Learning Objectives

Introduction to Python Programming Language

Anaconda installation on windows

Anaconda installation on Linux

Familiar with jupyter notebook

Overview of jupyter notebook

Python variables

Python built-in Data Types

Python Methods and functions

Operators in Python

Different type of Python Statements and loops

Python Expression Statements

Introduction to Python Programming Language

What is python?

- Python is an interpreted, interactive, object-oriented programming language.
- It incorporates modules, exceptions, dynamic typing, very high-level dynamic data types and classes.
- It supports multiple programming paradigms beyond object-oriented programming, such as procedural and functional programming.

- Example:

  print(“Hello world”)
- Output:

  Hello world

In [17]:
```
print("Hello world")
```

```
Hello world
```

Applications of python

GUI-based desktop applications

Graphic design, image processing applications, Games, and Scientific/ computational Applications

Web frameworks and applications

Enterprise and Business applications

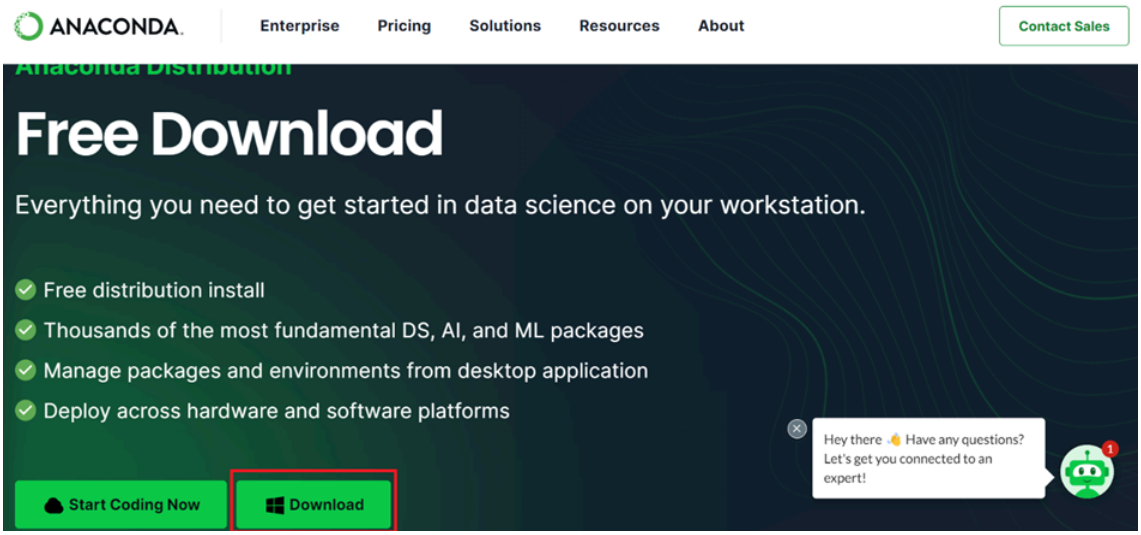
Operating Systems

Education

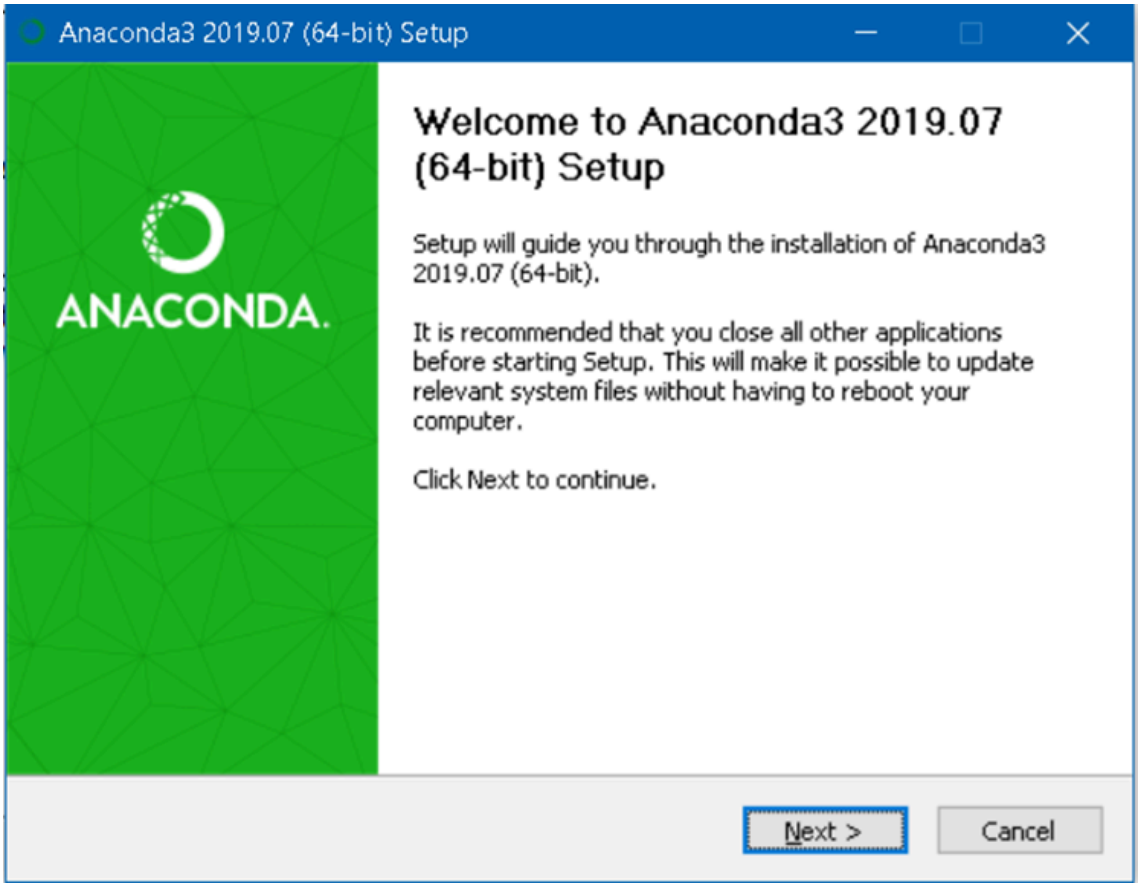
Python setup – Getting started with programming

Anaconda installation on windows

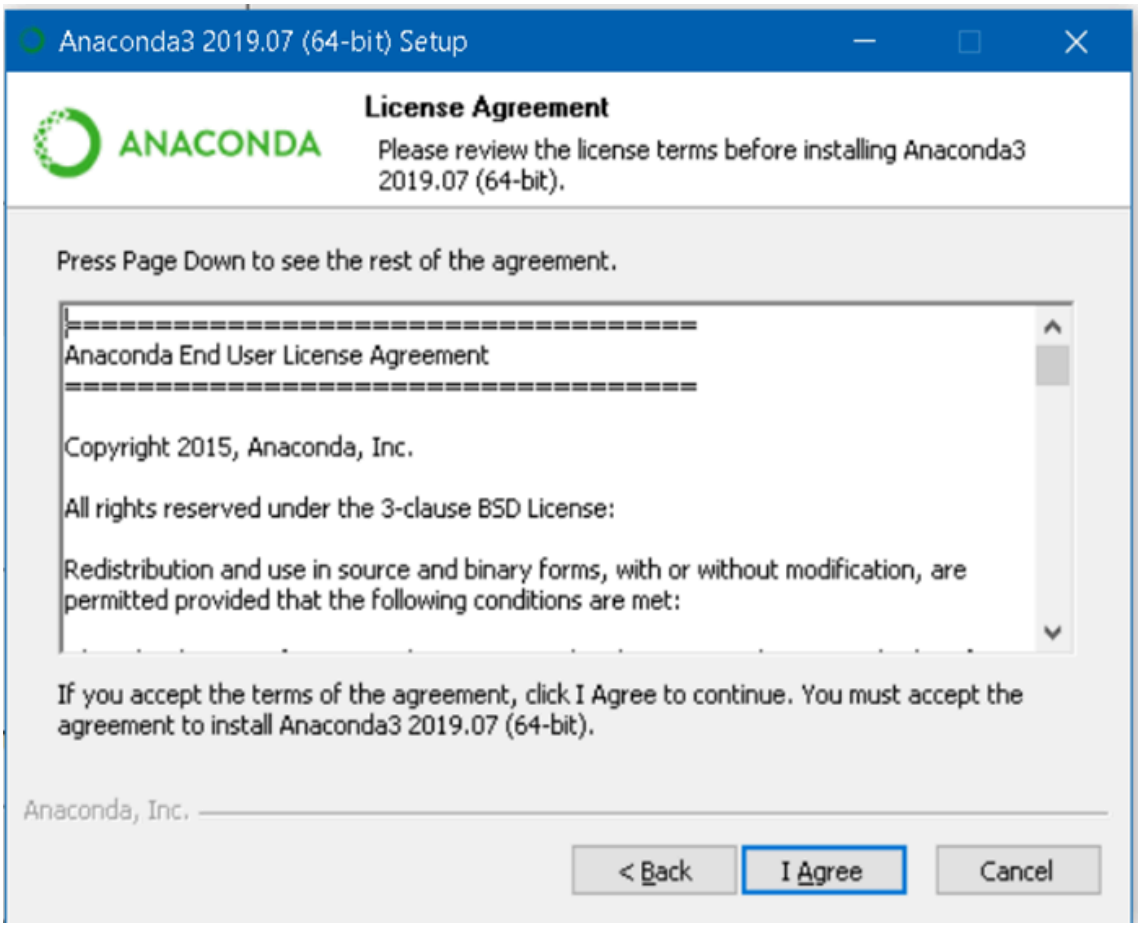
Step 1- At first, visit the following link: https://www.anaconda.com/download (https://www.anaconda.com/download) and the page will pop up like this, just click on Download.



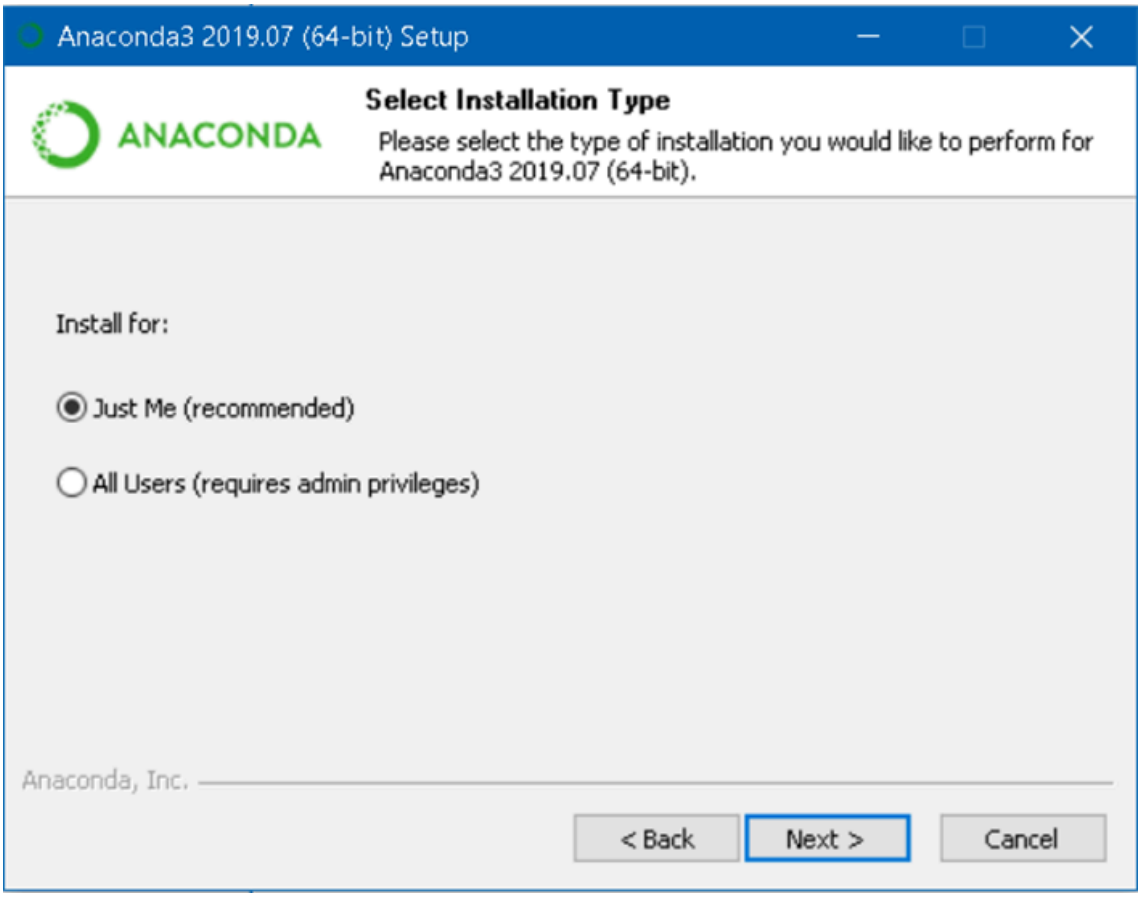
Step 2- After downloading the file, run the file. The file will open, Click Next



