrame would have a long rows, it would be hard to know all the columns. Therefore, we should use a method to know the colums. we do not know the number of rows. Let's use shape meathod.

```
print(df.shape) # as you can see 10000 rows and three columns
```

```
(10000, 3)
```

Let us get all the columns using columns.

```
print(df.columns)
```

```
Index(['Gender', 'Height', 'Weight'], dtype='object')
```

Now, let us get a specific column using the column key

```
heights = df['Height'] # this is now a series
```

```
print(heights)
```

```
    0       73.847017
    1       68.781904
    2       74.110105
    3       71.730978
    4       69.881796
              ...
    9995    66.172652
    9996    67.067155
    9997    63.867992
    9998    69.034243
    9999    61.944246
    Name: Height, Length: 10000, dtype: float64
```

```
weights = df['Weight'] # this is now a series
```

```
print(weights)
```

```
    0       241.893563
    1       162.310473
    2       212.740856
    3       220.042470
    4       206.349801
               ...
    9995    136.777454
    9996    170.867906
    9997    128.475319
    9998    163.852461
    9999    113.649103
    Name: Weight, Length: 10000, dtype: float64
```

```
print(len(heights) == len(weights))
```

```
True
```

The describe() method provides a descriptive statistical values of a dataset.

```
print(heights.describe()) # give statisical information about
height data
```

```
    count    10000.000000
    mean        66.367560
    std          3.847528
    min         54.263133
    25%         63.505620
    50%         66.318070
    75%         69.174262
    max         78.998742
    Name: Height, dtype: float64
```

```
    print(weights.describe())
```

```
    count    10000.000000
    mean       161.440357
    std         32.108439
    min         64.700127
    25%        135.818051
    50%        161.212928
    75%        187.169525
    max        269.989699
    Name: Weight, dtype: float64
```

```
print(df.describe())  # describe can also give statistical
information from a dataFrame
```

	Height	Weight
count	10000.000000	10000.000000
mean	66.367560	161.440357
std	3.847528	32.108439
min	54.263133	64.700127
25%	63.505620	135.818051
50%	66.318070	161.212928
75%	69.174262	187.169525
max	78.998742	269.989699

Similar to describe(), the info() method also give information about the dataset.

Modifying a DataFrame

Modifying a DataFrame: * We can create a new DataFrame * We can create a new column and add it to the DataFrame, * we can remove an existing column from a DataFrame, * we can modify an existing column in a DataFrame, * we can change the data type of column values in the DataFrame

Creating a DataFrame

As always, first we import the necessary packages. Now, lets import pandas and numpy, two best friends ever.

```
import pandas as pd
import numpy as np
data = [
    {"Name": "Asabeneh",
"Country":"Finland","City":"Helsinki"},
    {"Name": "David", "Country":"UK","City":"London"},
    {"Name": "John", "Country":"Sweden","City":"Stockholm"}]
df = pd.DataFrame(data)
print(df)
```

	Name	Country	City
0	Asabeneh	Finland	Helsinki
1	David	UK	London
2	John	Sweden	Stockholm

Adding a column to a DataFrame is like adding a key to a dictionary.

First let's use the previous example to create a DataFrame. After we create the DataFrame, we will start modifying the columns and column values.

Adding a New Column

Let's add a weight column in the DataFrame

```
weights = [74, 78, 69]
df['Weight'] = weights
df
```

	Name	Country	City	Weight
0	Asabeneh	Finland	Helsinki	74
1	David	UK	London	78
2	John	Sweden	Stockholm	69

Let's add a height column into the DataFrame aswell

```
heights = [173, 175, 169]
df['Height'] = heights
print(df)
```

	Name	Country	City	Weight	Height
0	Asabeneh	Finland	Helsinki	74	173
1	David	UK	London	78	175
2	John	Sweden	Stockholm	69	169

As you can see in the DataFrame above, we did add new columns, Weight and Height. Let's add one additional column called BMI(Body Mass Index) by calculating their BMI using thier mass and height. BMI is mass divided by height squared (in meters) - Weight/Height * Height.

As you can see, the height is in centimeters, so we shoud change it to meters. Let's modify the height row.

Modifying column values

```
df['Height'] = df['Height'] * 0.01
```

```
df
```

	Name	Country	City	Weight	Height
0	Asabeneh	Finland	Helsinki	74	1.73
1	David	UK	London	78	1.75
2	John	Sweden	Stockholm	69	1.69

```
# Using functions makes our code clean, but you can calculate
the bmi without one
def calculate_bmi ():
    weights = df['Weight']
    heights = df['Height']
    bmi = []
    for w,h in zip(weights, heights):
        b = w/(h*h)
        bmi.append(b)
    return bmi

bmi = calculate_bmi()
```

```
df['BMI'] = bmi
df
```

	Name	Country	City	Weight	Height	BMI
0	Asabeneh	Finland	Helsinki	74	1.73	24.725183
1	David	UK	London	78	1.75	25.469388
2	John	Sweden	Stockholm	69	1.69	24.158818

Formating DataFrame columns

The BMI column values of the DataFrame are float with many significant digits after decimal. Let's change it to one significant digit after point.

```
df['BMI'] = round(df['BMI'], 1)
print(df)
```

	Name	Country	City	Weight	Height	BMI
0	Asabeneh	Finland	Helsinki	74	1.73	24.7
1	David	UK	London	78	1.75	25.5
2	John	Sweden	Stockholm	69	1.69	24.2

The information in the DataFrame seems not yet complete, let's add birth year and current year columns.

```
birth_year = ['1769', '1985', '1990']
current_year = pd.Series(2020, index=[0, 1,2])
df['Birth Year'] = birth_year
df['Current Year'] = current_year
df
```

	Name	Country	City	Weight	Height	BMI	Birth Year	Current Year
0	Asabeneh	Finland	Helsinki	74	1.73	24.7	1769	2020
1	David	UK	London	78	1.75	25.5	1985	2020
2	John	Sweden	Stockholm	69	1.69	24.2	1990	2020

Checking data types of Column values

```
print(df.Weight.dtype)
```

```
    dtype('int64')
```

```
df['Birth Year'].dtype # it gives string object , we should
change this to number
```

```
df['Birth Year'] = df['Birth Year'].astype('int')
print(df['Birth Year'].dtype) # let's check the data type now
```

```
    dtype('int32')
```

Now same for the current year:

```
df['Current Year'] = df['Current Year'].astype('int')
df['Current Year'].dtype
```

```
    dtype('int32')
```

Now, the column values of birth year and current year are integers. We can calculate the age.

```
ages = df['Current Year'] - df['Birth Year']
ages
```

```
0    251
1     35
2     30
dtype: int32
```

```
df['Ages'] = ages
print(df)
```

	Name	Country	City	Weight	Height	BMI	Birth Year	Current Year	Ages
0	Asabeneh	Finland	Helsinki	74	1.73	24.7	1769	2019	250
1	David	UK	London	78	1.75	25.5	1985	2019	34
2	John	Sweden	Stockholm	69	1.69	24.2	1990	2019	29

The person in the first row lived so far for 251 years. It is unlikely for someone to live so long. Either it is a typo or the data is cooked. So lets fill that data with average of the columns without including outlier.

mean = (35 + 30)/ 2

```
mean = (35 + 30)/ 2
print('Mean: ',mean)      #it is good to add some description
to the output, so we know what is what
```

```
   Mean:  32.5
```

Boolean Indexing

```
print(df[df['Ages'] > 120])
```

	Name	Country	City	Weight	Height	BMI	Birth Year	Current Year	Ages
0	Asabeneh	Finland	Helsinki	74	1.73	24.7	1769	2020	251

```
print(df[df['Ages'] < 120])
```

	Name	Country	City	Weight	Height	BMI	Birth Year	Current Year	Ages
1	David	UK	London	78	1.75	25.5	1985	2020	35
2	John	Sweden	Stockholm	69	1.69	24.2	1990	2020	30

Exercises: Day 25

1. Read the hacker_news.csv file from data directory
2. Get the first five rows
3. Get the last five rows
4. Get the title column as pandas series
5. Count the number of rows and columns
    - Filter the titles which contain python
    - Filter the titles which contain JavaScript
    - Explore the data and make sense of it

🎉 CONGRATULATIONS ! 🎉

DAY-26 PYTHON FOR WEB

Python for Web

Python is a general purpose programming language and it can be used for many places. In this section, we will see how we use Python for the web. There are many Python web frame works. Django and Flask are the most popular ones. Today, we will see how to use Flask for web development.

Flask

Flask is a web development framework written in Python. Flask uses Jinja2 template engine. Flask can be also used with other modern front libraries such as React.

If you did not install the virtualenv package yet install it first. Virtual environment will allows to isolate project dependencies from the local machine dependencies.

Folder structure

After completing all the step, your project file structure should look like this:

```
├── Procfile
├── app.py
├── env
│   ├── bin
├── requirements.txt
├── static
│   └── css
│       └── main.css
└── templates
    ├── about.html
    ├── home.html
    ├── layout.html
    ├── post.html
    └── result.html
```

Setting up your project directory

Follow the following steps to get started with Flask.

Step 1: install virtualenv using the following command.

```
pip install virtualenv
```

Step 2:

```
asabeneh@Asabeneh:~/Desktop$ mkdir python_for_web
asabeneh@Asabeneh:~/Desktop$ cd python_for_web/
asabeneh@Asabeneh:~/Desktop/python_for_web$ virtualenv venv
asabeneh@Asabeneh:~/Desktop/python_for_web$ source
venv/bin/activate
(env) asabeneh@Asabeneh:~/Desktop/python_for_web$ pip freeze
(env) asabeneh@Asabeneh:~/Desktop/python_for_web$ pip install
Flask
(env) asabeneh@Asabeneh:~/Desktop/python_for_web$ pip freeze
Click==7.0
Flask==1.1.1
itsdangerous==1.1.0
Jinja2==2.10.3
MarkupSafe==1.1.1
Werkzeug==0.16.0
(env) asabeneh@Asabeneh:~/Desktop/python_for_web$
```

We created a project director named python_for_web. Inside the project we created a virtual environment *venv* which could be any name but I prefer to call it *venv*. Then we activated the virtual environment. We used pip freeze to check the installed packages in the project directory. The result of pip freeze was empty because a package was not installed yet.

Now, let's create app.py file in the project directory and write the following code. The app.py file will be the main file in the project. The following code has flask module, os module.

Creating routes

The home route.

```
# let's import the flask
from flask import Flask
import os # importing operating system module

app = Flask(__name__)

@app.route('/') # this decorator create the home route
def home ():
    return '<h1>Welcome</h1>'

@app.route('/about')
def about():
    return '<h1>About us</h1>'
```

```
if __name__ == '__main__':
    # for deployment we use the environ
    # to make it work for both production and development
    port = int(os.environ.get("PORT", 5000))
    app.run(debug=True, host='0.0.0.0', port=port)
```

To run the flask application, write python app.py in the main flask application directory.

After you run *python app.py* check local host 5000.

Let us add additional route. Creating about route

```
# let's import the flask
from flask import Flask
import os # importing operating system module

app = Flask(__name__)

@app.route('/') # this decorator create the home route
def home ():
    return '<h1>Welcome</h1>'

@app.route('/about')
def about():
    return '<h1>About us</h1>'

if __name__ == '__main__':
    # for deployment we use the environ
    # to make it work for both production and development
    port = int(os.environ.get("PORT", 5000))
    app.run(debug=True, host='0.0.0.0', port=port)
```

Now, we added the about route in the above code. How about if we want to render an HTML file instead of string? It is possible to render HTML file using the function *render_templae*. Let us create a folder called templates and create home.html and about.html in the project directory. Let us also import the *render_template* function from flask.

Creating templates

Create the HTML files inside templates folder.

home.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width,
initial-scale=1.0" />
    <title>Home</title>
  </head>

  <body>
    <h1>Welcome Home</h1>
  </body>
</html>
```

about.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width,
initial-scale=1.0" />
    <title>About</title>
  </head>

  <body>
    <h1>About Us</h1>
  </body>
</html>
```

Python Script

app.py

```
# let's import the flask
from flask import Flask, render_template
import os # importing operating system module

app = Flask(__name__)

@app.route('/') # this decorator create the home route
def home ():
    return render_template('home.html')

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

if __name__ == '__main__':
```

```
    # for deployment we use the environ
    # to make it work for both production and development
    port = int(os.environ.get("PORT", 5000))
    app.run(debug=True, host='0.0.0.0', port=port)
```

As you can see to go to different pages or to navigate we need a navigation. Let's add a link to each page or let's create a layout which we use to every page.

Navigation

```
<ul>
  <li><a href="/">Home</a></li>
  <li><a href="/about">About</a></li>
</ul>
```

Now, we can navigate between the pages using the above link. Let us create additional page which handle form data. You can call it any name, I like to call it post.html.