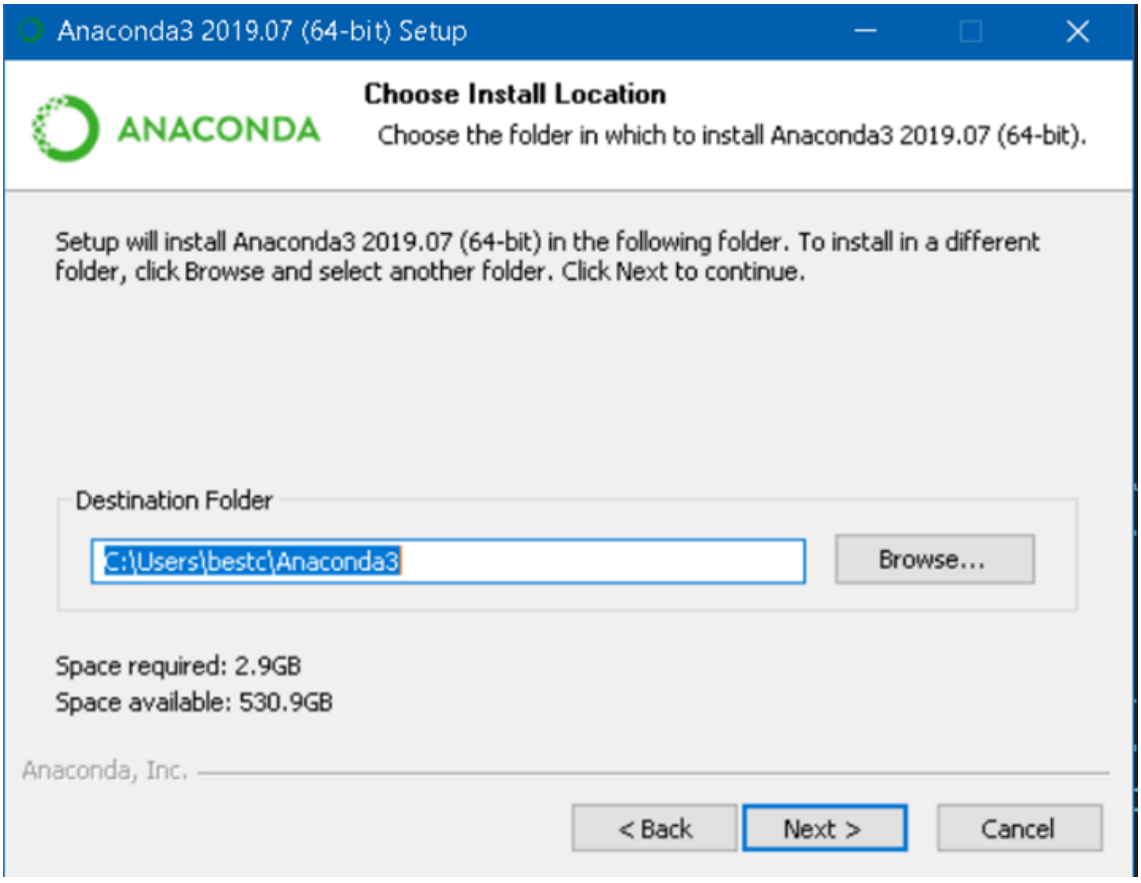
Step 3- And click I Agree to the license.



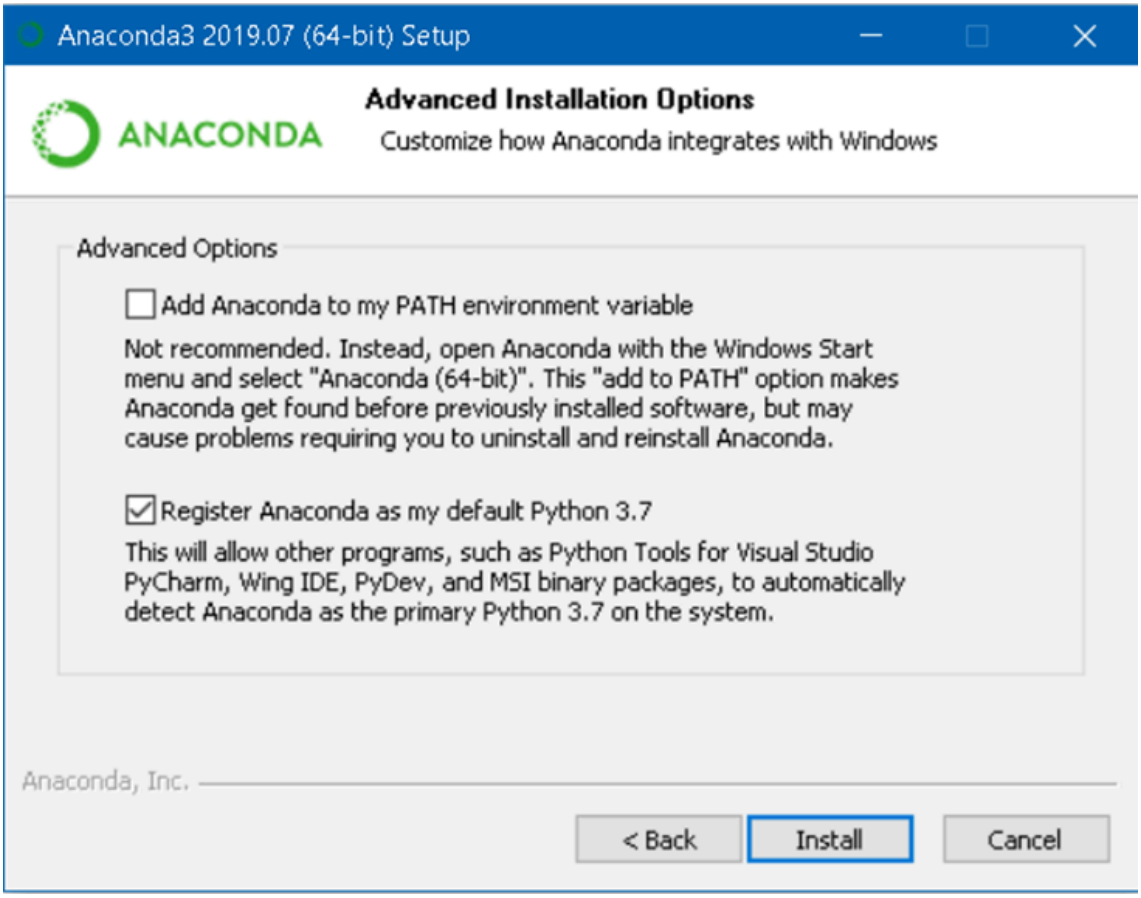
Step 4- Choose Just Me and click Next.



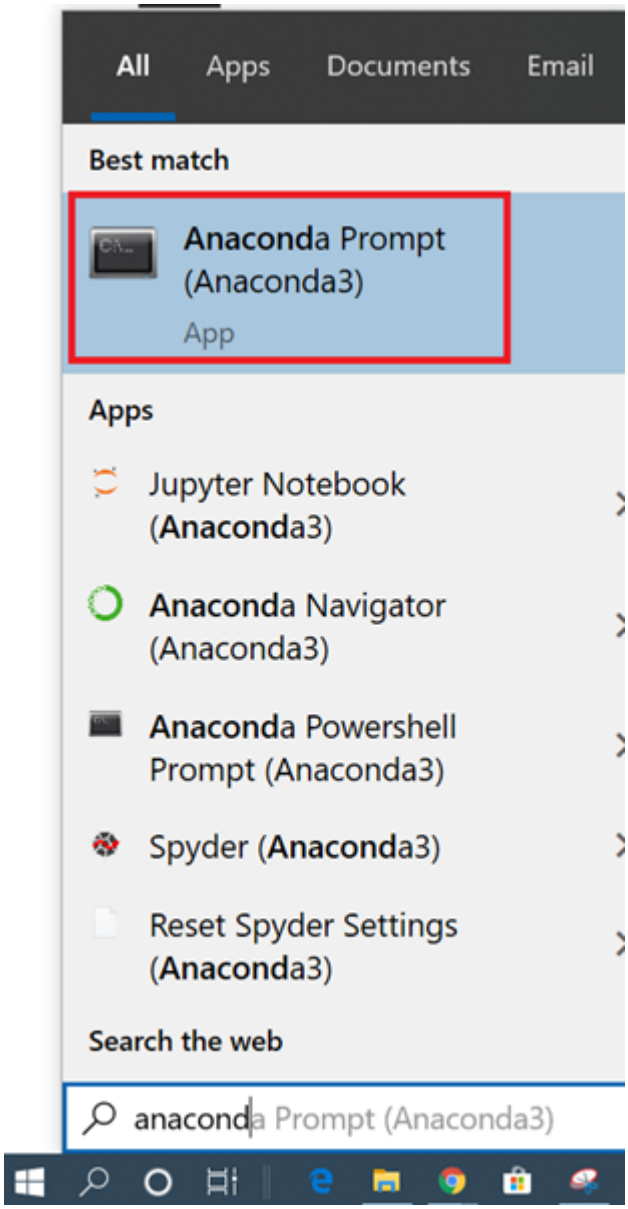
Step 5- Choose the installation location by clicking Browse or leave it as it is (default location) and continue to click Next.



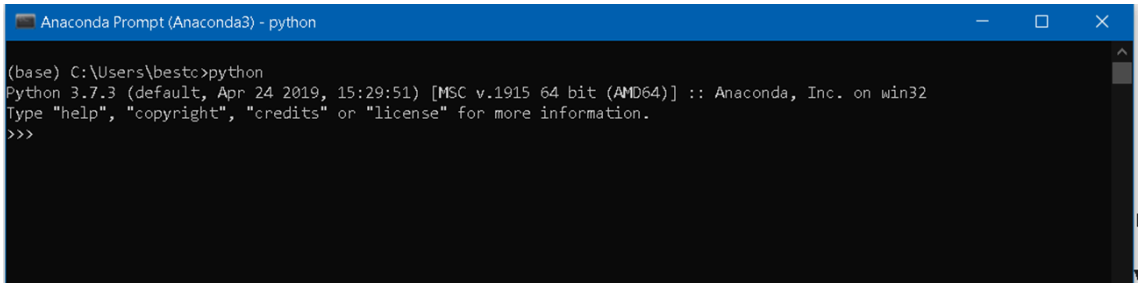
Step 6- Here, it is highly recommended to choose the second one “Register Anaconda as my default Python 3.7” and click Install.



Step 7- Once the installation is done, open the Anaconda Prompt from Windows start menu bar.

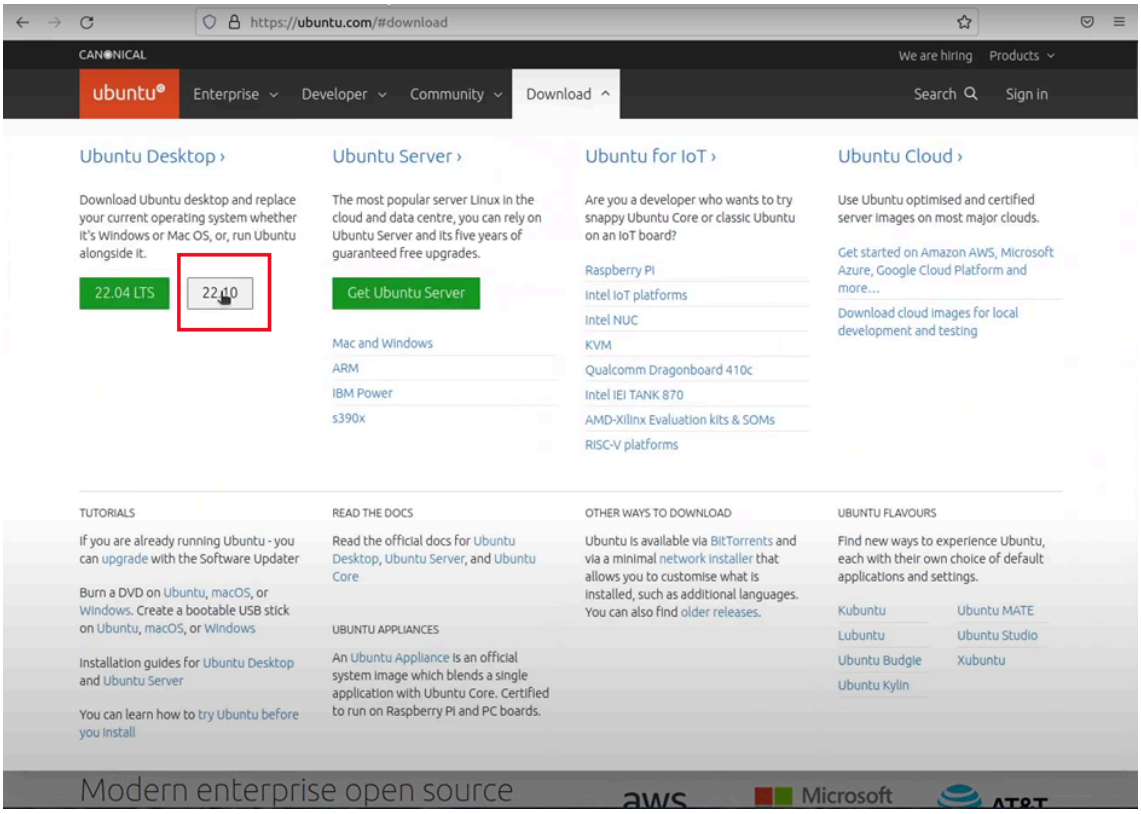


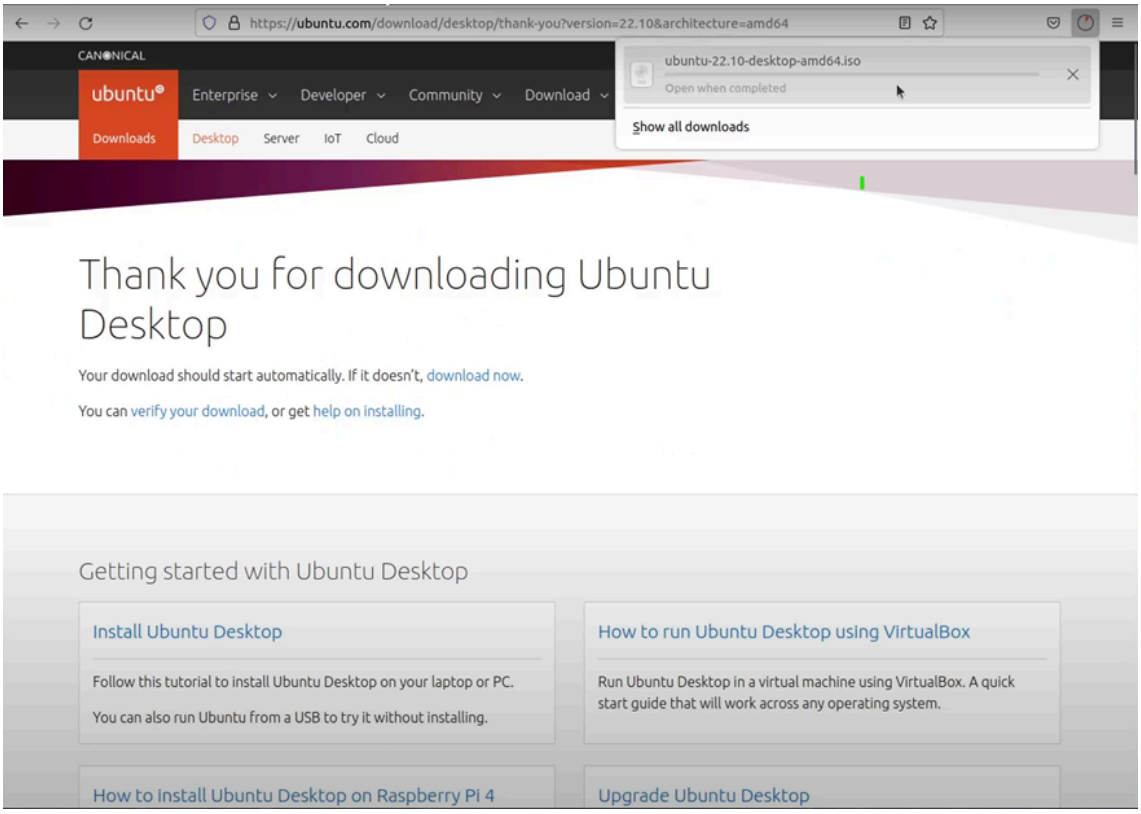
Step 8- Anaconda Prompt is shell similar to Windows Command Prompt (Windows Terminal) powered by Anaconda distribution. To check whether we have successfully installed Anaconda or not, type python command in the shell.



Anaconda installation on Linux

Step 1- At first, visit the following link: https://ubuntu.com/#download (https://ubuntu.com/#download) and the page will pop up like this. Click 22.10 or whatever version available





Step 2- Once it is downloaded go to Download folder and run .sh file

```
karthick@LinuxShellTips:~/Downloads$ ls -l
total 557480
-rw-rw-r-- 1 karthick karthick 570853747 Jun 22 18:39 Anaconda3-2021.05-Linux-x86_64.sh
karthick@LinuxShellTips:~/Downloads$ sha256sum Anaconda3-2021.05-Linux-x86_64.sh
2751ab3d678ff0277ae80f9e8a74f218cfc70fe9a9cdc7bb1c137d7e47e33d53  Anaconda3-2021.05-Linux-x86_64.sh
```

Step 3- Now run the downloaded .sh file to install anaconda. As a first step, it will ask you to read the license agreement once you press enter.

```
karthick@LinuxShellTips:~/Downloads$ bash Anaconda3-2021.05-Linux-x86_64.sh

Welcome to Anaconda3 2021.05

In order to continue the installation process, please review the license
agreement.
Please, press ENTER to continue
>>>
```

Step 4- In the next step, it will ask you to choose a location where the anaconda will be installed. It defaults to your home directory.

```
Anaconda3 will now be installed into this location:
/home/karthick/anaconda3

  - Press ENTER to confirm the location
  - Press CTRL-C to abort the installation
  - Or specify a different location below

[/home/karthick/anaconda3] >>>
```

Step 5- Packages will be installed and once the installation is completed it will ask to initialize Anaconda3 by running conda init. It defaults to No. You can choose Yes or No depending upon how you need it.

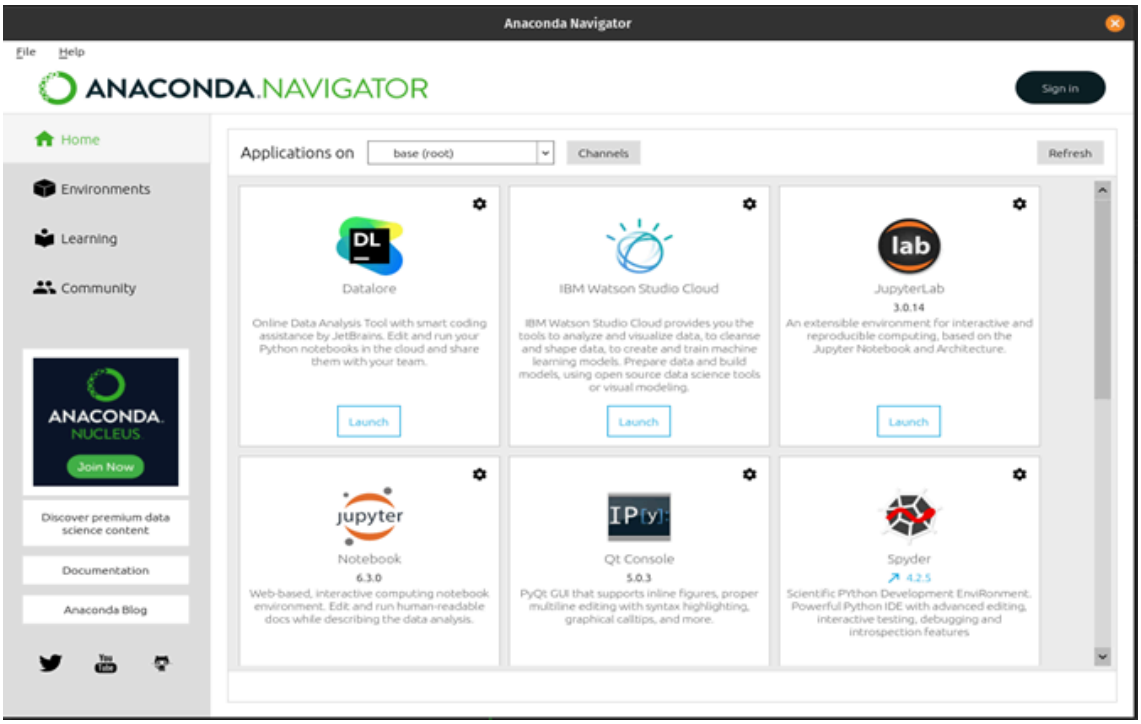
```
Preparing transaction: done
Executing transaction: done
installation finished.
Do you wish the installer to initialize Anaconda3
by running conda init? [yes|no]
[no] >>>

You have chosen to not have conda modify your shell scripts at all.
To activate conda's base environment in your current shell session:

eval "$(/home/karthick/anaconda3/bin/conda shell.YOUR_SHELL_NAME hook)"

To install conda's shell functions for easier access, first activate, then:

conda init

If you'd prefer that conda's base environment not be activated on startup,
   set the auto_activate_base parameter to false:

conda config --set auto_activate_base false

Thank you for installing Anaconda3!

===========================================================================

Working with Python and Jupyter notebooks is a breeze with PyCharm Pro,
designed to be used with Anaconda. Download now and have the best data
tools at your fingertips.

PyCharm Pro for Anaconda is available at: https://www.anaconda.com/pycharm
```

Step 6- Go to the directory where anaconda is installed and under the bin directory, there is a binary called “anaconda-navigator”. This will launch the GUI program for anaconda from where you can launch your tools.ck!

Write the following command-

$ /home/karthick/anaconda3/bin/anaconda-navigator

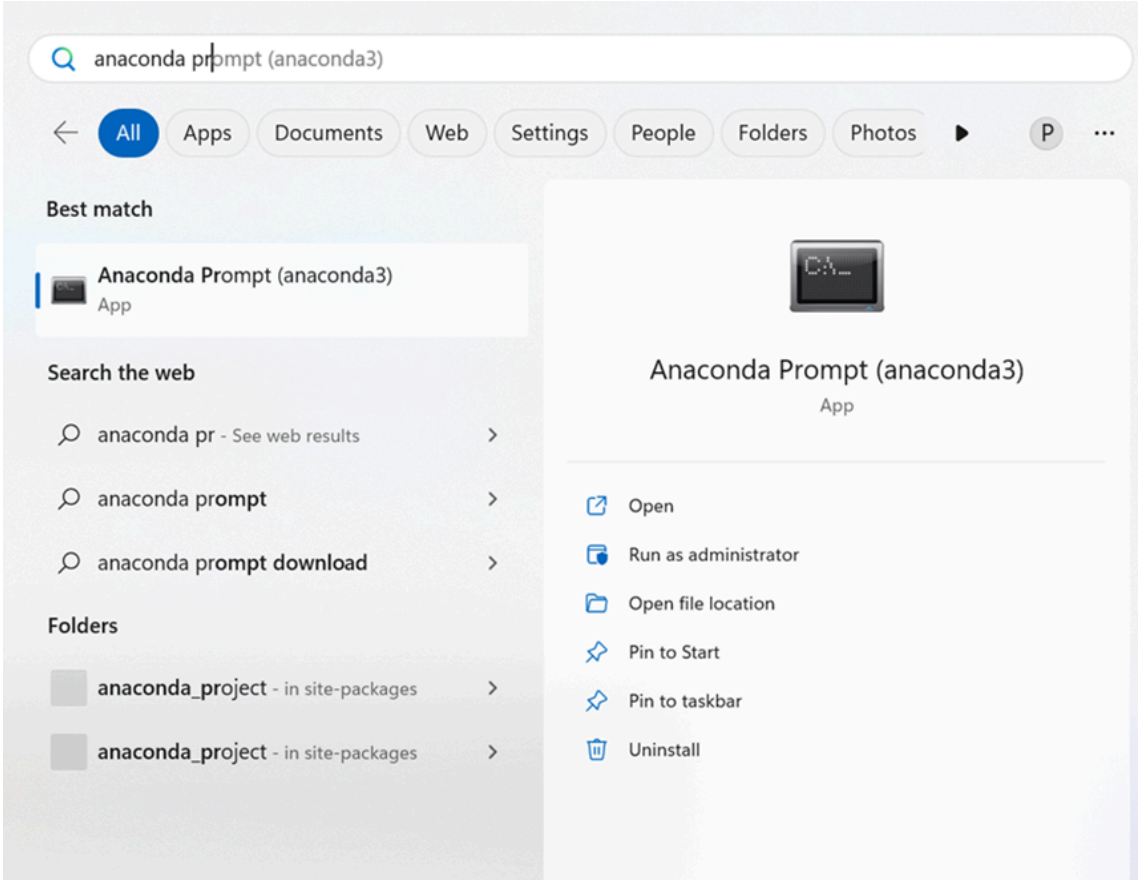
ANACONDA NAVIGATOR will launch as shown in img.



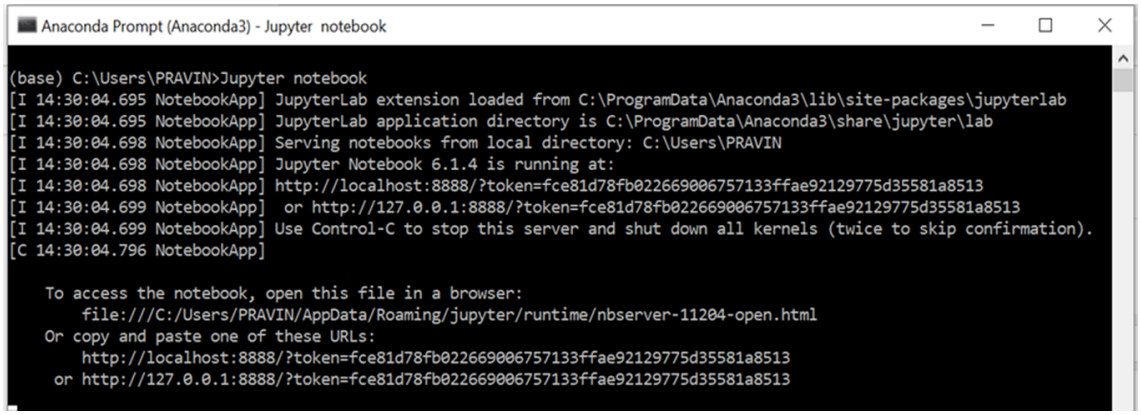
Familiar with Jupyter Notebook

The Jupyter Notebook is an open-source web application that you can use to create and share documents that contain live code, equations, visualizations, and text. Jupyter Notebook is maintained by the people at Project Jupyter. If you already installed Anaconda in your machine, then it’s very easy to use Jupyter notebook

Step 1- Press window key and Just type anaconda prompt and open.