We can inject data to the HTML files using Jinja2 template engine.

```
# let's import the flask
from flask import Flask, render_template, request, redirect,
url_for
import os # importing operating system module

app = Flask(__name__)

@app.route('/') # this decorator create the home route
def home ():
    techs = ['HTML', 'CSS', 'Flask', 'Python']
    name = '30 Days Of Python Programming'
    return render_template('home.html', techs=techs, name =
name, title = 'Home')

@app.route('/about')
def about():
    name = '30 Days Of Python Programming'
    return render_template('about.html', name = name, title =
'About Us')

@app.route('/post')
def post():
    name = 'Text Analyzer'
    return render_template('post.html', name = name, title =
name)
```

```
if __name__ == '__main__':
    # for deployment
    # to make it work for both production and development
    port = int(os.environ.get("PORT", 5000))
    app.run(debug=True, host='0.0.0.0', port=port)
```

Let's see the templates too:

home.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width,
initial-scale=1.0" />
    <title>Home</title>
  </head>

  <body>
    <ul>
      <li><a href="/">Home</a></li>
      <li><a href="/about">About</a></li>
    </ul>
    <h1>Welcome to {{name}}</h1>
     <ul>
    {% for tech in techs %}
      <li>{{tech}}</li>
    {% endfor %}
    </ul>
  </body>
</html>
```

about.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width,
initial-scale=1.0" />
    <title>About Us</title>
  </head>

  <body>
    <ul>
      <li><a href="/">Home</a></li>
      <li><a href="/about">About</a></li>
    </ul>
    <h1>About Us</h1>
```

```
    <h2>{{name}}</h2>
  </body>
</html>
```

Creating a layout

In the template files, there are lots of repeated codes, we can write a layout and we can remove the repetition. Let's create layout.html inside the templates folder. After we create the layout we will import to every file.

Serving Static File

Create a static folder in your project directory. Inside the static folder create CSS or styles folder and create a CSS stylesheet. We use the *url_for* module to serve the static file.

layout.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width,
initial-scale=1.0" />
    <link

href="https://fonts.googleapis.com/css?family=Lato:300,400|Nun
ito:300,400|Raleway:300,400,500&display=swap"
      rel="stylesheet"
    />
    <link
      rel="stylesheet"
      href="{{ url_for('static', filename='css/main.css') }}"
    />
    {% if title %}
    <title>30 Days of Python - {{ title}}</title>
    {% else %}
    <title>30 Days of Python</title>
    {% endif %}
  </head>

  <body>
    <header>
      <div class="menu-container">
        <div>
          <a class="brand-name nav-link"
href="/">30DaysOfPython</a>
        </div>
        <ul class="nav-lists">
          <li class="nav-list">
```

```
            <a class="nav-link active" href="{{
url_for('home') }}">Home</a>
          </li>
          <li class="nav-list">
            <a class="nav-link active" href="{{
url_for('about') }}">About</a>
          </li>
          <li class="nav-list">
            <a class="nav-link active" href="{{
url_for('post') }}"
              >Text Analyzer</a
            >
          </li>
        </ul>
      </div>
    </header>
    <main>
      {% block content %} {% endblock %}
    </main>
  </body>
</html>
```

Now, lets remove all the repeated code in the other template files and import the layout.html. The href is using *url_for* function with the name of the route function to connect each navigation route.

home.html

```
{% extends 'layout.html' %} {% block content %}
<div class="container">
  <h1>Welcome to {{name}}</h1>
  <p>
    This application clean texts and analyse the number of
word, characters and
    most frequent words in the text. Check it out by click
text analyzer at the
    menu. You need the following technologies to build this
web application:
  </p>
  <ul class="tech-lists">
    {% for tech in techs %}
    <li class="tech">{{tech}}</li>

    {% endfor %}
  </ul>
</div>

{% endblock %}
```

about.html

```
{% extends 'layout.html' %} {% block content %}
<div class="container">
  <h1>About {{name}}</h1>
  <p>
    This is a 30 days of python programming challenge. If you
have been coding
    this far, you are awesome. Congratulations for the job
well done!
  </p>
</div>
{% endblock %}
```

post.html

```
{% extends 'layout.html' %} {% block content %}
<div class="container">
  <h1>Text Analyzer</h1>
  <form action="https://thirtydaysofpython-
v1.herokuapp.com/post" method="POST">
    <div>
      <textarea rows="25" name="content" autofocus></textarea>
    </div>
    <div>
      <input type="submit" class="btn" value="Process Text" />
    </div>
  </form>
</div>

{% endblock %}
```

Request methods, there are different request methods(GET, POST, PUT, DELETE) are the common request methods which allow us to do CRUD(Create, Read, Update, Delete) operation.

In the post, route we will use GET and POST method alternative depending on the type of request, check how it looks in the code below. The request method is a function to handle request methods and also to access form data. app.py

```
# let's import the flask
from flask import Flask, render_template, request, redirect,
url_for
import os # importing operating system module

app = Flask(__name__)
# to stop caching static file
app.config['SEND_FILE_MAX_AGE_DEFAULT'] = 0
```

```
@app.route('/') # this decorator create the home route
def home ():
    techs = ['HTML', 'CSS', 'Flask', 'Python']
    name = '30 Days Of Python Programming'
    return render_template('home.html', techs=techs, name =
name, title = 'Home')

@app.route('/about')
def about():
    name = '30 Days Of Python Programming'
    return render_template('about.html', name = name, title =
'About Us')

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

@app.route('/post', methods= ['GET','POST'])
def post():
    name = 'Text Analyzer'
    if request.method == 'GET':
         return render_template('post.html', name = name,
title = name)
    if request.method =='POST':
        content = request.form['content']
        print(content)
        return redirect(url_for('result'))

if __name__ == '__main__':
    # for deployment
    # to make it work for both production and development
    port = int(os.environ.get("PORT", 5000))
    app.run(debug=True, host='0.0.0.0', port=port)
```

So far, we have seen how to use template and how to inject data to template, how to a common layout. Now, lets handle static file. Create a folder called static in the project director and create a folder called css. Inside css folder create main.css. Your main. css file will be linked to the layout.html.

You don't have to write the css file, copy and use it. Let's move on to deployment.

Deployment

Creating Heroku account

Heroku provides a free deployment service for both front end and fullstack applications. Create an account on heroku and install the heroku CLI for you machine. After installing heroku write the following command

Login to Heroku

```
asabeneh@Asabeneh:~$ heroku login
heroku: Press any key to open up the browser to login or q to
exit:
```

Let's see the result by clicking any key from the keyboard. When you press any key from you keyboard it will open the heroku login page and click the login page. Then you will local machine will be connected to the remote heroku server. If you are connected to remote server, you will see this.

```
asabeneh@Asabeneh:~$ heroku login
heroku: Press any key to open up the browser to login or q to
exit:
Opening browser to https://cli-
auth.heroku.com/auth/browser/be12987c-583a-4458-a2c2-
ba2ce7f41610
Logging in... done
Logged in as asabeneh@gmail.com
asabeneh@Asabeneh:~$
```

Create requirements and Procfile

Before we push our code to remote server, we need requirements

- requirements.txt
- Procfile

```
(env) asabeneh@Asabeneh:~/Desktop/python_for_web$ pip freeze
Click==7.0
Flask==1.1.1
itsdangerous==1.1.0
Jinja2==2.10.3
MarkupSafe==1.1.1
Werkzeug==0.16.0
(env) asabeneh@Asabeneh:~/Desktop/python_for_web$ touch
requirements.txt
(env) asabeneh@Asabeneh:~/Desktop/python_for_web$ pip freeze >
requirements.txt
(env) asabeneh@Asabeneh:~/Desktop/python_for_web$ cat
requirements.txt
Click==7.0
Flask==1.1.1
itsdangerous==1.1.0
Jinja2==2.10.3
MarkupSafe==1.1.1
Werkzeug==0.16.0
```

```
(env) asabeneh@Asabeneh:~/Desktop/python_for_web$ touch
Procfile
(env) asabeneh@Asabeneh:~/Desktop/python_for_web$ ls
Procfile          env/              static/
app.py            requirements.txt  templates/
(env) asabeneh@Asabeneh:~/Desktop/python_for_web$
```

The Procfile will have the command which run the application in the web server in our case on Heroku.

```
web: python app.py
```

Pushing project to heroku

Now, it is ready to be deployed. Steps to deploy the application on heroku

1. git init
2. git add .
3. git commit -m "commit message"
4. heroku create 'name of the app as one word'
5. git push heroku master
6. heroku open(to launch the deployed application)

After this step you will get an application like this

Exercises: Day 26

1. You will build this application. Only the text analyser part is left

🎉 CONGRATULATIONS ! 🎉

DAY-27 PYTHON WITH MongoDB

Python is a backend technology and it can be connected with different data base applications. It can be connected to both SQL and noSQL databases. In this section, we connect Python with MongoDB database which is noSQL database.

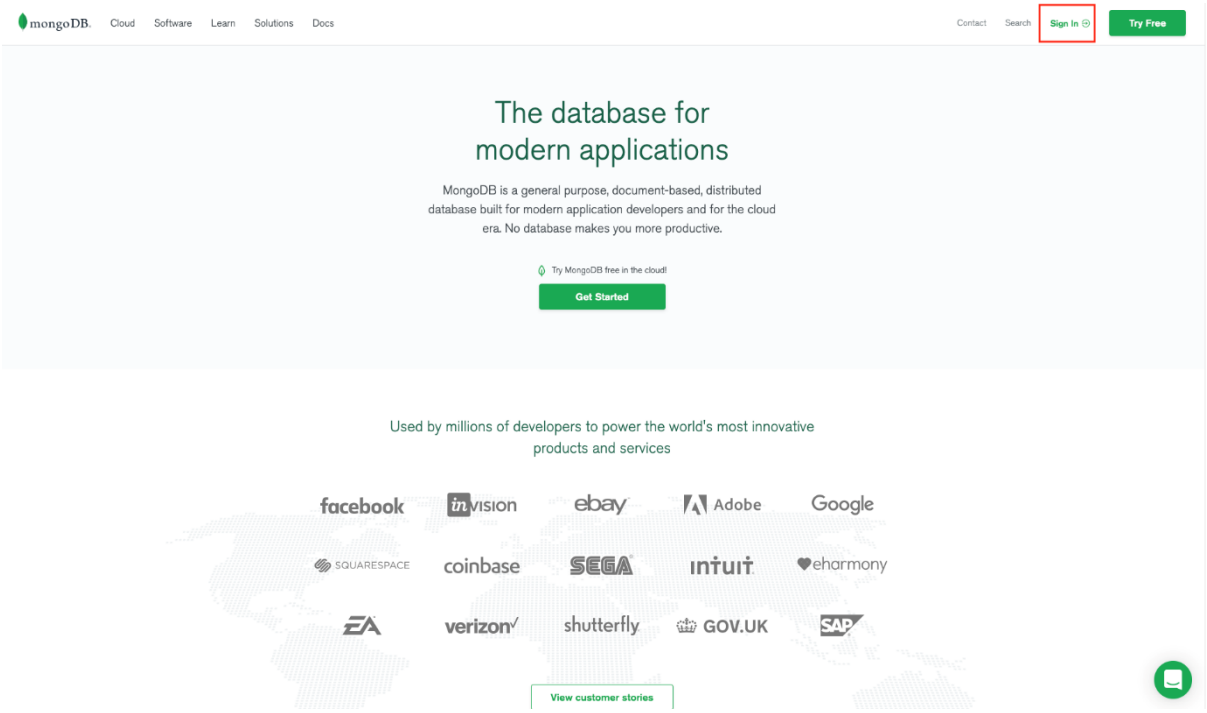
MongoDB

MongoDB is a NoSQL database. MongoDB stores data in a JSON like document which make MongoDB very flexible and scalable. Let us see the different terminologies of SQL and NoSQL databases. The following table will make the difference between SQL versus NoSQL databases.

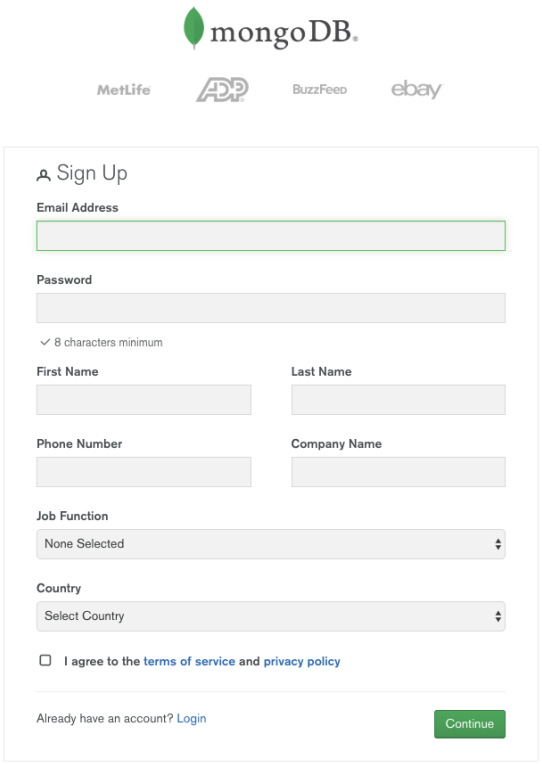
SQL versus NoSQL

SQL	NoSQL
Database	Database
Tables	Collections
Rows	Documents
Columns	Fields
Index	Index
Join	Embedding and Linking
Group by	Aggregation
Primary Key	_id field

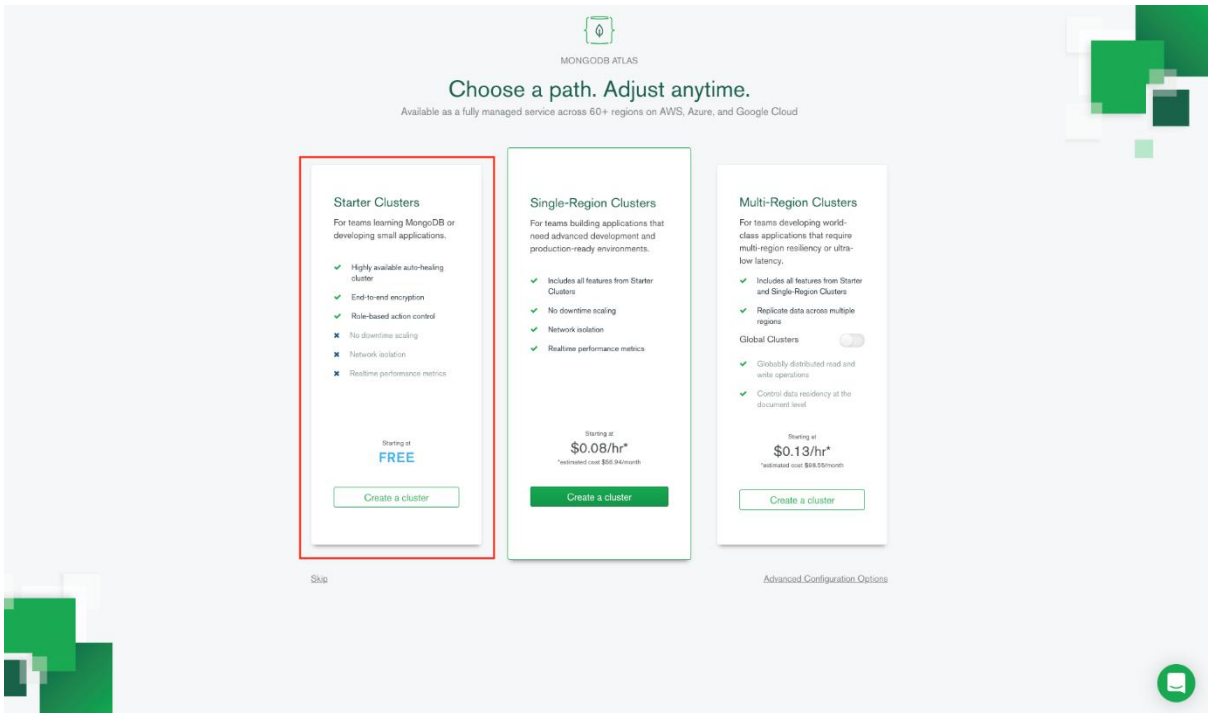
In this section, we will focus on a NoSQL database MongoDB. Lets sign up on mongoDB by click on the sign in button then click register on the next page.



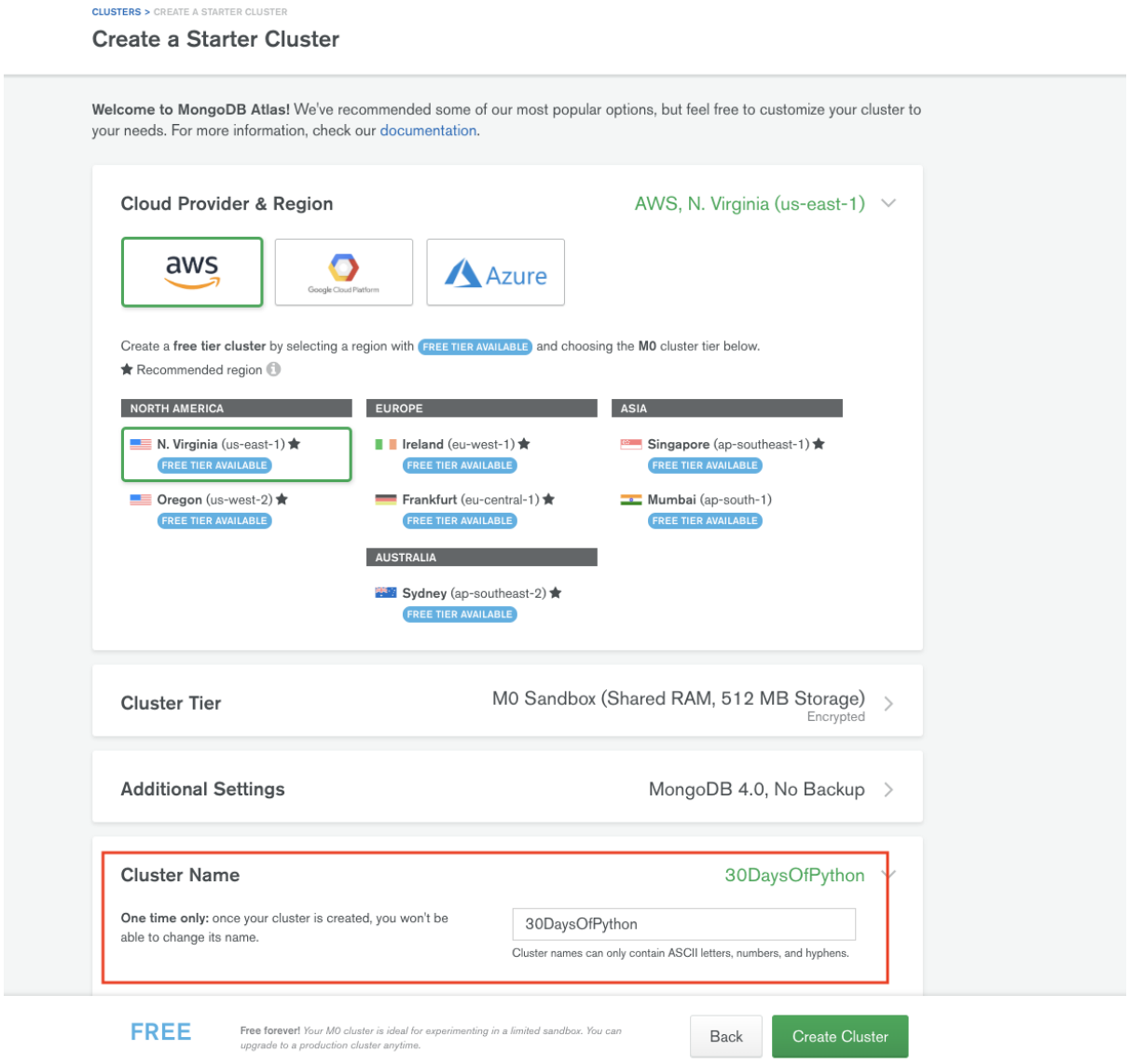
Complete the fields and click continue



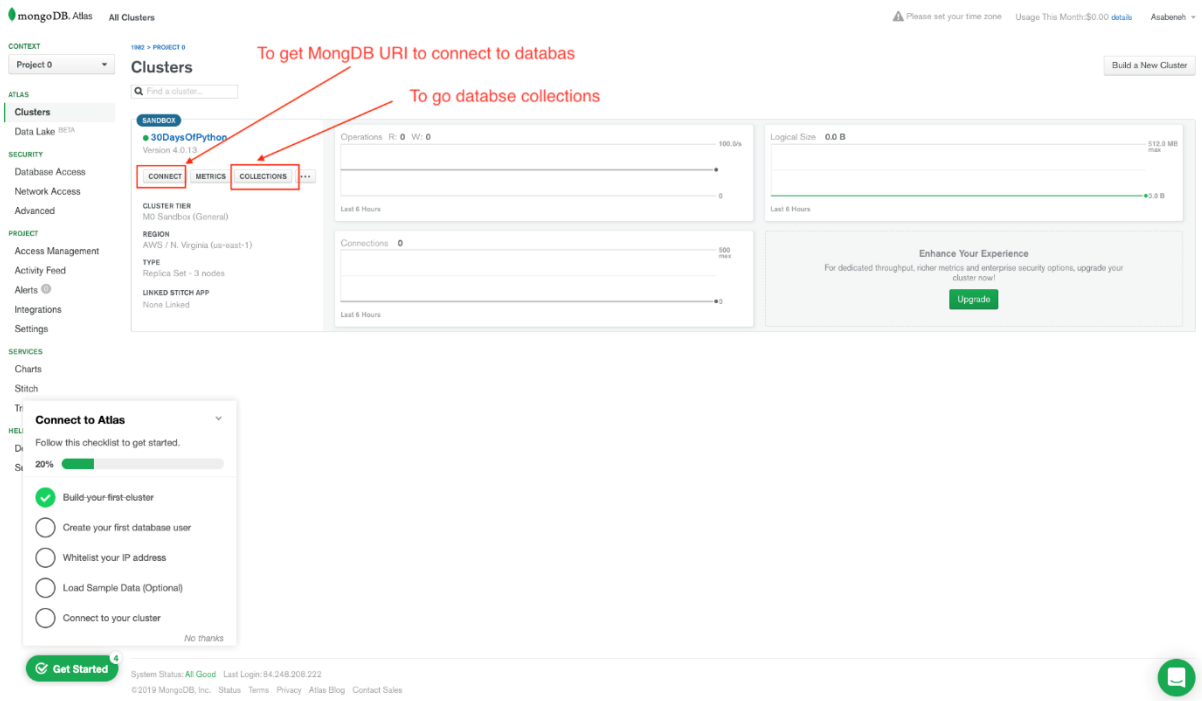
Select the free plan



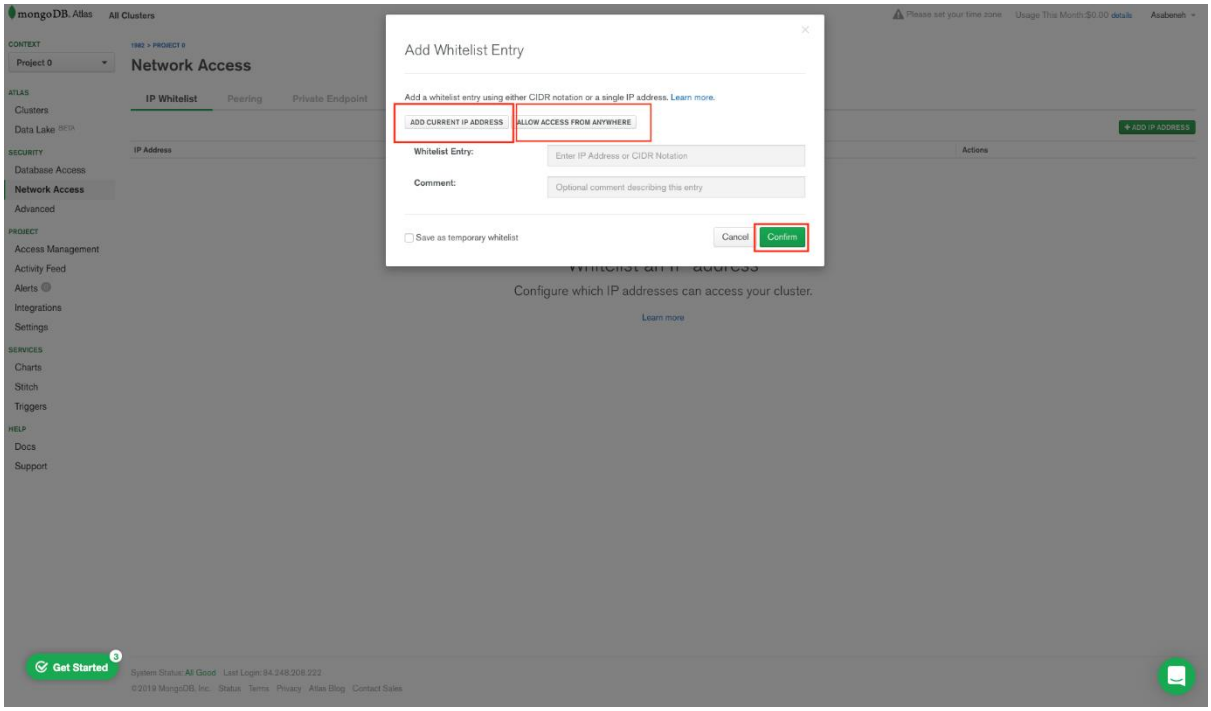
Choose the proximate free region and give any name for you cluster.



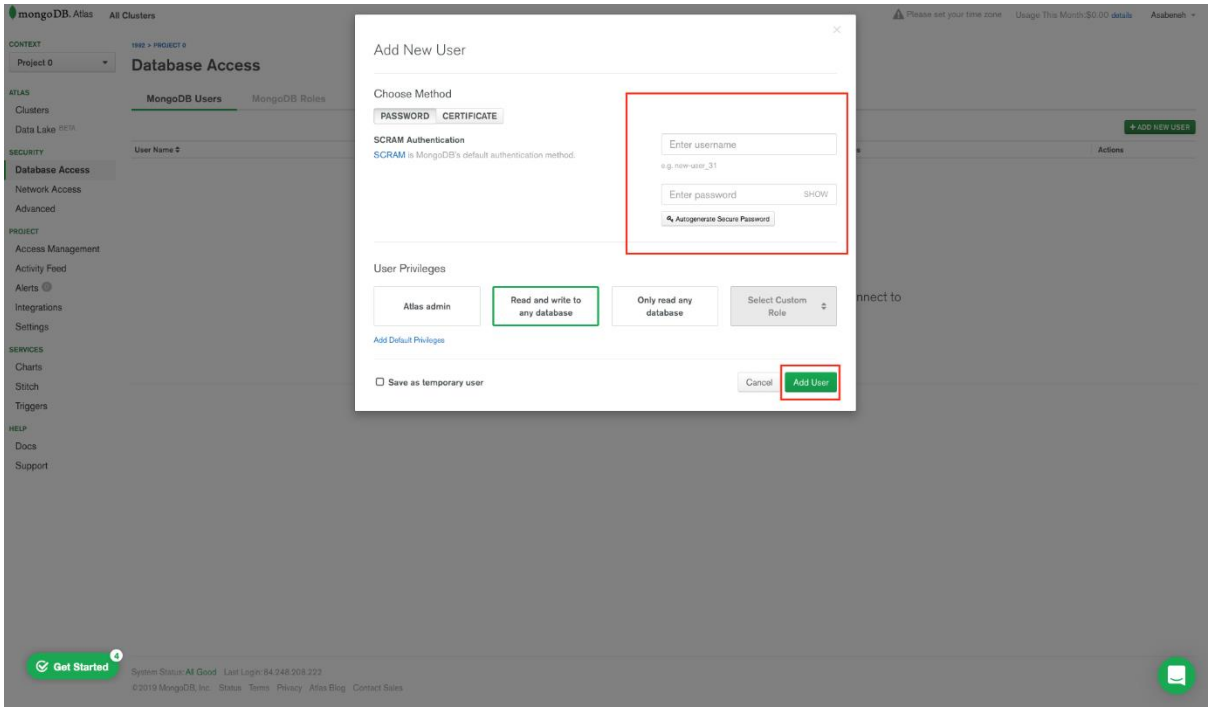
Now, a free sandbox is created



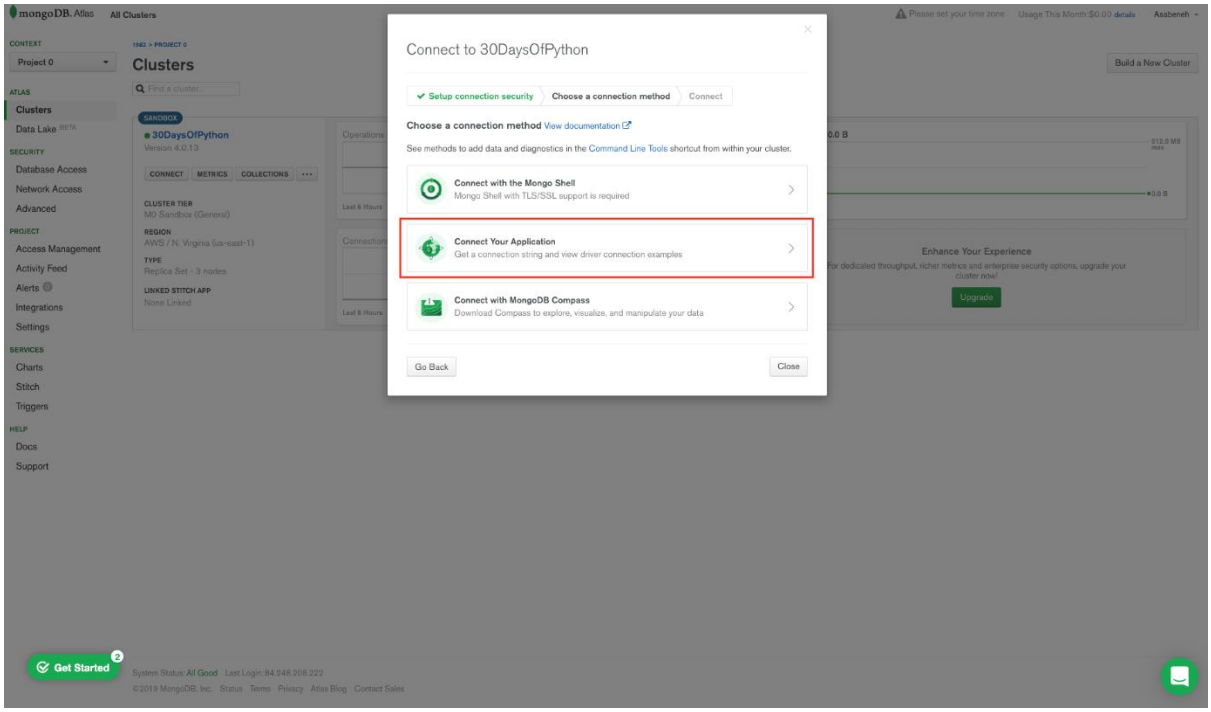
All local host access



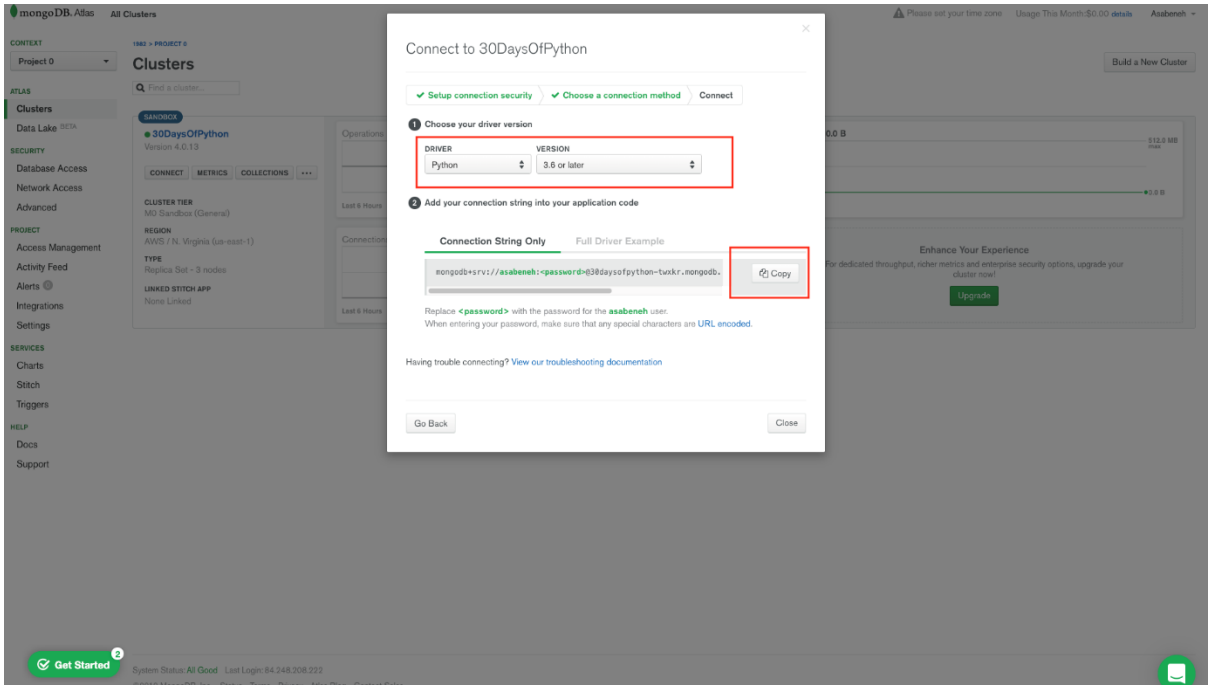
Add user and password



Create a mongoDB uri link



Select Python 3.6 or above driver



Getting Connection String(MongoDB URI)

Copy the connection string link and you will get something like this:

```
mongodb+srv://asabeneh:<password>@30daysofpython-
twxkr.mongodb.net/test?retryWrites=true&w=majority
```

Do not worry about the url, it is a means to connect your application with mongoDB. Let us replace the password placeholder with the password you used to add a user.

Example:

```
mongodb+srv://asabeneh:123123123@30daysofpython-
twxkr.mongodb.net/test?retryWrites=true&w=majority
```

Now, I replaced everything and the password is 123123 and the name of the database is *thirty_days_python*. This is just an example, your password must be stronger than the example password.

Python needs a mongoDB driver to access mongoDB database. We will use *pymongo* with *dnspython* to connect our application with mongoDB base . Inside your project directory install pymongo and dnspython.

```
pip install pymongo dnspython
```

The "dnspython" module must be installed to use mongodb+srv:// URIs. The dnspython is a DNS toolkit for Python. It supports almost all record types.

Connecting Flask application to MongoDB Cluster

```
# let's import the flask
from flask import Flask, render_template
import os # importing operating system module
MONGODB_URI =
'mongodb+srv://asabeneh:your_password_goes_here@30daysofpython
-twxkr.mongodb.net/test?retryWrites=true&w=majority'
client = pymongo.MongoClient(MONGODB_URI)
print(client.list_database_names())

app = Flask(__name__)
if __name__ == '__main__':
    # for deployment we use the environ
    # to make it work for both production and development
    port = int(os.environ.get("PORT", 5000))
    app.run(debug=True, host='0.0.0.0', port=port)
```

When we run the above code we get the default mongoDB databases.

```
['admin', 'local']
```

Creating a database and collection

Let us create a database, database and collection in mongoDB will be created if it doesn't exist. Let's create a data base
name *thirty_days_of_python* and *students* collection.

To create a database:

```
db = client.name_of_databse # we can create a database like
this or the second way
db = client['name_of_database']
```

```
# let's import the flask
from flask import Flask, render_template
import os # importing operating system module
MONGODB_URI =
'mongodb+srv://asabeneh:your_password_goes_here@30daysofpython
-twxkr.mongodb.net/test?retryWrites=true&w=majority'
client = pymongo.MongoClient(MONGODB_URI)
# Creating database
db = client.thirty_days_of_python
# Creating students collection and inserting a document
db.students.insert_one({'name': 'Asabeneh', 'country':
'Finland', 'city': 'Helsinki', 'age': 250})
print(client.list_database_names())

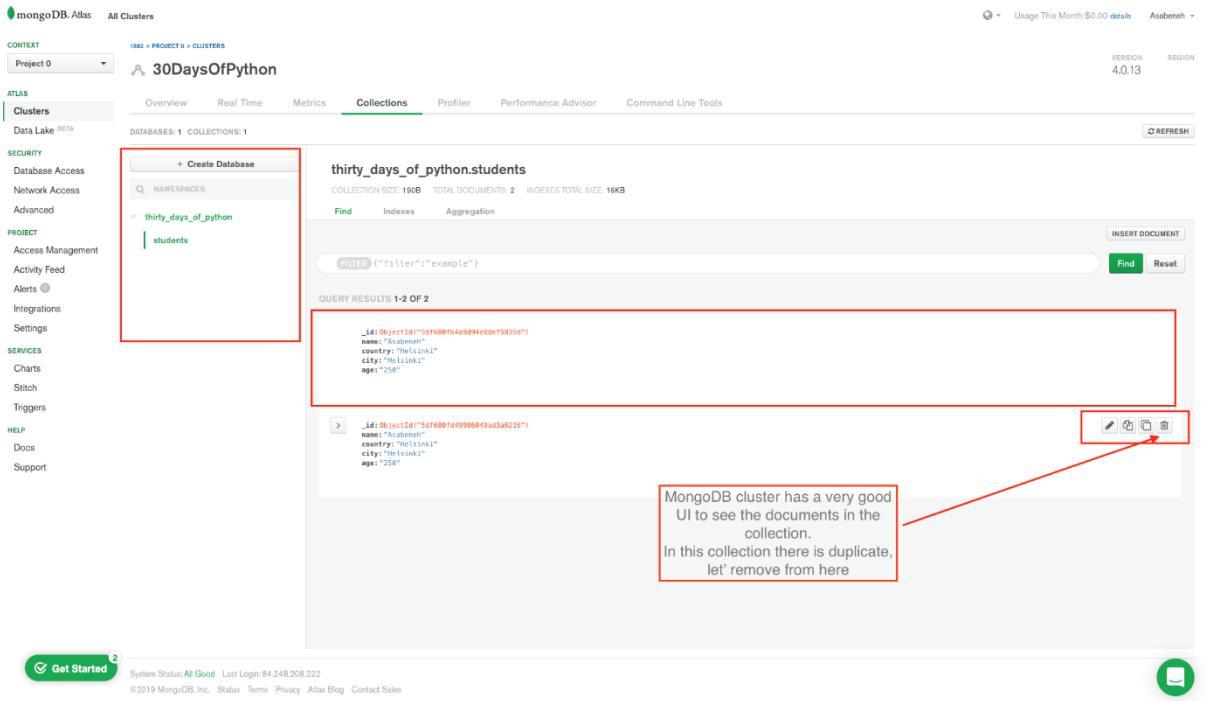
app = Flask(__name__)
if __name__ == '__main__':
    # for deployment we use the environ
```

```
    # to make it work for both production and development
    port = int(os.environ.get("PORT", 5000))
    app.run(debug=True, host='0.0.0.0', port=port)
```

After we create a database, we also created a students collection and we used *insert_one()* method to insert a document. Now, the database *thirty_days_of_python* and *students* collection have been created and the document has been inserted. Check your mongoDB cluster and you will see both the database and the collection. Inside the collection, there will be a document.