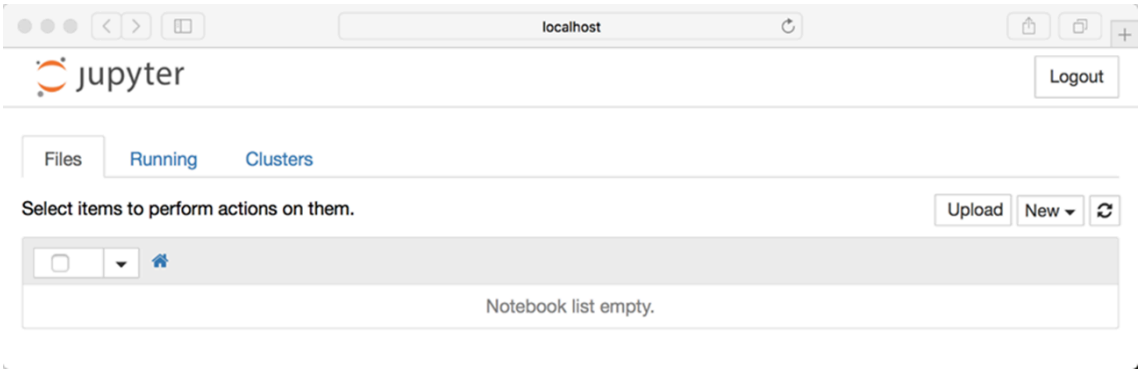


Step 2- Just Run command Jupyter notebook and hit enter



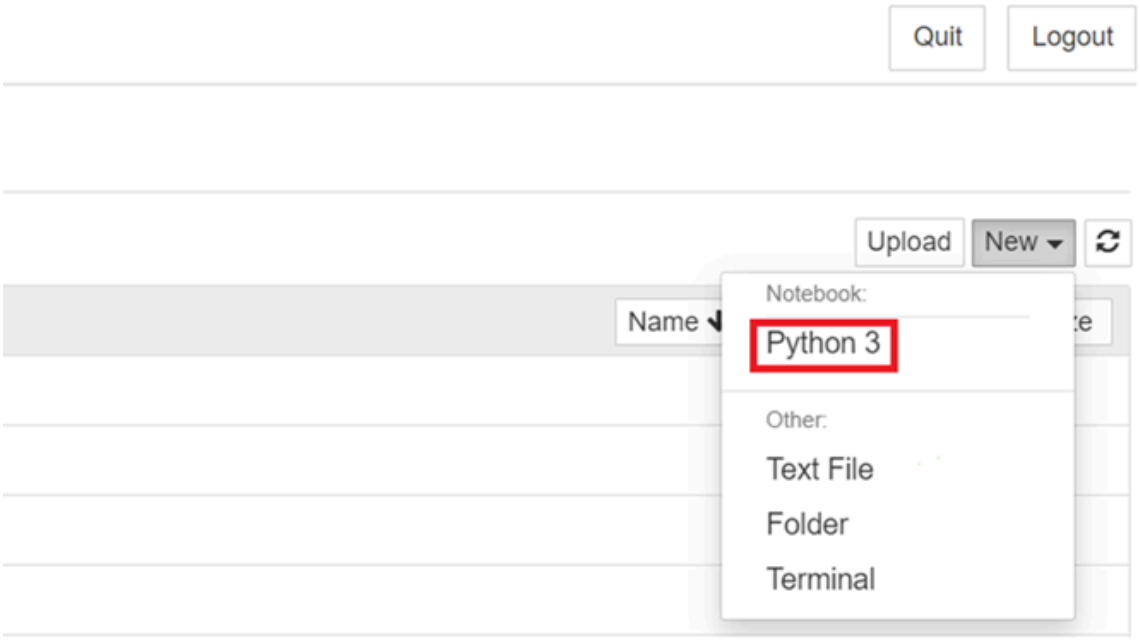
Jupyter notebook will open in your default browser, should start (or open a new tab) to the following URL: [http://localhost:8888/tree (http://localhost:8888/tree)](http://localhost:8888/tree)

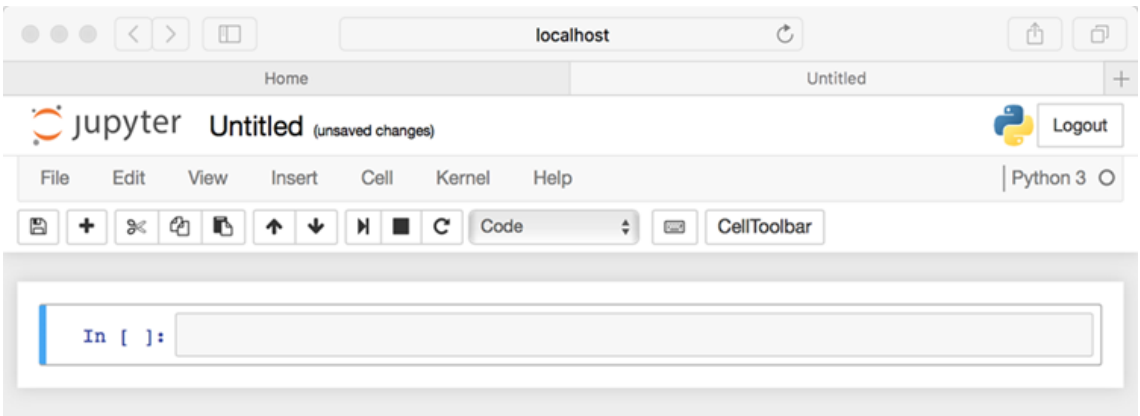
Your browser should now look something like this:



Step 3- To creating a notebook, Click on New and choose Python3 Now that you know how to start a Notebook server, you should probably learn how to create an actual Notebook document.

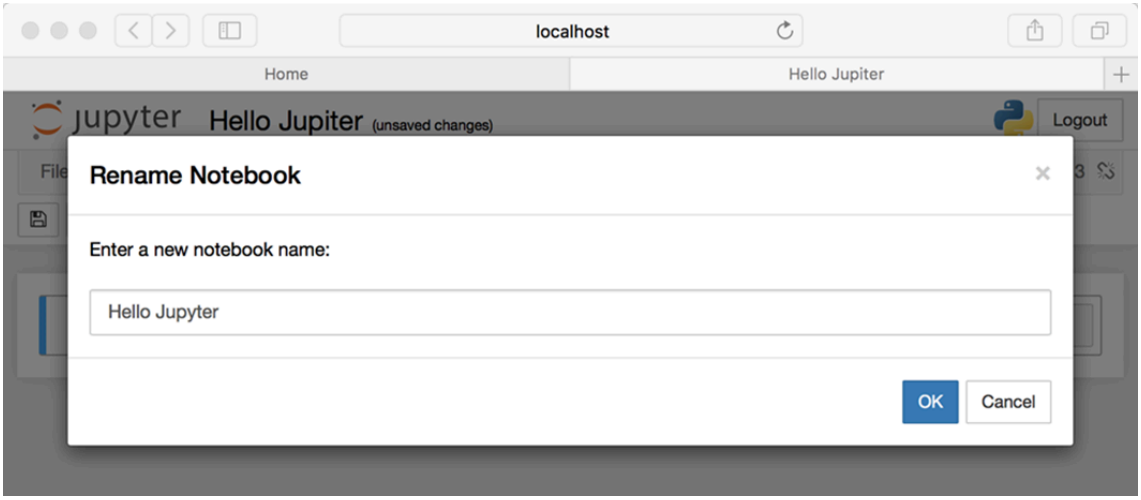
All you need to do is click on the new button (upper right), and it will open up a list of choices. Here choose python2 or Python 3, so we can create a Notebook that uses either of these. For simplicity's sake, let's choose Python 3.





Step 4- Naming You will notice that at the top of the page is the word Untitled. This is the title for the page and the name of your Notebook. Since that isn't a very descriptive name, let's change it! Just move your mouse over the word Untitled and click on the text. You

should now see an in-browser dialog titled Rename Notebook. Let's rename this one to



Step 5- Running Cells Running a cell means that you will execute the cell's contents. To execute a cell, you can just select the cell and click the Run button that is in the row of buttons along the top. It's towards the middle. If you prefer using your keyboard, you can just press Shift+Enter.

In [1]:
```
from platform import python_version

print(python_version())
```

```
3.9.13
```

In [17]:
```
print("Hello Jupyter")
```

```
Hello Jupyter
```

Note: Python version check

Comments in python

Writing the comments in Python is very easy. The comments in Python start with the '#' mark.

In [18]:
```
# This is a comment
print("Hello World")
```

```
Hello World
```

Jupyter Notebook overview

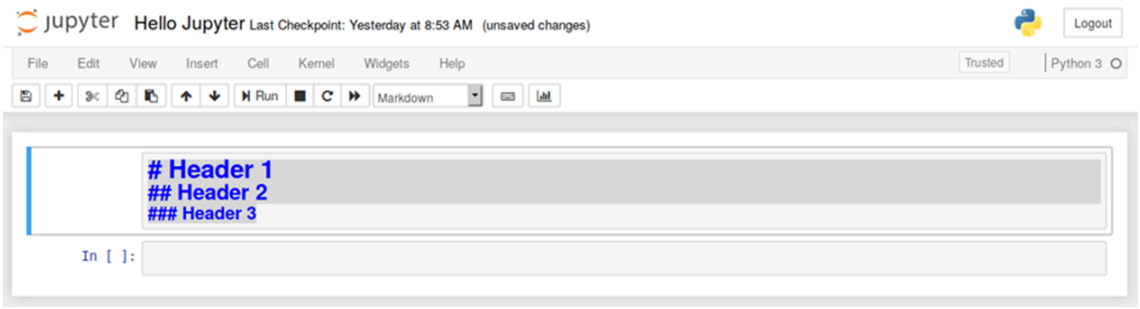
Step 1- In the Jupyter Notebook has several menus that you can use to interact with your Notebook. The menu runs along the top of the Notebook just like menus do in other applications. Here is a list of the current menus:

- File
- Edit

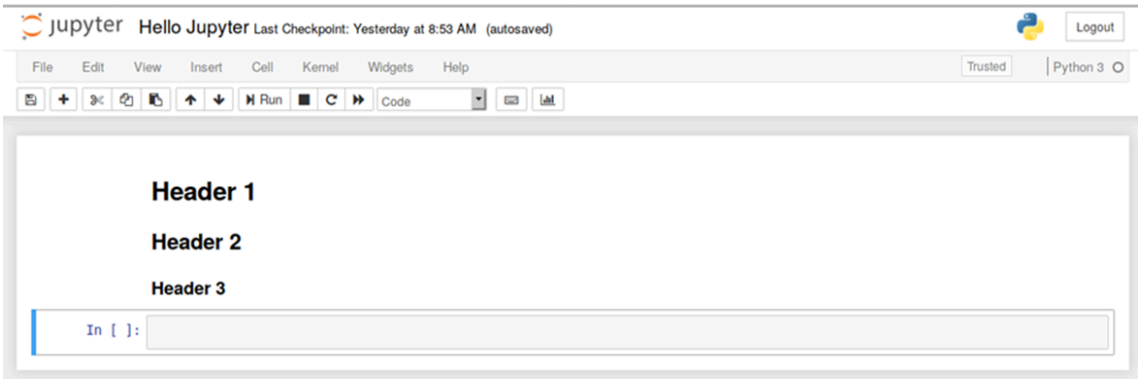
- View
- Insert
- Cell
- Kernel
- Widgets
- Help

Explore all the Menu one by one to use of Jupyter notebook.

Step 2- How to create larger or smaller headers Creating headers in Markdown is also quite simple. You just have to use the humble Hash sign. The more Hash signs you use, the smaller the header. Jupyter Notebook even kind of previews it for you:



Step 3- Then when you run(shift+enter) the cell, you will end up with a nicely formatted header:



Exporting Notebooks When you are working with Jupyter Notebooks, you will find that you need to share your results with non-technical people. When that happens, you can use the nbconvert tool which comes with Jupyter Notebook to convert or export your Notebook into one of the following formats:

- HTML
- LaTeX
- PDF
- RevealJS
- Markdown
- ReStructured Text
- Executable script

The nbconvert tool uses Jinja templates under the covers to convert your Notebook files (.ipynb) into these other formats.

Notebook Extensions

While Jupyter Notebooks have lots of functionality built in, you can add new functionality through extensions. Jupyter actually supports four types of extensions:KernelIPython kernel NotebookNotebook server

Conclusion

- The Jupyter Notebook is quite useful not only for learning and teaching a programming language such as Python but also for sharing your data.
- You can turn your Notebook into a slideshow or share it online with GitHub. If you want to share a Notebook without requiring your users to install anything, you can use binder for that.

Python Variables

- A variable is a container for a value. It can be assigned a name; you can use it to refer to it later in the program. Based on the data type of the variable.
- The interpreter allocates memory and decides what can be stored in the reserved memory.