```
['thirty_days_of_python', 'admin', 'local']
```

If you see this on the mongoDB cluster, it means you have successfully created a database and a collection.



If you have seen on the figure, the document has been created with a long id which acts as a primary key. Every time we create a document mongoDB create and unique id for it.

Inserting many documents to collection

The *insert_one()* method inserts one item at a time if we want to insert many documents at once either we use *insert_many()* method or for loop. We can use for loop to inset many documents at once.

```
# let's import the flask
```

```
from flask import Flask, render_template
import os # importing operating system module
MONGODB_URI =
'mongodb+srv://asabeneh:your_password_goes_here@30daysofpython
-twxkr.mongodb.net/test?retryWrites=true&w=majority'
client = pymongo.MongoClient(MONGODB_URI)

students = [

{'name':'David','country':'UK','city':'London','age':34},

{'name':'John','country':'Sweden','city':'Stockholm','age':28}
,

{'name':'Sami','country':'Finland','city':'Helsinki','age':25}
,
    ]
for student in students:
    db.students.insert_one(student)


app = Flask(__name__)
if __name__ == '__main__':
    # for deployment we use the environ
    # to make it work for both production and development
    port = int(os.environ.get("PORT", 5000))
    app.run(debug=True, host='0.0.0.0', port=port)
```

MongoDB Find

The *find()* and *findOne()* methods are common method to find data in a collection in mongoDB database. It is similar to the SELECT statement in a MySQL database. Let us use the *find_one()* method to get a document in a database collection.

- *find_one({"_id": ObjectId("id"}): Gets the first occurrence if an id is not provided

```
# let's import the flask
from flask import Flask, render_template
import os # importing operating system module
MONGODB_URI =
'mongodb+srv://asabeneh:your_password_goes_here@30daysofpython
-twxkr.mongodb.net/test?retryWrites=true&w=majority'
client = pymongo.MongoClient(MONGODB_URI)
db = client['thirty_days_of_python'] # accessing the database
student = db.students.find_one()
print(student)


app = Flask(__name__)
```

```
if __name__ == '__main__':
    # for deployment we use the environ
    # to make it work for both production and development
    port = int(os.environ.get("PORT", 5000))
    app.run(debug=True, host='0.0.0.0', port=port)
```

```
{'_id': ObjectId('5df68a21f106fe2d315bbc8b'), 'name':
'Asabeneh', 'country': 'Helsinki', 'city': 'Helsinki', 'age':
250}
```

The above query returns the first entry but we can target specific document using specific _id. Let us do one example, use David's id to get David object. '_id':ObjectId('5df68a23f106fe2d315bbc8c')

```
# let's import the flask
from flask import Flask, render_template
import os # importing operating system module
from bson.objectid import ObjectId # id object
MONGODB_URI =
'mongodb+srv://asabeneh:your_password_goes_here@30daysofpython
-twxkr.mongodb.net/test?retryWrites=true&w=majority'
client = pymongo.MongoClient(MONGODB_URI)
db = client['thirty_days_of_python'] # accessing the database
student =
db.students.find_one({'_id':ObjectId('5df68a23f106fe2d315bbc8c
')})
print(student)

app = Flask(__name__)
if __name__ == '__main__':
    # for deployment we use the environ
    # to make it work for both production and development
    port = int(os.environ.get("PORT", 5000))
    app.run(debug=True, host='0.0.0.0', port=port)
```

```
{'_id': ObjectId('5df68a23f106fe2d315bbc8c'), 'name': 'David',
'country': 'UK', 'city': 'London', 'age': 34}
```

We have seen, how to use *find_one()* using the above examples. Let's move one to *find()*

- *find()*: returns all the occurrence from a collection if we don't pass a query object. The object is pymongo.cursor object.

```
# let's import the flask
from flask import Flask, render_template
import os # importing operating system module

MONGODB_URI =
'mongodb+srv://asabeneh:your_password_goes_here@30daysofpython
-twxkr.mongodb.net/test?retryWrites=true&w=majority'
client = pymongo.MongoClient(MONGODB_URI)
db = client['thirty_days_of_python'] # accessing the database
students = db.students.find()
for student in students:
    print(student)

app = Flask(__name__)
if __name__ == '__main__':
    # for deployment we use the environ
    # to make it work for both production and development
    port = int(os.environ.get("PORT", 5000))
    app.run(debug=True, host='0.0.0.0', port=port)
```

```
{'_id': ObjectId('5df68a21f106fe2d315bbc8b'), 'name':
'Asabeneh', 'country': 'Finland', 'city': 'Helsinki', 'age':
250}
{'_id': ObjectId('5df68a23f106fe2d315bbc8c'), 'name': 'David',
'country': 'UK', 'city': 'London', 'age': 34}
{'_id': ObjectId('5df68a23f106fe2d315bbc8d'), 'name': 'John',
'country': 'Sweden', 'city': 'Stockholm', 'age': 28}
{'_id': ObjectId('5df68a23f106fe2d315bbc8e'), 'name': 'Sami',
'country': 'Finland', 'city': 'Helsinki', 'age': 25}
```

We can specify which fields to return by passing second object in the *find({}, {})*. 0 means not include and 1 means include but we can not mix 0 and 1, except for _id.

```
# let's import the flask
from flask import Flask, render_template
import os # importing operating system module

MONGODB_URI =
'mongodb+srv://asabeneh:your_password_goes_here@30daysofpython
-twxkr.mongodb.net/test?retryWrites=true&w=majority'
client = pymongo.MongoClient(MONGODB_URI)
db = client['thirty_days_of_python'] # accessing the database
students = db.students.find({}, {"_id":0,  "name": 1,
"country":1}) # 0 means not include and 1 means include
for student in students:
    print(student)
```

```
app = Flask(__name__)
if __name__ == '__main__':
    # for deployment we use the environ
    # to make it work for both production and development
    port = int(os.environ.get("PORT", 5000))
    app.run(debug=True, host='0.0.0.0', port=port)
```

```
{'name': 'Asabeneh', 'country': 'Finland'}
{'name': 'David', 'country': 'UK'}
{'name': 'John', 'country': 'Sweden'}
{'name': 'Sami', 'country': 'Finland'}
```

Find with Query

In mongoDB find take a query object. We can pass a query object and we can filter the documents we like to filter out.

```
# let's import the flask
from flask import Flask, render_template
import os # importing operating system module

MONGODB_URI =
'mongodb+srv://asabeneh:your_password_goes_here@30daysofpython
-twxkr.mongodb.net/test?retryWrites=true&w=majority'
client = pymongo.MongoClient(MONGODB_URI)
db = client['thirty_days_of_python'] # accessing the database

query = {
    "country":"Finland"
}
students = db.students.find(query)

for student in students:
    print(student)


app = Flask(__name__)
if __name__ == '__main__':
    # for deployment we use the environ
    # to make it work for both production and development
    port = int(os.environ.get("PORT", 5000))
    app.run(debug=True, host='0.0.0.0', port=port)
```

```
{'_id': ObjectId('5df68a21f106fe2d315bbc8b'), 'name':
'Asabeneh', 'country': 'Finland', 'city': 'Helsinki', 'age':
250}
{'_id': ObjectId('5df68a23f106fe2d315bbc8e'), 'name': 'Sami',
'country': 'Finland', 'city': 'Helsinki', 'age': 25}
```

Query with modifiers

```
# let's import the flask
from flask import Flask, render_template
import os # importing operating system module
import pymongo

MONGODB_URI =
'mongodb+srv://asabeneh:your_password_goes_here@30daysofpython
-twxkr.mongodb.net/test?retryWrites=true&w=majority'
client = pymongo.MongoClient(MONGODB_URI)
db = client['thirty_days_of_python'] # accessing the database

query = {
    "city":"Helsinki"
}
students = db.students.find(query)
for student in students:
    print(student)


app = Flask(__name__)
if __name__ == '__main__':
    # for deployment we use the environ
    # to make it work for both production and development
    port = int(os.environ.get("PORT", 5000))
    app.run(debug=True, host='0.0.0.0', port=port)
```

```
{'_id': ObjectId('5df68a21f106fe2d315bbc8b'), 'name':
'Asabeneh', 'country': 'Finland', 'city': 'Helsinki', 'age':
250}
{'_id': ObjectId('5df68a23f106fe2d315bbc8e'), 'name': 'Sami',
'country': 'Finland', 'city': 'Helsinki', 'age': 25}
```

Find query with modifier

```
# let's import the flask
from flask import Flask, render_template
import os # importing operating system module
import pymongo

MONGODB_URI =
'mongodb+srv://asabeneh:your_password_goes_here@30daysofpython
-twxkr.mongodb.net/test?retryWrites=true&w=majority'
client = pymongo.MongoClient(MONGODB_URI)
db = client['thirty_days_of_python'] # accessing the database
query = {
    "country":"Finland",
    "city":"Helsinki"
}
students = db.students.find(query)
for student in students:
    print(student)


app = Flask(__name__)
if __name__ == '__main__':
    # for deployment we use the environ
    # to make it work for both production and development
    port = int(os.environ.get("PORT", 5000))
    app.run(debug=True, host='0.0.0.0', port=port)
```

```
{'_id': ObjectId('5df68a21f106fe2d315bbc8b'), 'name':
'Asabeneh', 'country': 'Finland', 'city': 'Helsinki', 'age':
250}
{'_id': ObjectId('5df68a23f106fe2d315bbc8e'), 'name': 'Sami',
'country': 'Finland', 'city': 'Helsinki', 'age': 25}
```

Query with modifiers

```
# let's import the flask
from flask import Flask, render_template
import os # importing operating system module
import pymongo

MONGODB_URI =
'mongodb+srv://asabeneh:your_password_goes_here@30daysofpython
-twxkr.mongodb.net/test?retryWrites=true&w=majority'
client = pymongo.MongoClient(MONGODB_URI)
db = client['thirty_days_of_python'] # accessing the database
query = {"age":{"$gt":30}}
students = db.students.find(query)
for student in students:
```

```
    print(student)

app = Flask(__name__)
if __name__ == '__main__':
    # for deployment we use the environ
    # to make it work for both production and development
    port = int(os.environ.get("PORT", 5000))
    app.run(debug=True, host='0.0.0.0', port=port)
```

```
{'_id': ObjectId('5df68a21f106fe2d315bbc8b'), 'name':
'Asabeneh', 'country': 'Finland', 'city': 'Helsinki', 'age':
250}
{'_id': ObjectId('5df68a23f106fe2d315bbc8c'), 'name': 'David',
'country': 'UK', 'city': 'London', 'age': 34}
```

```
# let's import the flask
from flask import Flask, render_template
import os # importing operating system module
import pymongo

MONGODB_URI =
'mongodb+srv://asabeneh:your_password_goes_here@30daysofpython
-twxkr.mongodb.net/test?retryWrites=true&w=majority'
client = pymongo.MongoClient(MONGODB_URI)
db = client['thirty_days_of_python'] # accessing the database
query = {"age":{"$gt":30}}
students = db.students.find(query)
for student in students:
    print(student)
```

```
{'_id': ObjectId('5df68a23f106fe2d315bbc8d'), 'name': 'John',
'country': 'Sweden', 'city': 'Stockholm', 'age': 28}
{'_id': ObjectId('5df68a23f106fe2d315bbc8e'), 'name': 'Sami',
'country': 'Finland', 'city': 'Helsinki', 'age': 25}
```

Limiting documents

We can limit the number of documents we return using the *limit()* method.