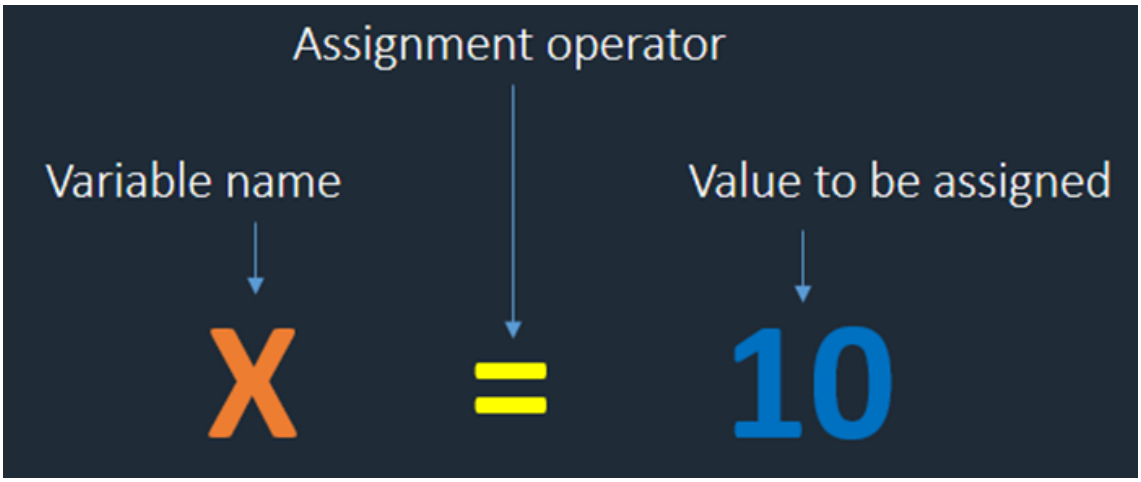


Image source: https://www.aipython.in/python-variables/

Rules for variable names

Not allowed naming conventions are:

- Variable name cannot start with numbers (e.g. -> 0_a, 1_test)
- White spaces (blanks) cannot be present in the variable name (e.g. -> a b, my variable). The most common practice is to fill the gap with an underscore character.
- Special sign/symbol (“+” and “-“) cannot be used in variable name (e.g. -> a+b , x-y)

- Python Reserve keyword cannot be used for naming a variable. (e.g. -> if, for, while etc)
- Variables names are case-sensitive. (e.g.-> a and A is different)

Let's define some variables

In [37]:
```
myvar = "Hello"
my_var = "Hello"
_my_var = "Hello"
myVar = "Hello"
```

In [38]:
```
print(myvar)
print(my_var)
print(_my_var)
print(myVar)
```

```
Hello
Hello
Hello
Hello
```

Types of variables

- There are two types of variables

1. Local Variable
2. Global Variable

1. Local Variables

- Variable which are inside the function is known as local variable. Let's consider an example.

In [42]:
```
def hello():
    a = 10          # Here a is local variable
    print(a)
    return
hello()
```

```
10
```

Note:- Only focus on the concept, functions are explained in depth later.

2. Global Variables

- A variable declared outside of the function or in global scope is known as global variable.Let's consider an example.

In [44]:
```
a = "Welcome to Code Unnati 2.0" # Here a is global variable
def hello():
    a = 20
    print(a)  # Here a is local variable
    return
print(a)
```

```
Welcome to Code Unnati 2.0
```

Keywords in python

Python reserves 33 keywords in 3.3 versions for its use. Keywords are case sensitive in python. You can’t use a keyword as variable name, function name or any other identifier name. Here is the list of keywords in python.

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

Image source: https://miro.medium.com/v2/resize:fit:939/1*XAEgNgrxV1waXEe-vXilzA.png

Python built-in data types

Standard Data Types

- The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters.
- Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

Python has five standard data types:

- Numbers
- String
- List
- Tuple
- Dictionary

Python Numbers

Image source:https://www.freepik.com/

Number data types store numeric values. Number objects are created when you assign a value to them.

In [57]:
```
#This is integer datatype
a = 10
print(a)
print(type(a))
```

```
10
<class 'int'>
```

In [58]:
```
#Multiple assignment
b,c,d = 11, 12, 13
```

In [202]:
```
print(a,b,c,d)
```

```
10 11 12 13
```

Operators

Arithematic operator

In [203]:
```
# addition, sub, mul, div: +, -, *, / etc
x = 5
y = 3

print(x + y)
```

```
8
```

Assignment operator

In [204]:
```
x = 5
x += 3

print(x)
```

```
8
```

In [205]:
```
x = 5

x *= 3

print(x)
```

```
15
```

Comaprison operator

In [206]:
```
x = 5
y = 3

print(x == y)

# returns False because 5 is not equal to 3
```

```
False
```

In [207]:
```
x = 5
y = 3

print(x != y)

# returns True because 5 is not equal to 3
```

```
True
```

Logical operator

In [208]:
```
x = 5

print(x > 3 and x < 10)

# returns True because 5 is greater than 3 AND 5 is less than 10
```

```
True
```

In [209]:
```
x = 5

print(x > 3 or x < 4)

# returns True because one of the conditions are true (5 is greater tha
```

```
True
```

In [210]:
```
x = 5

print(not(x > 3 and x < 10))

# returns False because not is used to reverse the result
```

```
False
```

In [212]:
```
#use input keyword to take input from user

#aks user to enter the value and store it in varialbe x

x = int(input("Enter the value of x : "))

#aks user to enter the value and store it in varialbe y

y = int(input("Enter the value of y : "))

#perform multiplication of x and y and store answer in variable Z

z = x*y

#Print answer

print(z)
```

```
Enter the value of x : 2
Enter the value of y : 1
2
```

Now, let's discuss more about the operators

Precedence Level	Operator	Explanation
1 (highest)	( )	Parentheses
2	**	Exponentiation
3	-a, +a	Negative, positive argument
4	* , / , // , %, @	Multiplication, division, floor division, modulus, at
5	+, -	Addition, subtraction
6	< , <=, >, >=, ==, !=	Less than, less than or equal, greater, greater or equal, equal, not equal
7	not	Boolean Not
8	and	Boolean And
9	or	Boolean Or

Image source: https://global.discourse-cdn.com/codecademy/original/5X/a/6/5/9/a6596ee00717c2b98c57de7e4450d6c22e8c67ec.p (https://global.discourse-cdn.com/codecademy/original/5X/a/6/5/9/a6596ee00717c2b98c57de7e4450d6c22e8c67ec.p

Python Strings

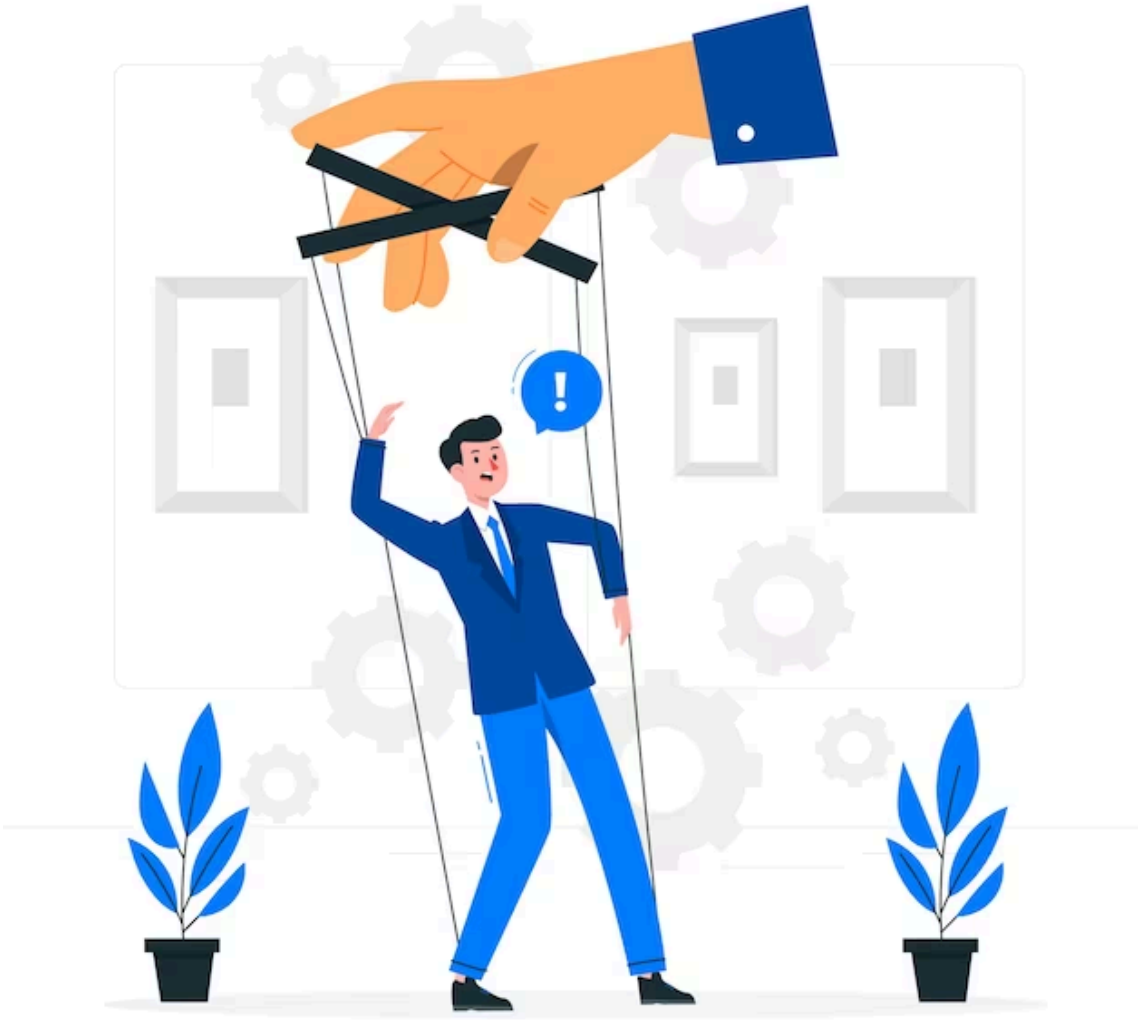


Image source:https://www.freepik.com/

- Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows either pair of single or double quotes.
- Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 to the end.
- The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

In [43]:
```
statement = "      Python is a programming language      "
```

In [44]:
```
print(statement)
```

```
      Python is a programming language
```

In [45]:
```
print(type(statement)) #type is used to check the datatype
```

```
<class 'str'>
```

Methods in python strings

Lower Method

In [46]:
```
statement.lower() #to convert text into lowercase
```

Out[46]: '   python is a programming language   '

Upper Method

In [47]:
```
statement.upper()#to convert text into uppercase
```

Out[47]: '   PYTHON IS A PROGRAMMING LANGUAGE   '

Title Method

In [48]:
```
statement.title()#to convert text into snakewriting
```

Out[48]: '   Python Is A Programming Language   '

Strips Methods

In [49]:
```
statement.lstrip() #to remove the space from left side
```

Out[49]: 'Python is a programming language   '

In [50]:
```
statement.rstrip()#to remove the space from right side
```

Out[50]: '   Python is a programming language'

In [51]:
```
statement.strip()#to remove the space from both side
```

Out[51]: 'Python is a programming language'

String Formatting

- Strings in Python can be formatted with the use of format() method which is very versatile and powerful tool for formatting of Strings.
- Format method in String contains curly braces {} as placeholders which can hold arguments according to position or keyword to specify the order.

Let's understand it with an example

In [52]:
```
string = "{} {} {}".format('Python','Programming','Language')
string
```

Out[52]:
```
'Python Programming Language'
```

Here in above code, three curly empty placeholders provided to add the words like Python, Programming and Language respectively. Similarly you can try fstring method as well as follows:

In [64]:
```
name = 'Priyanka'
course = 'Code Unnti Program 2.0'
print(f"Hello, My name is {name} and I registered for {course}")
```

```
Hello, My name is Priyanka and I registered for Code Unnti Program 2.0
```

The idea behind f-strings is to make string interpolation simpler, to create an f-string, prefix the string with the letter “f ”.

The string itself can be formatted in much the same way that you would with str.format(). F-strings provide a concise and convenient way to embed python expressions inside string literals for formatting.

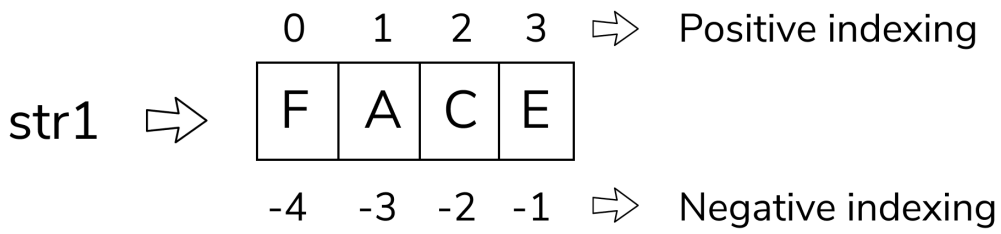
This are very few basic string methods you learnt. Additionally, python provides methods to work with strings. You can explore it more.

String Slicing

Syntax:

str1[Start:End:Steps]

In [265]:
```
str1 = "FACE"
```



In [266]:
```
str1[0]
```

Out[266]:
```
'F'
```

In [267]:
```
str1[2]
```

Out[267]:
```
'C'
```

In [268]:
```
str1[0:2]
```

Out[268]: 'FA'

In [269]:
```
str1[::-1]
```

Out[269]: 'ECAF'

Python Lists

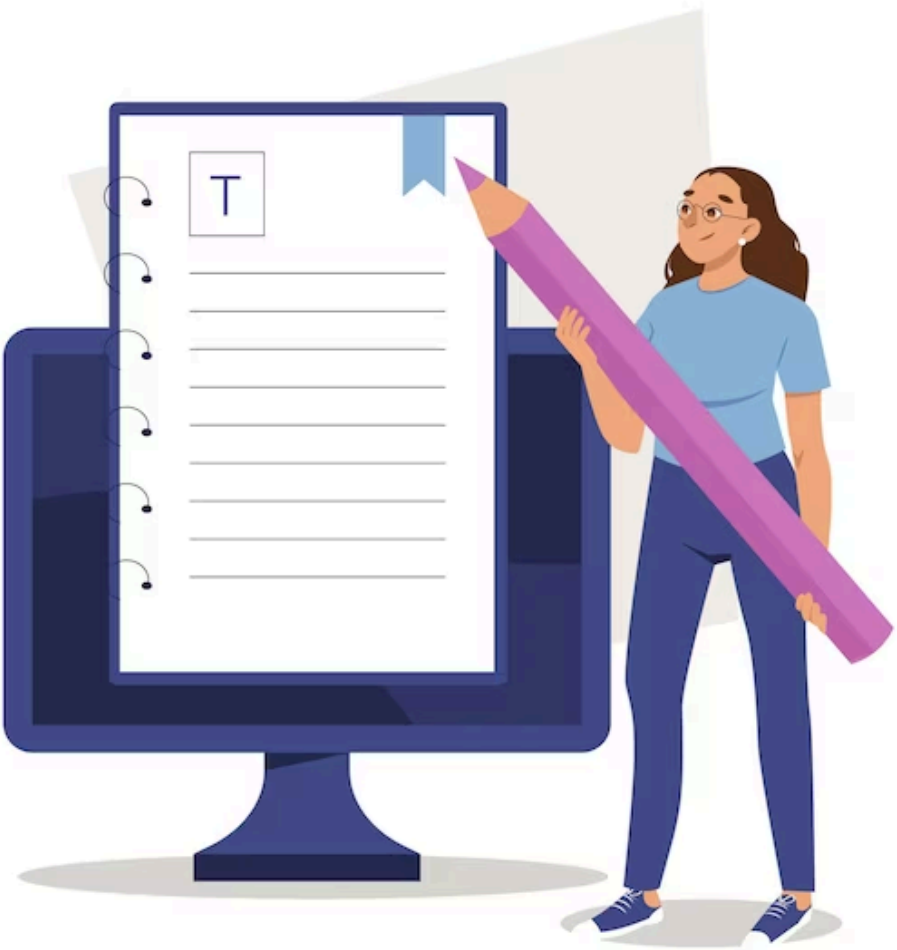


Image source:https://www.freepik.com/

- Lists are used to store multiple items in a single variable.
- List are mutable datatypes(changeable).
- Lists are created using square brackets
- List items are ordered and allow duplicate values.
- List items are indexed, the first item has index [0], the second item has index [1] etc.