```
# let's import the flask
from flask import Flask, render_template
import os # importing operating system module
import pymongo

MONGODB_URI =
'mongodb+srv://asabeneh:your_password_goes_here@30daysofpython
-twxkr.mongodb.net/test?retryWrites=true&w=majority'
client = pymongo.MongoClient(MONGODB_URI)
db = client['thirty_days_of_python'] # accessing the database
```

```
db.students.find().limit(3)
```

Find with sort

By default, sort is in ascending order. We can change the sorting to descending order by adding -1 parameter.

```
# let's import the flask
from flask import Flask, render_template
import os # importing operating system module
import pymongo

MONGODB_URI =
'mongodb+srv://asabeneh:your_password_goes_here@30daysofpython
-twxkr.mongodb.net/test?retryWrites=true&w=majority'
client = pymongo.MongoClient(MONGODB_URI)
db = client['thirty_days_of_python'] # accessing the database
students = db.students.find().sort('name')
for student in students:
    print(student)


students = db.students.find().sort('name',-1)
for student in students:
    print(student)

students = db.students.find().sort('age')
for student in students:
    print(student)

students = db.students.find().sort('age',-1)
for student in students:
    print(student)

app = Flask(__name__)
if __name__ == '__main__':
    # for deployment we use the environ
    # to make it work for both production and development
    port = int(os.environ.get("PORT", 5000))
    app.run(debug=True, host='0.0.0.0', port=port)
```

Ascending order

```
{'_id': ObjectId('5df68a21f106fe2d315bbc8b'), 'name':
'Asabeneh', 'country': 'Finland', 'city': 'Helsinki', 'age':
250}
{'_id': ObjectId('5df68a23f106fe2d315bbc8c'), 'name': 'David',
'country': 'UK', 'city': 'London', 'age': 34}
```

```
{'_id': ObjectId('5df68a23f106fe2d315bbc8d'), 'name': 'John',
'country': 'Sweden', 'city': 'Stockholm', 'age': 28}
{'_id': ObjectId('5df68a23f106fe2d315bbc8e'), 'name': 'Sami',
'country': 'Finland', 'city': 'Helsinki', 'age': 25}
```

Descending order

```
{'_id': ObjectId('5df68a23f106fe2d315bbc8e'), 'name': 'Sami',
'country': 'Finland', 'city': 'Helsinki', 'age': 25}
{'_id': ObjectId('5df68a23f106fe2d315bbc8d'), 'name': 'John',
'country': 'Sweden', 'city': 'Stockholm', 'age': 28}
{'_id': ObjectId('5df68a23f106fe2d315bbc8c'), 'name': 'David',
'country': 'UK', 'city': 'London', 'age': 34}
{'_id': ObjectId('5df68a21f106fe2d315bbc8b'), 'name':
'Asabeneh', 'country': 'Finland', 'city': 'Helsinki', 'age':
250}
```

Update with query

We will use *update_one()* method to update one item. It takes two object one is a query and the second is the new object. The first person, Asabeneh got a very implausible age. Let us update Asabeneh's age.

```
# let's import the flask
from flask import Flask, render_template
import os # importing operating system module
import pymongo

MONGODB_URI =
'mongodb+srv://asabeneh:your_password_goes_here@30daysofpython
-twxkr.mongodb.net/test?retryWrites=true&w=majority'
client = pymongo.MongoClient(MONGODB_URI)
db = client['thirty_days_of_python'] # accessing the database

query = {'age':250}
new_value = {'$set':{'age':38}}

db.students.update_one(query, new_value)
# lets check the result if the age is modified
for student in db.students.find():
    print(student)


app = Flask(__name__)
if __name__ == '__main__':
    # for deployment we use the environ
    # to make it work for both production and development
    port = int(os.environ.get("PORT", 5000))
```

```
    app.run(debug=True, host='0.0.0.0', port=port)
```

```
{'_id': ObjectId('5df68a21f106fe2d315bbc8b'), 'name':
'Asabeneh', 'country': 'Finland', 'city': 'Helsinki', 'age':
38}
{'_id': ObjectId('5df68a23f106fe2d315bbc8c'), 'name': 'David',
'country': 'UK', 'city': 'London', 'age': 34}
{'_id': ObjectId('5df68a23f106fe2d315bbc8d'), 'name': 'John',
'country': 'Sweden', 'city': 'Stockholm', 'age': 28}
{'_id': ObjectId('5df68a23f106fe2d315bbc8e'), 'name': 'Sami',
'country': 'Finland', 'city': 'Helsinki', 'age': 25}
```

When we want to update many documents at once we use *upate_many()* method.

Delete Document

The method *delete_one()* deletes one document. The *delete_one()* takes a query object parameter. It only removes the first occurrence. Let us remove one John from the collection.

```
# let's import the flask
from flask import Flask, render_template
import os # importing operating system module
import pymongo

MONGODB_URI =
'mongodb+srv://asabeneh:your_password_goes_here@30daysofpython
-twxkr.mongodb.net/test?retryWrites=true&w=majority'
client = pymongo.MongoClient(MONGODB_URI)
db = client['thirty_days_of_python'] # accessing the database

query = {'name':'John'}
db.students.delete_one(query)

for student in db.students.find():
    print(student)
# lets check the result if the age is modified
for student in db.students.find():
    print(student)


app = Flask(__name__)
if __name__ == '__main__':
    # for deployment we use the environ
    # to make it work for both production and development
    port = int(os.environ.get("PORT", 5000))
    app.run(debug=True, host='0.0.0.0', port=port)
```

```
{'_id': ObjectId('5df68a21f106fe2d315bbc8b'), 'name':
'Asabeneh', 'country': 'Finland', 'city': 'Helsinki', 'age':
38}
{'_id': ObjectId('5df68a23f106fe2d315bbc8c'), 'name': 'David',
'country': 'UK', 'city': 'London', 'age': 34}
{'_id': ObjectId('5df68a23f106fe2d315bbc8e'), 'name': 'Sami',
'country': 'Finland', 'city': 'Helsinki', 'age': 25}
```

As you can see John has been removed from the collection.

When we want to delete many documents we use *delete_many()* method, it takes a query object. If we pass an empty query object to *delete_many({})* it will delete all the documents in the collection.

Drop a collection

Using the *drop()* method we can delete a collection from a database.

```
# let's import the flask
from flask import Flask, render_template
import os # importing operating system module
import pymongo

MONGODB_URI =
'mongodb+srv://asabeneh:your_password_goes_here@30daysofpython
-twxkr.mongodb.net/test?retryWrites=true&w=majority'
client = pymongo.MongoClient(MONGODB_URI)
db = client['thirty_days_of_python'] # accessing the database
db.students.drop()
```

Now, we have deleted the students collection from the database.

💻 Exercises: Day 27

🎉 CONGRATULATIONS ! 🎉

DAY-28 API

API

API stands for Application Programming Interface. The kind of API we will cover in this section is going to be Web APIs. Web APIs are the defined interfaces through which interactions happen between an enterprise and applications that use its assets, which also is a Service Level Agreement (SLA) to specify the functional provider and expose the service path or URL for its API users.

In the context of web development, an API is defined as a set of specifications, such as Hypertext Transfer Protocol (HTTP) request messages, along with a definition of the structure of response messages, usually in an XML or a JavaScript Object Notation (JSON) format.

Web API has been moving away from Simple Object Access Protocol (SOAP) based web services and service-oriented architecture (SOA) towards more direct representational state transfer (REST) style web resources.

Social media services, web APIs have allowed web communities to share content and data between communities and different platforms.

Using API, content that is created in one place dynamically can be posted and updated to multiple locations on the web.

For example, Twitter's REST API allows developers to access core Twitter data and the Search API provides methods for developers to interact with Twitter Search and trends data.

Many applications provide API end points. Some examples of API such as the countries API, cat's breed API.

In this section, we will cover a RESTful API that uses HTTP request methods to GET, PUT, POST and DELETE data.

Building API

RESTful API is an application program interface (API) that uses HTTP requests to GET, PUT, POST and DELETE data. In the previous sections, we have learned about python, flask and mongoDB. We will use the knowledge we acquire to develop a RESTful API using Python flask and mongoDB database. Every application which has CRUD(Create, Read, Update, Delete) operation has an API to create data, to get data, to update data or to delete data from a database.

To build an API, it is good to understand HTTP protocol and HTTP request and response cycle.

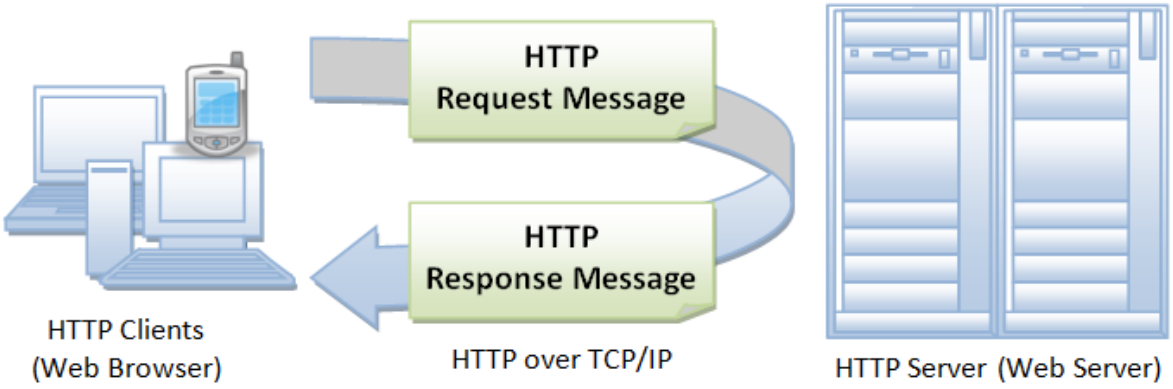
HTTP(Hypertext Transfer Protocol)

HTTP is an established communication protocol between a client and a server. A client in this case is a browser and server is the place where you access data. HTTP is a network protocol used to deliver resources which could be files on the World Wide Web, whether they are HTML files, image files, query results, scripts, or other file types.

A browser is an HTTP client because it sends requests to an HTTP server (Web server), which then sends responses back to the client.

Structure of HTTP

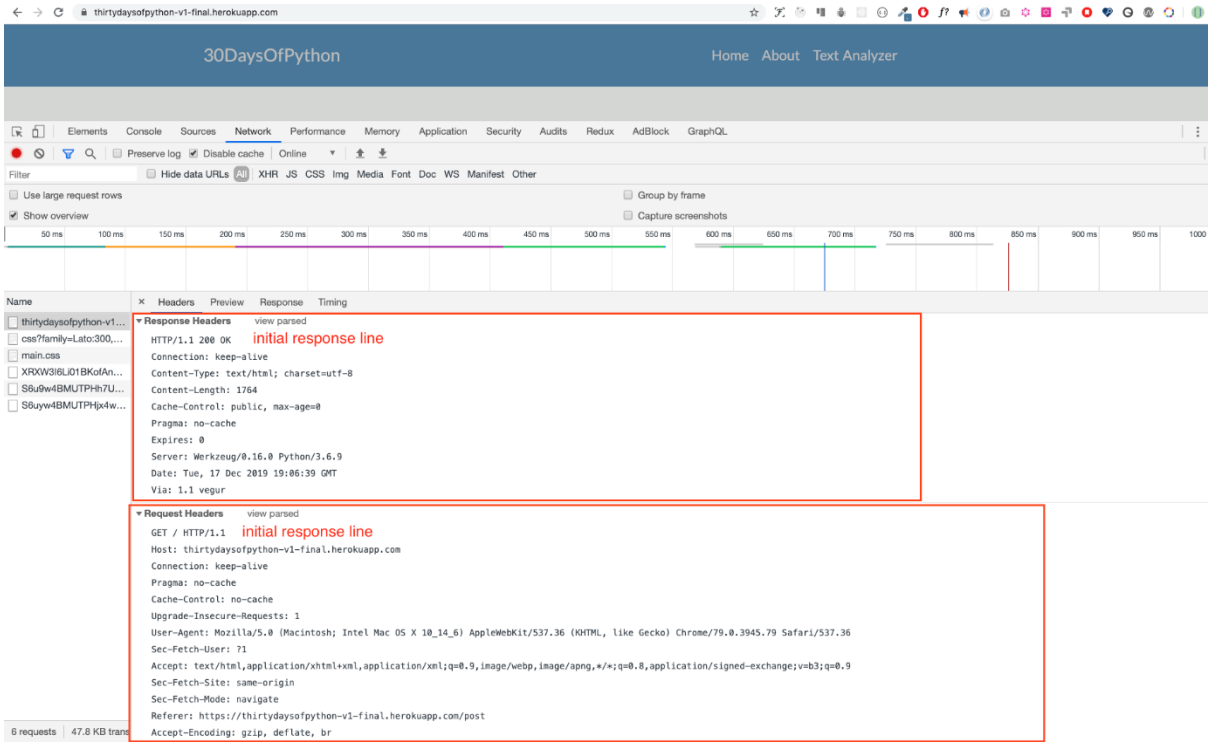
HTTP uses client-server model. An HTTP client opens a connection and sends a request message to an HTTP server and the HTTP server returns response message which is the requested resources. When the request response cycle completes the server closes the connection.



The format of the request and response messages are similar. Both kinds of messages have

- an initial line,
- zero or more header lines,
- a blank line (i.e. a CRLF by itself), and
- an optional message body (e.g. a file, or query data, or query output).

Let us an example of request and response messages by navigating this site:https://thirtydaysofpython-v1-final.herokuapp.com/. This site has been deployed on Heroku free dyno and in some months may not work because of high request. Support this work to make the server run all the time.



Initial Request Line(Status Line)

The initial request line is different from the response. A request line has three parts, separated by spaces:

- method name(GET, POST, HEAD)
- path of the requested resource,
- the version of HTTP being used. eg GET / HTTP/1.1

GET is the most common HTTP that helps to get or read resource and POST is a common request method to create resource.

Initial Response Line(Status Line)

The initial response line, called the status line, also has three parts separated by spaces:

- HTTP version
- Response status code that gives the result of the request, and a reason which describes the status code. Example of status lines are: HTTP/1.0 200 OK or HTTP/1.0 404 Not Found Notes:

The most common status codes are: 200 OK: The request succeeded, and the resulting resource (e.g. file or script output) is returned in the message body. 500 Server Error A complete list of HTTP status code can be found here. It can be also found here.

Header Fields

As you have seen in the above screenshot, header lines provide information about the request or response, or about the object sent in the message body.