Let's create first list

In [69]:
```
#create list

list1 = ["parul", 22, "Bharat", 45.5, "vadodara", 99]
print(lst1)

print(type(lst1))
```

```
['parul', 22, 'Bharat', 45.5, 'vadodara', 99]
<class 'list'>
```

In [71]:
```
list1[0] #access 1st position item from list
```

Out[71]:
```
'parul'
```

In [73]:
```
list1[3] #access 4th position item from list
```

Out[73]:
```
45.5
```

Methods in python lists

Append Method

The append() method appends an element to the end of the list.

Syntax:list.append(elmnt)

In [75]:
```
fruits = ['apple', 'banana', 'cherry']
fruits.append("orange")
print(fruits)
```

```
['apple', 'banana', 'cherry', 'orange']
```

Extend Method

The extend() method adds the specified list elements (or any iterable) to the end of the current list.

Syntax: list.extend(iterable)

In [76]:
```
fruits = ['apple', 'banana', 'cherry']
cars = ['Ford', 'BMW', 'Volvo']

fruits.extend(cars)
print(fruits)
```

```
['apple', 'banana', 'cherry', 'Ford', 'BMW', 'Volvo']
```

Insert Method

The insert() method inserts the specified value at the specified position.

Syntax: list.insert(index-pos, elmnt)

In [77]:
```
fruits = ['apple', 'banana', 'cherry']

fruits.insert(1, "orange") # insert orange at index 1
print(fruits)
```

```
['apple', 'orange', 'banana', 'cherry']
```

Pop Method

The pop() method removes the element at the specified position.

Syntax: list.pop(index-pos)

In [81]:
```
fruits = ['apple', 'banana', 'cherry']

fruits.pop(1) # if no argument pass in pop by default it will remove la
print(fruits)
```

```
['apple', 'cherry']
```

Remove Method

remove() method removes the first occurrence of the element with the specified value.

Syntax: list.remove(elmnt)

In [5]:
```
fruits = ['apple', 'banana', 'cherry']

fruits.remove("banana")
print(fruits)
```

```
['apple', 'cherry']
```

Sort Method

method sorts the list ascending by default. You can also make a function to decide the sorting criteria(s).

Syntax: list.sort(reverse=True|False, key=myFunc)

In [82]:
```
cars = ['Ford', 'BMW', 'Volvo']

cars.sort()  # The words will be arrange in aplphabetical order
print(cars)
```

```
['BMW', 'Ford', 'Volvo']
```

Split Method

It splits the sentence into words by seperating them in interverted commas

In [83]:
```
# write a program to take input a sentence and split it into word.

text = input("Enter the word: ")  # To take the input from the user "in
```

```
Enter the word: I love python programming language
```

In [85]:
```
text.split()
```

Out[85]:
```
['I', 'love', 'python', 'programming', 'language']
```

Python Tuples

- Tuples are Immutable datatypes(unchangeable)
- A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.
- The main differences between lists and tuples are: Lists are enclosed in brackets ([]) and their elements and size can be changed, while tuples are enclosed in parentheses (()) and cannot be updated. Tuples can be thought of as read-only lists.

Let's create first tuple

In [177]:
```
#create tuples -

tup1 = ("parul", 22, "Bharat", 45.5, "vadodara", 99)

print(tup1)
print(type(tup1))
```

```
('parul', 22, 'Bharat', 45.5, 'vadodara', 99)
<class 'tuple'>
```

In [178]:
```
tup1[4] # To access 5th item in tuple
```

Out[178]:
```
'vadodara'
```

In [179]:
```
tup1[2:4]
```

Out[179]:
```
('Bharat', 45.5)
```

In [180]:
```
tup2=("python", 87)

tup3 = tup1 + tup2
print(tup3)
```

```
('parul', 22, 'Bharat', 45.5, 'vadodara', 99, 'python', 87)
```

In [181]:
```
#immutable

del tup3[3]    #del is keyword used for deleting the items.
print(tup3)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_17108\2753664307.py in <module>
      1 #immutable
      2
----> 3 del tup3[3]    #del is keyword used for deleting the items.
      4 print(tup3)

TypeError: 'tuple' object doesn't support item deletion
```

Methods in python tuples

Count Method

Returns the number of times a specified value occurs in a tuple

In [213]:
```
numbers = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)

x = numbers.count(5)  # 5 is the element in tuple

print(x)
```

```
2
```

Index Method

Searches the tuple for a specified value and returns the position of where it was found

In [214]:
```
numbers = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)

x = numbers.index(8) # 8 is the element in tuple

print(x)
```

```
3
```

Python Dictionary

Image source:https://www.freepik.com/

- Dictionaries are used to store data values in key:value pairs.
- Dictionaries are written with curly brackets, and have keys and values:

In [131]:
```
cars = {
  "brand": "Mahindra",
  "model": "Thar",
  "year": 2010
}
```

In [132]:
```
print(cars)
print(type(cars))
```

```
{'brand': 'Mahindra', 'model': 'Thar', 'year': 2010}
<class 'dict'>
```

In [133]:
```
print(cars["model"]) #Accessing Items
```

```
Thar
```

Methods in python Dictionaries

Accessing Method

In [134]:
```
#print only key values
print(cars.keys())      # To access all keys
```

```
dict_keys(['brand', 'model', 'year'])
```

In [135]:
```
#print only values
print(cars.values())   # To access all values
```

```
dict_values(['Mahindra', 'Thar', 2010])
```

In [136]:
```
cars.items()          # To access all keys,values
```

Out[136]:
```
dict_items([('brand', 'Mahindra'), ('model', 'Thar'), ('year', 2010)])
```

Update Method

The update() method will update the dictionary with the items from the given argument.

In [137]:
```
cars.update({"year": 2012})
cars
```

Out[137]:
```
{'brand': 'Mahindra', 'model': 'Thar', 'year': 2012}
```

Adding Items

In [138]:
```
cars["color"] = "black"  # To add key-value pair
print(cars)
```

```
{'brand': 'Mahindra', 'model': 'Thar', 'year': 2012, 'color': 'black'}
```

Removing Items`

In [139]:
```
cars.pop("model")
cars
```

Out[139]:
```
{'brand': 'Mahindra', 'year': 2012, 'color': 'black'}
```

Python Sets

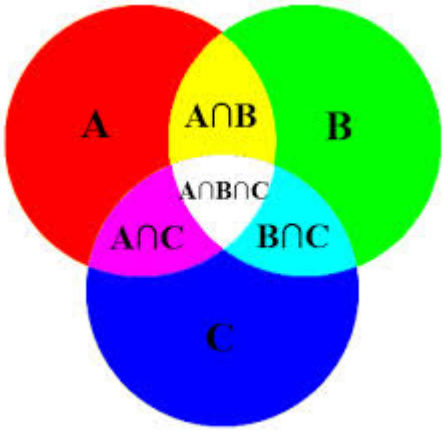


Image source:https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcTeXIDecxbviB9o8zGiTs3NxKTXc798jKIqng&usqp=CAU

The simple funcationality of set is it just ignores the duplicates in the set. Sets are:

1. Unoredered
2. Unchangeable
3. Not allowed duplicate values

In [184]:
```
subjects1 = {'Physics', 'Hindi', 'Chemistry', 'maths', 'Hindi'}
print(subjects1)
print(type(subjects))
```

```
{'maths', 'Hindi', 'Chemistry', 'Physics'}
<class 'set'>
```

In [185]:
```
subjects2 = {'History', 'Social Science', 'Biology','Hindi'}
print(subjects2)
```

```
{'Social Science', 'History', 'Hindi', 'Biology'}
```

In [186]:
```
subjects1|subjects2  # This pipe symbol used to make union of two sets
```

Out[186]:
```
{'Biology',
 'Chemistry',
 'Hindi',
 'History',
 'Physics',
 'Social Science',
 'maths'}
```

In [187]:
```
subjects1.union(subjects2) # This is another method to make union of tw
```

Out[187]:
```
{'Biology',
 'Chemistry',
 'Hindi',
 'History',
 'Physics',
 'Social Science',
 'maths'}
```

In [188]:
```
subjects1.intersection(subjects2)  # To find the common iterm in both t
```

Out[188]:
```
{'Hindi'}
```

Python Built in Functions in data types

Len Function

returns the numbers of items in datatypes

In [220]:
```
fruits = ['apple', 'banana', 'cherry']
len(fruits)
```

Out[220]: 3

In [229]:
```
name = 'Shivaji Maharaj'  #Counts the spaces as well
len(name)
```

Out[229]: 15

Sorted Function

Returns a sorted elements

In [226]:
```
theater = {
  "Movie": "Pushpa",
  "Actor": "Allu Arjun",
  "year": 2021
}
sorted(theater)
```

Out[226]:
```
['Actor', 'Movie', 'year']
```

Max Function

Returns a maximum elements

In [227]:
```
numbers = [1, 3, 7, 8, 7, 5, 4, 6, 8, 5]
max(numbers)
```

Out[227]: 8

Sum Function

Returns a summation of elements

In [245]:
```
numbers = [1, 3, 7, 8, 7, 5, 4, 6, 8, 5]
sum(numbers)
```

Out[245]: 54

Dir Function

Returns a methods in datatypes

In [237]:
```
dir(list)
```

Out[237]:
```
['__add__',
 '__class__',
 '__class_getitem__',
 '__contains__',
 '__delattr__',
 '__delitem__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__gt__',
 '__hash__',
 '__iadd__',
 '__imul__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__mul__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__reversed__',
 '__rmul__',
 '__setattr__',
 '__setitem__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'append',
 'clear',
 'copy',
 'count',
 'extend',
 'index',
 'insert',
 'pop',
 'remove',
 'reverse',
 'sort']
```

In [239]:
```
dir(str)
```

Out[239]:
```
['__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',
 'encode',
 'endswith',
 'expandtabs',
 'find',
 'format',
 'format_map',
 'index',
 'isalnum',
 'isalpha',
 'isascii',
 'isdecimal',
 'isdigit',
 'isidentifier',
 'islower',
 'isnumeric',
 'isprintable',
 'isspace',
 'istitle',
 'isupper',
 'join',
 'ljust',
 'lower',
 'lstrip',
 'maketrans',
```

```
 'partition',
 'removeprefix',
 'removesuffix',
 'replace',
 'rfind',
 'rindex',
 'rjust',
 'rpartition',
 'rsplit',
 'rstrip',
 'split',
 'splitlines',
 'startswith',
 'strip',
 'swapcase',
 'title',
 'translate',
 'upper',
 'zfill']
```

Type casting

Image source:https://www.freepik.com/

- In Python, Type Casting is a process in which we convert a literal of one type to another.
- Inbuilt functions int(), float() and str() shall be used for typecasting.

In [252]:
```
n = 10  # Integer
print(type(n))

n = float(n)  # Converted to float
print(type(n))
```

```
<class 'int'>
<class 'float'>
```

In [254]:
```
n = 10 # Integer
n = str(n) # Converted to string
print(type(n))
```

```
<class 'str'>
```

In [257]:
```
fruits = ['apple', 'banana', 'cherry'] # List
print(type(fruits))

fruits = tuple(fruits) # Converted to tuple
print(type(fruits))
```

```
<class 'list'>
<class 'tuple'>
```

Exercise 1

Image source:https://www.freepik.com/

1. Create an empty list called fruits_list.

2. Add "apple", "banana", "orange", "grape", and "mango" to the fruits_list.
3. Print the first, last, and second & third fruits in the list using slicing.
4. Print the number of fruits in the list.
5. Replace the third fruit with "pear" in the list.
6. Check if "apple" is present in the fruits_list and print the result.
7. Remove "banana" and the last fruit from the list.
8. Create another list additional_fruits with "watermelon" and "kiwi", then concatenate it with fruits_list.
9. Sort the fruits_list in alphabetical order.
10. Count the number of times "kiwi" appears in the fruits_list.