```
GET / HTTP/1.1
Host: thirtydaysofpython-v1-final.herokuapp.com
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/79.0.3945.79
Safari/537.36
Sec-Fetch-User: ?1
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/we
bp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Referer: https://thirtydaysofpython-v1-
final.herokuapp.com/post
Accept-Encoding: gzip, deflate, br
Accept-Language: en-GB,en;q=0.9,fi-FI;q=0.8,fi;q=0.7,en-
CA;q=0.6,en-US;q=0.5,fr;q=0.4
```

The message body

An HTTP message may have a body of data sent after the header lines. In a response, this is where the requested resource is returned to the client (the most common use of the message body), or perhaps explanatory text if there's an error. In a request, this is where user-entered data or uploaded files are sent to the server.

If an HTTP message includes a body, there are usually header lines in the message that describe the body. In particular,

The Content-Type: header gives the MIME-type of the data in the body(text/html, application/json, text/plain, text/css, image/gif). The Content-Length: header gives the number of bytes in the body.

Request Methods

The GET, POST, PUT and DELETE are the HTTP request methods which we are going to implement an API or a CRUD operation application.

1. GET: GET method is used to retrieve and get information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data.
2. POST: POST request is used to create data and send data to the server, for example, creating a new post, file upload, etc. using HTML forms.
3. PUT: Replaces all current representations of the target resource with the uploaded content and we use it modify or update data.
4. DELETE: Removes data

💻 Exercises: Day 28

1. Read about API and HTTP

🎉 CONGRATULATIONS ! 🎉

DAY-29 BUILDING API

Building API

In this section, we will cove a RESTful API that uses HTTP request methods to GET, PUT, POST and DELETE data.

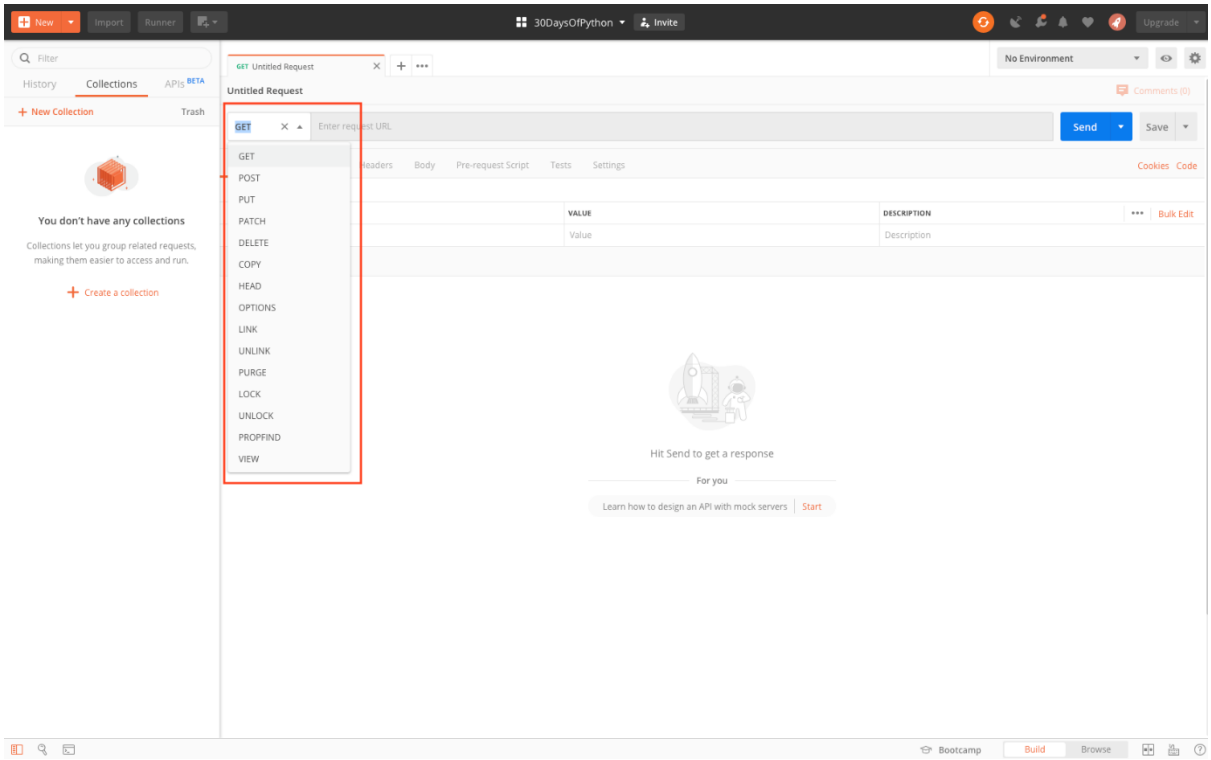
RESTful API is an application program interface (API) that uses HTTP requests to GET, PUT, POST and DELETE data. In the previous sections, we have learned about python, flask and mongoDB. We will use the knowledge we acquire to develop a RESTful API using python flask and mongoDB. Every application which has CRUD(Create, Read, Update, Delete) operation has an API to create data, to get data, to update data or to delete data from database.

The browser can handle only get request. Therefore, we have to have a tool which can help us to handle all request methods(GET, POST, PUT, DELETE).

Examples of API

- Countries API: https://restcountries.eu/rest/v2/all
- Cats breed API: https://api.thecatapi.com/v1/breeds

Postman is a very popular tool when it comes to API development. So, if you like to do this section you need to download postman. An alternative of Postman is Insomnia.



Structure of an API

An API end point is a URL which can help to retrieve, create, update or delete a resource. The structure looks like this:
Example: https://api.twitter.com/1.1/lists/members.json Returns the members of the specified list. Private list members will only be shown if the authenticated user owns the specified list. The name of the company name followed by version followed by the purpose of the API. The methods: HTTP methods & URLs

The API uses the following HTTP methods for object manipulation:

```
GET         Used for object retrieval
POST        Used for object creation and object actions
PUT         Used for object update
DELETE      Used for object deletion
```

Let us build an API which collects information about 30DaysOfPython students. We will collect the name, country, city, date of birth, skills and bio.

To implement this API, we will use:

- Postman
- Python
- Flask
- MongoDB

Retrieving data using get

In this step, let us use dummy data and return it as a json. To return it as json, will use json module and Response module.

```
# let's import the flask

from flask import Flask,  Response
import json

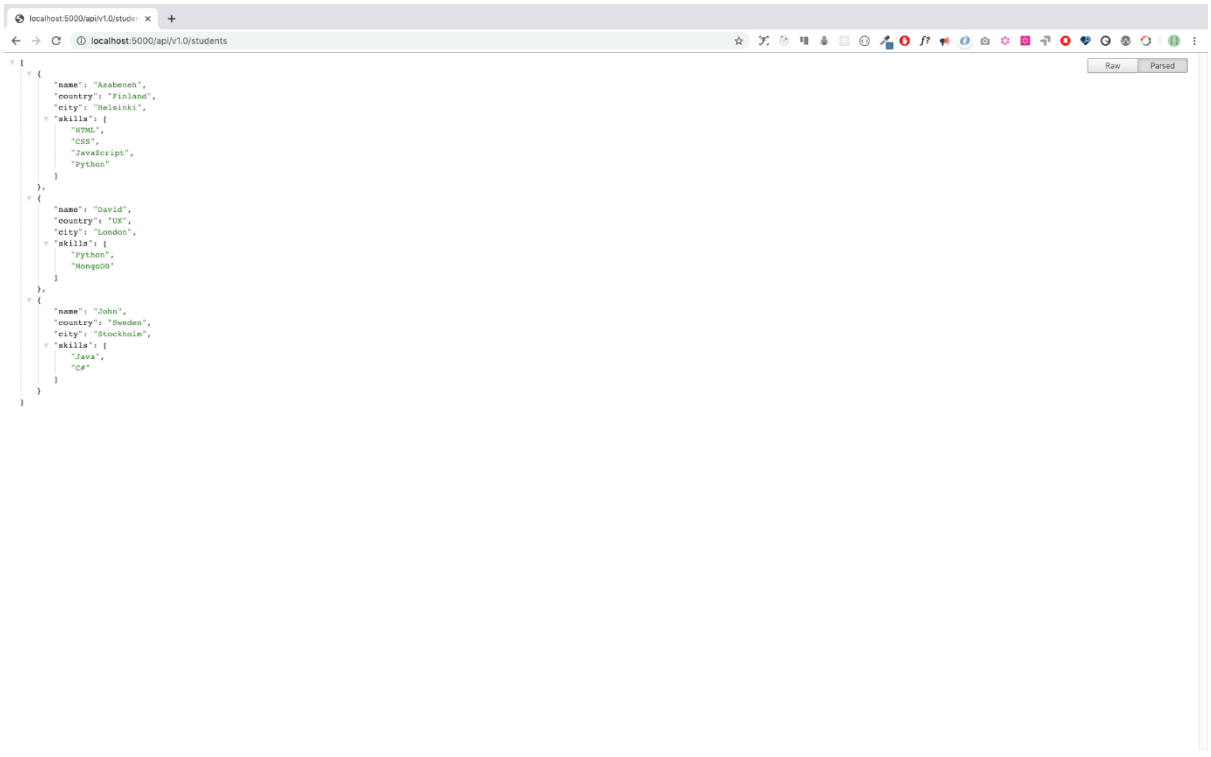
app = Flask(__name__)