Python Functions

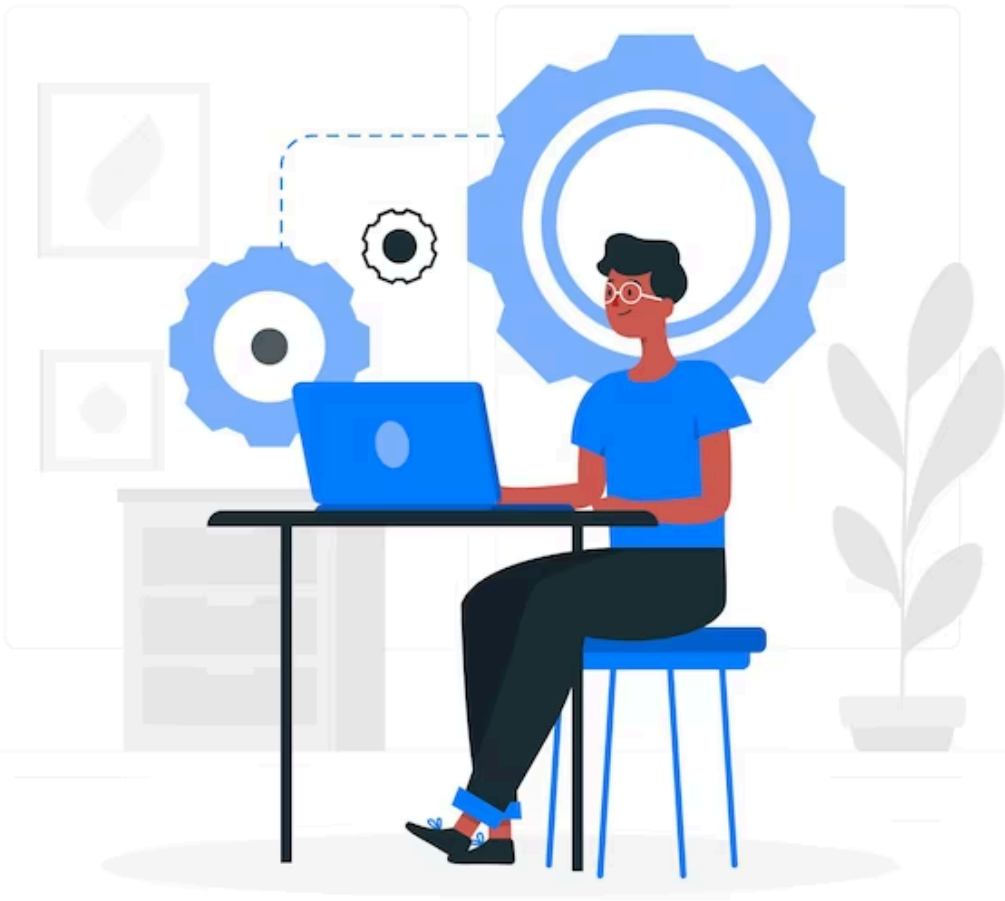


Image source:https://www.freepik.com/

- A function is a small, reusable block of code that takes some inputs, does something with those inputs, and gives you back a result.
- It's like a mini-program that you can use in your big program whenever you want.

Let's break it down with a simple example:

Imagine you have a magical recipe that adds two numbers and gives you the answer. We'll call this recipe "add_numbers". Here's how you would create the recipe (function) in Python:

In [270]:
```
def add_numbers(num1, num2):
    result = num1 + num2
    return result
```

In this recipe, we have:

- def: This keyword tells Python that we are creating a new function.
- add_numbers: The name of our function, you can think of it as the name of our magical recipe.
- num1 and num2: These are placeholders for the ingredients (Parameters) we will put into our recipe.
- : It's like saying, "Here comes the recipe!"
- result = num1 + num2: Inside the recipe, we are adding the ingredients num1 and num2.
- return result: The return keyword means we are giving back the result after doing the magic!
- Now, let's use this magical recipe (function) to add two numbers:

In [271]:
```
# Using our magical recipe "add_numbers"
sum_result = add_numbers(5, 3)
print("The sum is:", sum_result)
```

```
The sum is: 8
```

So, we put the numbers 5 and 3 (Arguments) as ingredients into our magical recipe "add_numbers," and it gave us back the answer 8 as a result!

That's the basic idea of a function in Python. It helps us organize our code, makes it reusable, and saves us a lot of time and effort! Just like how a magical box (function) helps us make cakes (perform a task) with different ingredients (inputs) every time we open it!

Python Conditional Statements

Python if statement

- The If statement is the most fundamental decision-making statement, in which the code is executed based on whether it meets the specified condition.
- It has a code body that only executes if the condition in the if statement is true. The statement can be a single line or a block of code.

In [2]:

```
number = 10

# check if number is greater than 0
if number > 0:
    print('Number is positive.')

print('The if statement is easy')
```

```
Number is positive.
The if statement is easy
```

Python if...else Statement

- This statement is used when both the true and false parts of a given condition are specified to be executed.
- When the condition is true, the statement inside the if block is executed; if the condition is false, the statement outside the if block is executed.

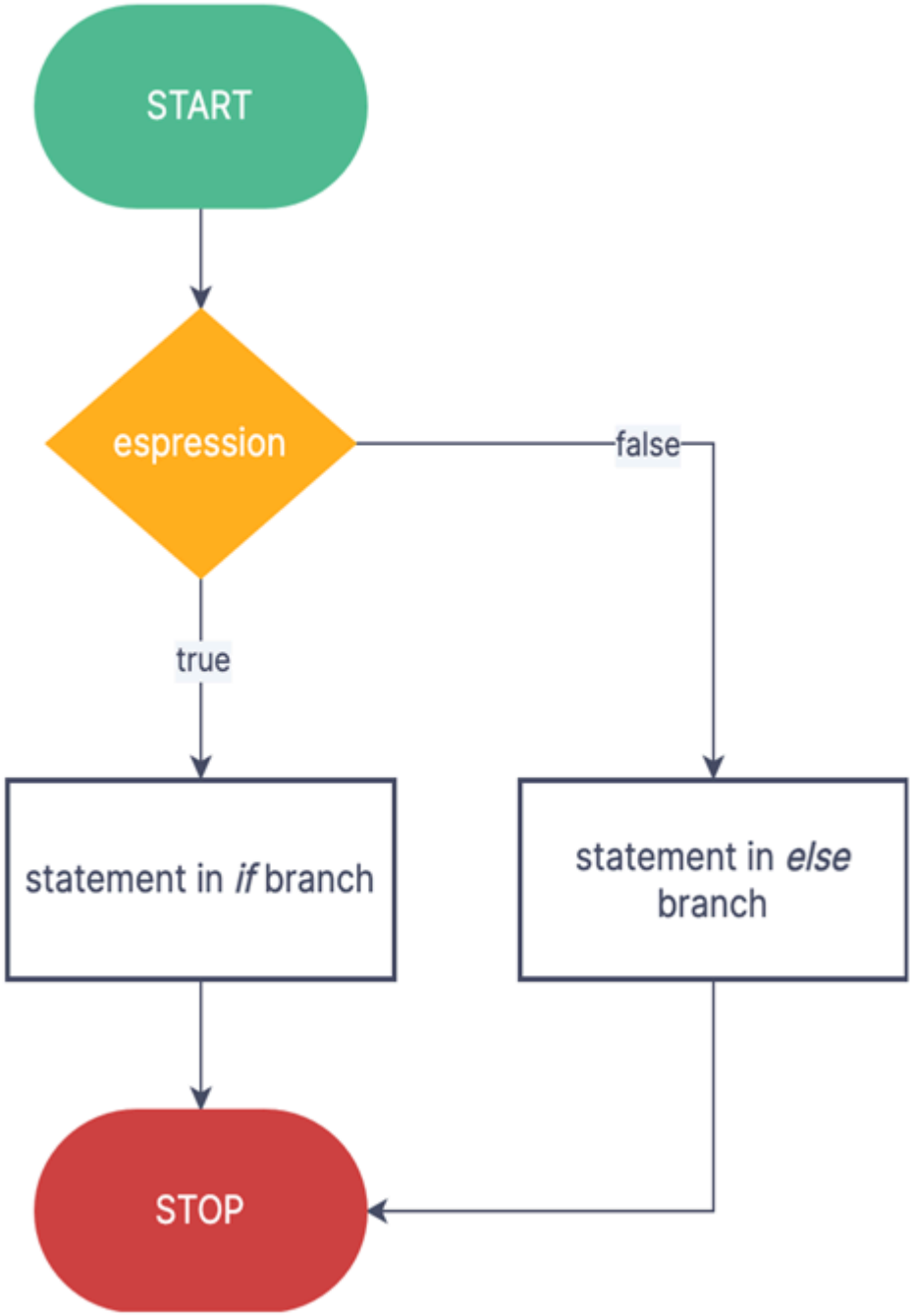


Image source: https://uploads-ssl.webflow.com/6184b461a39ff1011f8c0582/620239fdd8763fe2a551f490_If-else%20Flowchart.png

In [64]:
```
number = 10

if number > 0:
    print('Positive number')

else:
    print('Negative number')

print('This statement is always executed')
```

```
Positive number
This statement is always executed
```

Python if...elif...else Statement

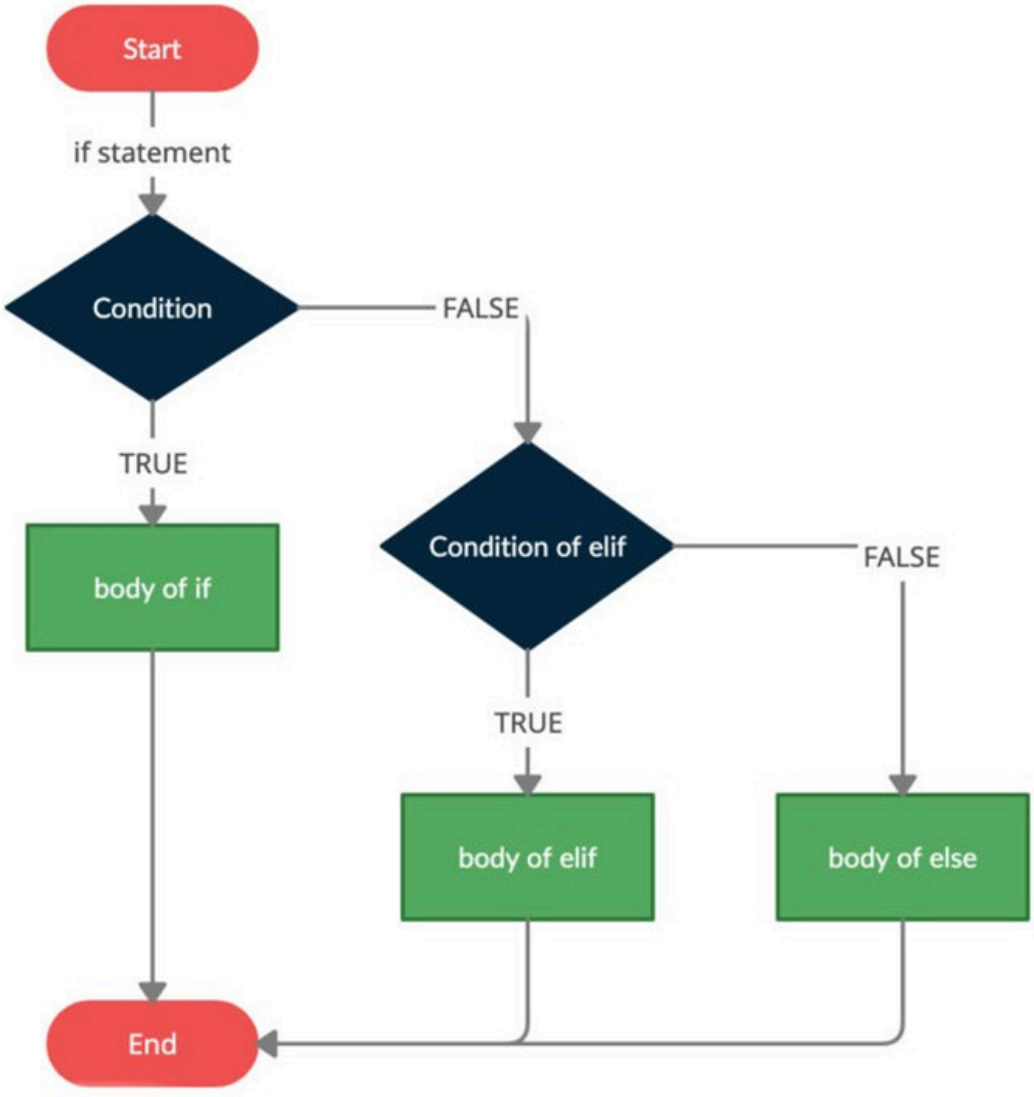


Image source: https://d1whtlypfis84e.cloudfront.net/guides/wp-content/uploads/2021/06/29090322/if-else-flowchart.jpg

- In this case, the If condition is evaluated first.

- If it is false, the Elif statement will be executed; if it also comes false, the Else

In [4]:
```
number = int(input("Enter Number  " ))

if number > 0:
    print("Positive number")

elif number == 0:
    print('Zero')
else:
    print('Negative number')

print('This statement is always executed')
```

```
Enter Number  2
Positive number
This statement is always executed
```

Python Loops

- In general, statements are executed sequentially.
- The first statement in a function is executed first, followed by the second, and so on. There may be a situation when you need to execute a block of code several number of times.

Types of Loops

1. While loop
2. For loop

Let's discuss while loop first.

1. While Loops

A while loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.

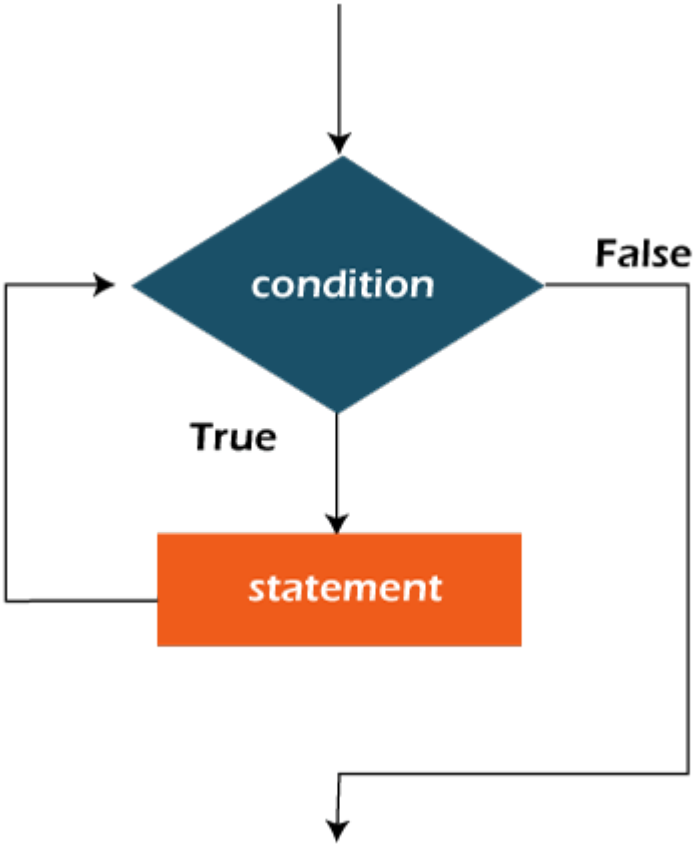


Image source: https://static.javatpoint.com/core/images/java-while-loop.png

In [5]:
```
counter = 0

while counter < 3:
    print('Hello world')
    counter = counter + 1
```

```
Hello world
Hello world
Hello world
```

2. For Loops

- It could iterate over the items of any sequence, such as a list or a string.

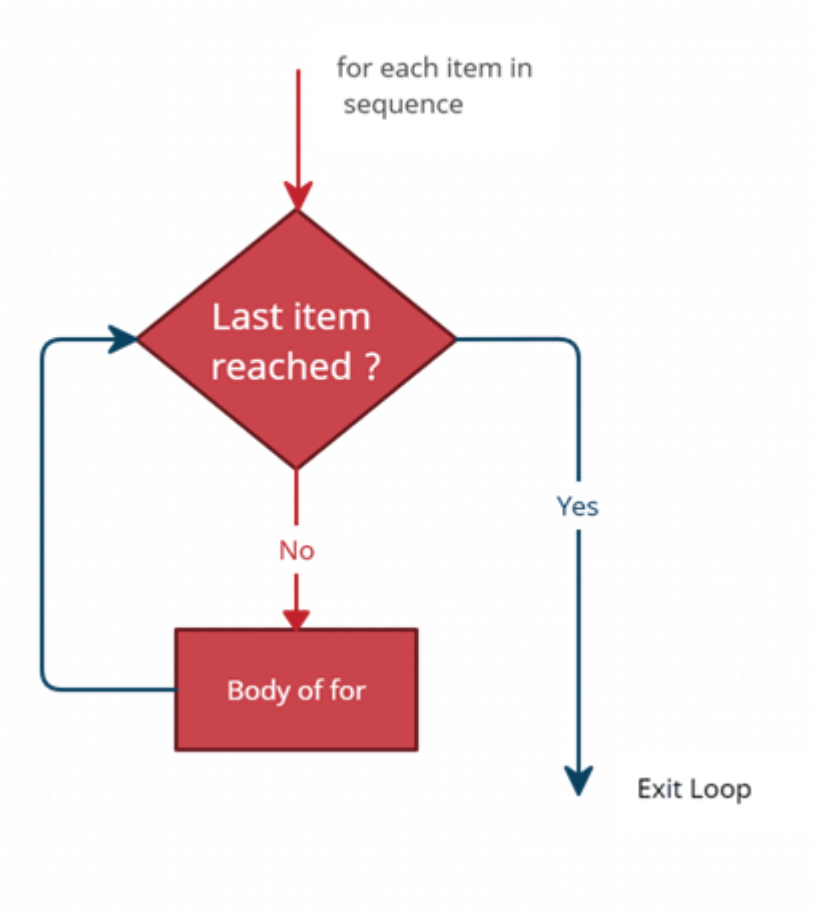


Image source: https://www.programmingfunda.com/wp-content/uploads/2020/10/python-for-loop-flowchart-848x1024.png

In [10]:
```
# use of range() to define a range of values- range(start, stop, step)

for i in range(10):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

In [11]:
```
for letter in "Python":
    print(letter)
```

```
P
y
t
h
o
n
```

In [12]:
```
fruits = ['apple', 'banana', 'cherry']

for fruit in fruits:
    print(fruit)
```

```
apple
banana
cherry
```

Python expression statement

Pass statement

- In Python programming, the pass statement is a null statement which can be used as a placeholder for future code.
- Suppose we have a loop or a function that is not implemented yet, but we want to implement it in the future.

In [13]:
```
n = 10

# use pass inside if statement
if n > 10:
    pass
```

Del statement

- del keyword deletes the element.

In [14]:
```
x = ["physics", "chemistry", "maths"]

del x[0]

print(x)
```

```
['chemistry', 'maths']
```

Break statement

- Break statement is use when we want to break the loop after specific iteration

In [15]:
```python
#Python break Statement
i = 1
while i < 9:
  print(i)
  if i == 3:
    break
  i += 1
```

```
1
2
3
```

Continue statement

- Continue statement is used when we want to skip any specific iteration.

In [16]:
```python
#Python continue Statement
i = 0
while i < 10:
  i += 1
  if i == 3:
    continue
  print(i)
```

```
1
2
4
5
6
7
8
9
10
```

Embark on a Code Adventure: Let's Start Programming!

Question 1: You are at the grocery store and have a list of items and their prices.

- Write a Python function to calculate the total bill amount for a shopping cart containing these items.
- The function should take two lists as input: one containing the item names and another containing their corresponding prices.
- Make sure to handle scenarios where the item prices are not given or the lists are of different lengths.

In [45]:
```python
def calculate_total_bill(items, prices):
    if len(items) != len(prices):
        return "Error: The number of items and prices should be the sam

    total_bill = 0
    for price in prices:
        total_bill += price
    return total_bill

# Example usage:
items_list = ["Apple", "Banana", "Milk", "Bread"]
prices_list = [1.2, 0.6, 2.3, 1.5]
total_bill_amount = calculate_total_bill(items_list, prices_list)
print("Total bill amount:", total_bill_amount)
```

```
Total bill amount: 5.6
```

Question 2: Some items in the grocery store have discounts.

- Write a Python function that calculates the discounted bill amount by reducing the price of items with a discount of 10%.
- The function should take the original price list and a list of items eligible for the discount, and then return the discounted total bill amount.

In [46]:
```python
def calculate_discounted_bill(original_prices, discount_items):
    discounted_bill = 0
    for i in range(len(original_prices)):
        if discount_items[i]:
            discounted_bill += original_prices[i] * 0.9
        else:
            discounted_bill += original_prices[i]
    return discounted_bill

# Example usage:
original_prices_list = [10, 5, 8, 12]
discount_items_list = [True, False, True, False]
discounted_bill_amount = calculate_discounted_bill(original_prices_list
print("Discounted bill amount:", discounted_bill_amount)
```

```
Discounted bill amount: 33.2
```

Exercise 2: Counting Even Numbers

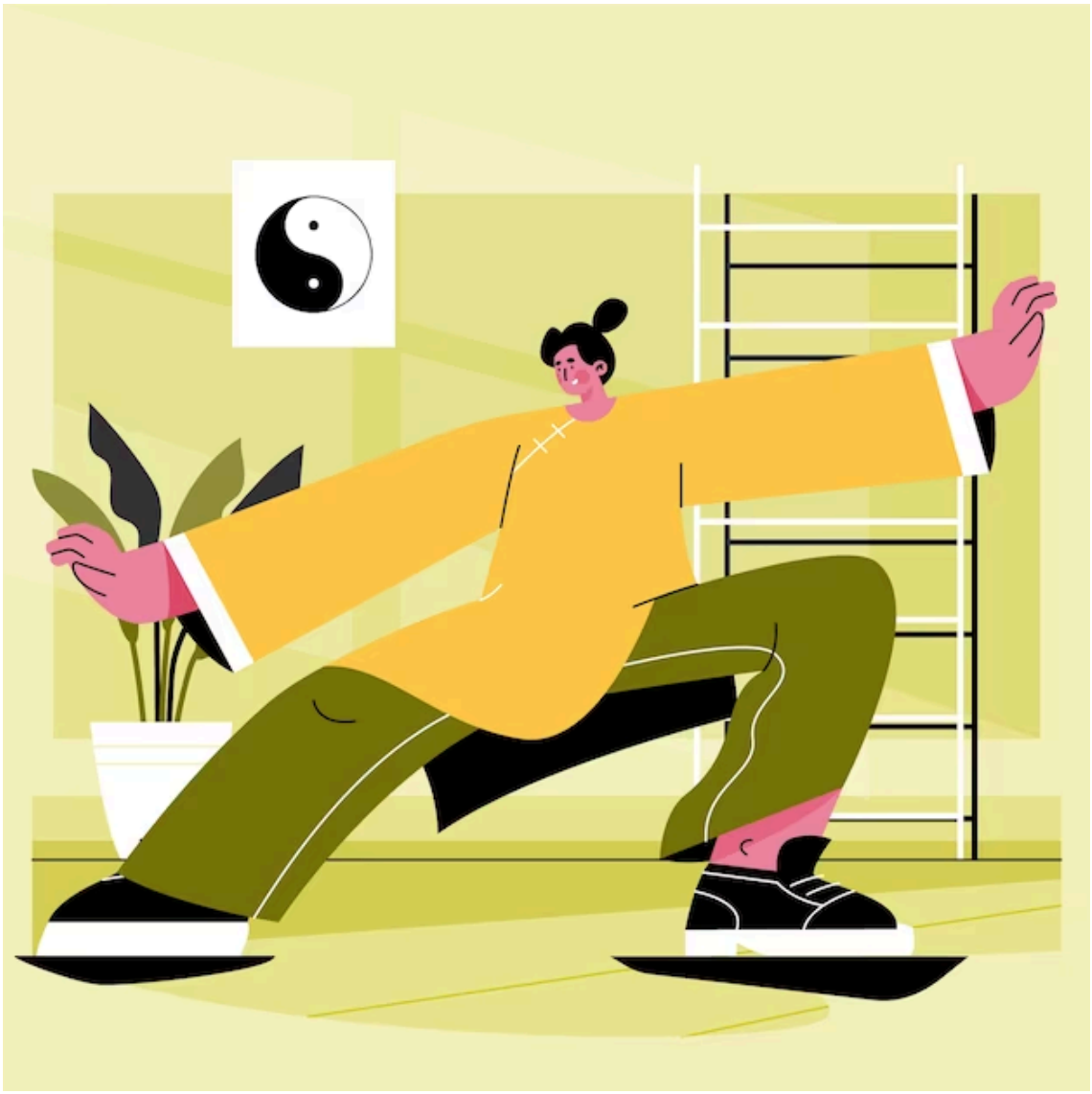


Image source:https://www.freepik.com/

Question: Write a Python function that takes a list of numbers as input and returns the count of even numbers in the list.

Python Lambda Function

- A lambda function is a small anonymous function. A lambda function can take any number of arguments, but can only have one expression. It's mostly used in pandas to maniputate the data

Syntax

- lambda arguments : expression

In [47]:
```
x = lambda a: a + 10
print(x(5))
```

```
15
```

Here 'a' is the defined variable and after colon(:), write what to return. Now you are ready to call it just like fuction.

In [48]:
```
x = lambda a, b : a * b
print(x(5, 6))
```

```
30
```

Exception Handling in python

- Exception handling in Python is a crucial concept that allows programmers to gracefully handle errors and unexpected situations in their code.
- It ensures that the program does not crash abruptly but instead provides useful information about what went wrong.
- Here are three fundamental exception handling methods for beginners:

1. try-except block

- The try-except block is used to catch exceptions and handle them gracefully.
- Code that might raise an exception is placed within the try block, and if an exception occurs, it is caught and processed in the except block.

Example:

In [51]:
```
try:
    numerator = int(input("Enter the numerator: "))
    denominator = int(input("Enter the denominator: "))
    result = numerator / denominator
    print("Result:", result)
except ZeroDivisionError:
    print("Error: Cannot divide by zero.")
except ValueError:
    print("Error: Please enter valid integers for numerator and denomin
```

```
Enter the numerator: 10
Enter the denominator: 10.1
Error: Please enter valid integers for numerator and denominator.
```

2. try-except-else block

- The try-except-else block is an extension of the try-except block.
- The else block is executed if no exceptions occur in the try block.
- It is useful when you want to execute some code only if no exceptions were raised.

Example:

In [53]:
```
try:
    number = int(input("Enter a number: "))
except ValueError:
    print("Error: Please enter a valid integer.")
else:
    square = number ** 2
    print("Square of the number:", square)
```

```
Enter a number: 10
Square of the number: 100
```

3. try-except-finally block

- The try-except-finally block ensures that the code inside the finally block is executed, regardless of whether an exception occurred or not.
- It is typically used for clean-up operations or releasing resources.

Example:

In [54]:
```
try:
    file = open("data.txt", "r")
    content = file.read()
    print(content)
except FileNotFoundError:
    print("Error: File not found.")
finally:
    if 'file' in locals():
        file.close()
```

```
Error: File not found.
```