@app.route('/api/v1.0/students', methods = ['GET'])
def students ():
    student_list = [
        {
            'name':'Asabeneh',
            'country':'Finland',
            'city':'Helsinki',
            'skills':['HTML', 'CSS','JavaScript','Python']
        },
        {
            'name':'David',
            'country':'UK',
            'city':'London',
```

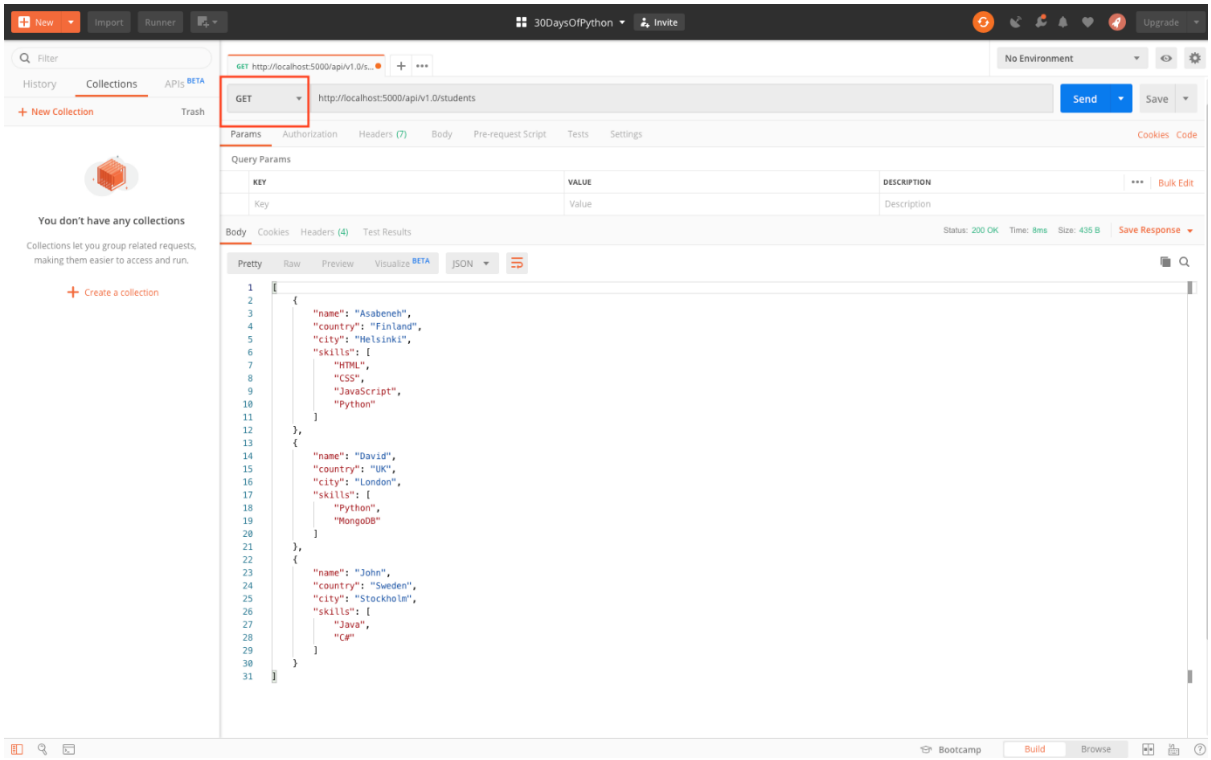
```
            'skills':['Python','MongoDB']
        },
        {
            'name':'John',
            'country':'Sweden',
            'city':'Stockholm',
            'skills':['Java','C#']
        }
    ]
    return Response(json.dumps(student_list),
mimetype='application/json')


if __name__ == '__main__':
    # for deployment
    # to make it work for both production and development
    port = int(os.environ.get("PORT", 5000))
    app.run(debug=True, host='0.0.0.0', port=port)
```

When you request the http://localhost:5000/api/v1.0/students url on the browser you will get this:



When you request the http://localhost:5000/api/v1.0/students url on the browser you will get this:



In stead of displaying dummy data let us connect the flask application with MongoDB and get data from mongoDB database.

```
# let's import the flask

from flask import Flask,  Response
import json
import pymongo


app = Flask(__name__)

#
MONGODB_URI='mongodb+srv://asabeneh:your_password@30daysofpyth
on-twxkr.mongodb.net/test?retryWrites=true&w=majority'
client = pymongo.MongoClient(MONGODB_URI)
db = client['thirty_days_of_python'] # accessing the database

@app.route('/api/v1.0/students', methods = ['GET'])
def students ():

    return Response(json.dumps(student),
mimetype='application/json')


if __name__ == '__main__':
    # for deployment
    # to make it work for both production and development
    port = int(os.environ.get("PORT", 5000))
```

```
    app.run(debug=True, host='0.0.0.0', port=port)
```

By connecting the flask, we can fetch students collection data from the thirty_days_of_python database.

```
[
    {
        "_id": {
            "$oid": "5df68a21f106fe2d315bbc8b"
        },
        "name": "Asabeneh",
        "country": "Finland",
        "city": "Helsinki",
        "age": 38
    },
    {
        "_id": {
            "$oid": "5df68a23f106fe2d315bbc8c"
        },
        "name": "David",
        "country": "UK",
        "city": "London",
        "age": 34
    },
    {
        "_id": {
            "$oid": "5df68a23f106fe2d315bbc8e"
        },
        "name": "Sami",
        "country": "Finland",
        "city": "Helsinki",
        "age": 25
    }
]
```

Getting a document by id

We can access signle document using an id, let's access Asabeneh using his id. http://localhost:5000/api/v1.0/students/5df68a21f106fe2d315bbc8b

```
# let's import the flask

from flask import Flask,  Response
import json
from bson.objectid import ObjectId
import json
from bson.json_util import dumps
import pymongo
```

```
app = Flask(__name__)

#
MONGODB_URI='mongodb+srv://asabeneh:your_password@30daysofpyth
on-twxkr.mongodb.net/test?retryWrites=true&w=majority'
client = pymongo.MongoClient(MONGODB_URI)
db = client['thirty_days_of_python'] # accessing the database

@app.route('/api/v1.0/students', methods = ['GET'])
def students ():

    return Response(json.dumps(student),
mimetype='application/json')
@app.route('/api/v1.0/students/<id>', methods = ['GET'])
def single_student (id):
    student = db.students.find({'_id':ObjectId(id)})
    return Response(dumps(student),
mimetype='application/json')

if __name__ == '__main__':
    # for deployment
    # to make it work for both production and development
    port = int(os.environ.get("PORT", 5000))
    app.run(debug=True, host='0.0.0.0', port=port)
```

```
[
    {
        "_id": {
            "$oid": "5df68a21f106fe2d315bbc8b"
        },
        "name": "Asabeneh",
        "country": "Finland",
        "city": "Helsinki",
        "age": 38
    }
]
```

Creating data using POST

We use the POST request method to create data

```
# let's import the flask

from flask import Flask,  Response
import json
from bson.objectid import ObjectId
import json
from bson.json_util import dumps
import pymongo
from datetime import datetime
```

```
app = Flask(__name__)

#
MONGODB_URI='mongodb+srv://asabeneh:your_password@30daysofpyth
on-twxkr.mongodb.net/test?retryWrites=true&w=majority'
client = pymongo.MongoClient(MONGODB_URI)
db = client['thirty_days_of_python'] # accessing the database

@app.route('/api/v1.0/students', methods = ['GET'])
def students ():

    return Response(json.dumps(student),
mimetype='application/json')
@app.route('/api/v1.0/students/<id>', methods = ['GET'])
def single_student (id):
    student = db.students.find({'_id':ObjectId(id)})
    return Response(dumps(student),
mimetype='application/json')
@app.route('/api/v1.0/students', methods = ['POST'])
def create_student ():
    name = request.form['name']
    country = request.form['country']
    city = request.form['city']
    skills = request.form['skills'].split(', ')
    bio = request.form['bio']
    birthyear = request.form['birthyear']
    created_at = datetime.now()
    student = {
        'name': name,
        'country': country,
        'city': city,
        'birthyear': birthyear,
        'skills': skills,
        'bio': bio,
        'created_at': created_at

    }
    db.students.insert_one(student)
    return ;
def update_student (id):
if __name__ == '__main__':
    # for deployment
    # to make it work for both production and development
    port = int(os.environ.get("PORT", 5000))
    app.run(debug=True, host='0.0.0.0', port=port)
```

Updating using PUT

```
# let's import the flask

from flask import Flask,  Response
import json
from bson.objectid import ObjectId
import json
from bson.json_util import dumps
import pymongo
from datetime import datetime


app = Flask(__name__)

#
MONGODB_URI='mongodb+srv://asabeneh:your_password@30daysofpyth
on-twxkr.mongodb.net/test?retryWrites=true&w=majority'
client = pymongo.MongoClient(MONGODB_URI)
db = client['thirty_days_of_python'] # accessing the database

@app.route('/api/v1.0/students', methods = ['GET'])
def students ():

    return Response(json.dumps(student),
mimetype='application/json')
@app.route('/api/v1.0/students/<id>', methods = ['GET'])
def single_student (id):
    student = db.students.find({'_id':ObjectId(id)})
    return Response(dumps(student),
mimetype='application/json')
@app.route('/api/v1.0/students', methods = ['POST'])
def create_student ():
    name = request.form['name']
    country = request.form['country']
    city = request.form['city']
    skills = request.form['skills'].split(', ')
    bio = request.form['bio']
    birthyear = request.form['birthyear']
    created_at = datetime.now()
    student = {
        'name': name,
        'country': country,
        'city': city,
        'birthyear': birthyear,
        'skills': skills,
        'bio': bio,
        'created_at': created_at

    }
    db.students.insert_one(student)
```

```
    return
@app.route('/api/v1.0/students/<id>', methods = ['PUT']) #
this decorator create the home route
def update_student (id):
    query = {"_id":ObjectId(id)}
    name = request.form['name']
    country = request.form['country']
    city = request.form['city']
    skills = request.form['skills'].split(', ')
    bio = request.form['bio']
    birthyear = request.form['birthyear']
    created_at = datetime.now()
    student = {
        'name': name,
        'country': country,
        'city': city,
        'birthyear': birthyear,
        'skills': skills,
        'bio': bio,
        'created_at': created_at

    }
    db.students.update_one(query, student)
    # return Response(dumps({"result":"a new student has been
created"}), mimetype='application/json')
    return
def update_student (id):
if __name__ == '__main__':
    # for deployment
    # to make it work for both production and development
    port = int(os.environ.get("PORT", 5000))
    app.run(debug=True, host='0.0.0.0', port=port)
```

Deleting a document using Delete

```
# let's import the flask

from flask import Flask,  Response
import json
from bson.objectid import ObjectId
import json
from bson.json_util import dumps
import pymongo
from datetime import datetime


app = Flask(__name__)

#
```

```
MONGODB_URI='mongodb+srv://asabeneh:your_password@30daysofpyth
on-twxkr.mongodb.net/test?retryWrites=true&w=majority'
client = pymongo.MongoClient(MONGODB_URI)
db = client['thirty_days_of_python'] # accessing the database

@app.route('/api/v1.0/students', methods = ['GET'])
def students ():

    return Response(json.dumps(student),
mimetype='application/json')
@app.route('/api/v1.0/students/<id>', methods = ['GET'])
def single_student (id):
    student = db.students.find({'_id':ObjectId(id)})
    return Response(dumps(student),
mimetype='application/json')
@app.route('/api/v1.0/students', methods = ['POST'])
def create_student ():
    name = request.form['name']
    country = request.form['country']
    city = request.form['city']
    skills = request.form['skills'].split(', ')
    bio = request.form['bio']
    birthyear = request.form['birthyear']
    created_at = datetime.now()
    student = {
        'name': name,
        'country': country,
        'city': city,
        'birthyear': birthyear,
        'skills': skills,
        'bio': bio,
        'created_at': created_at

    }
    db.students.insert_one(student)
    return
@app.route('/api/v1.0/students/<id>', methods = ['PUT']) #
this decorator create the home route
def update_student (id):
    query = {"_id":ObjectId(id)}
    name = request.form['name']
    country = request.form['country']
    city = request.form['city']
    skills = request.form['skills'].split(', ')
    bio = request.form['bio']
    birthyear = request.form['birthyear']
    created_at = datetime.now()
    student = {
        'name': name,
        'country': country,
        'city': city,
```

```
        'birthyear': birthyear,
        'skills': skills,
        'bio': bio,
        'created_at': created_at

    }
    db.students.update_one(query, student)
    # return Response(dumps({"result":"a new student has been
created"}), mimetype='application/json')
    return
@app.route('/api/v1.0/students/<id>', methods = ['PUT']) #
this decorator create the home route
def update_student (id):
    query = {"_id":ObjectId(id)}
    name = request.form['name']
    country = request.form['country']
    city = request.form['city']
    skills = request.form['skills'].split(', ')
    bio = request.form['bio']
    birthyear = request.form['birthyear']
    created_at = datetime.now()
    student = {
        'name': name,
        'country': country,
        'city': city,
        'birthyear': birthyear,
        'skills': skills,
        'bio': bio,
        'created_at': created_at

    }
    db.students.update_one(query, student)
    # return Response(dumps({"result":"a new student has been
created"}), mimetype='application/json')
    return ;
@app.route('/api/v1.0/students/<id>', methods = ['DELETE'])
def delete_student (id):
    db.students.delete_one({"_id":ObjectId(id)})
    return
if __name__ == '__main__':
    # for deployment
    # to make it work for both production and development
    port = int(os.environ.get("PORT", 5000))
    app.run(debug=True, host='0.0.0.0', port=port)
```

💻 Exercises: Day 29

1. Implement the above example and develop this

🍾 CONGRATULATIONS ! 🍾

DAY-30 CONCLUSIONS

Congratulations on reaching the last page of the book. Now the next steps is to make projects, take reference of out 100 projects of python guide.