Python

# Basics

## Formatted strings

Using f at the beginning and {} for the variables to consider as string, which is the new features in python 3.0

|  |
| --- |
| name = input('Enter your name: ');  age = 29;  print(f'Hi {name} your age is {age}'); |

Previously in python 2.0, we use + and str(convert to string)

|  |
| --- |
| name = input('Enter your name: ');  age = 29;  print('Hi ' + name + ' your age is ' + str(age) ); |

Also using format in python 2.0

|  |
| --- |
| name = input('Enter your name: ');  age = 29;  print('Hi {} your age is {}'.format(name,age)); |

As format is little bit complicate using the first one using f in the front of the string.

## String Indexes and slicing

Using [] get the value of the index starting from 0, it will print c

|  |
| --- |
| name = 'Chiranjit';  print(name[0]); |

And also we can give start:stop indexes, it will print **ch** starting from 0 and ending at 1

|  |
| --- |
| name = 'Chiranjit';  print(name[0:2]); |

Also we can use stepover [start:stop:stepover], it will print **Ciajt** as it stepover every 2 character

|  |
| --- |
| name = 'Chiranjit';  print(name[0:9:2]); |

Also [start:], if we not specify stop indexes it will print till the end of the string **Chiranji**t

|  |
| --- |
| name = 'Chiranjit';  print(name[0:]); |

Also [-] negative index is to start from the end, it will print **t**

|  |
| --- |
| name = 'Chiranjit';  print(name[-1]); |

**Now the reversing the string using stepover as negative, it will reverse the string**

|  |
| --- |
| name = 'Chiranjit';  print(name[::-1]); |

## Immutability

Strings in python is immutable, that means the value cannot be changed once assigned. Only way to change is to completely reassign the value.

Below will give error, as we are changing one of the indexes instead of full change

|  |
| --- |
| name = 'Chiranjit';  name[0] = 'V';  print(name); |

Also Tuple is immutable

## Built-in Functions and Methods

* In built-in methods we use methodname(). E.g.:- len(‘chiranjit’)
* Whereas Methods can be used as .methodname. E.g:-

|  |
| --- |
| txt = "hello, and welcome to my world."  x = txt.capitalize()  print (x) |

## Lists

List is a order sequence of object that can be of any type. It is nothing but array in other programming language.

|  |
| --- |
| li = [1,2,3,4]  li2 = [1,2,'a','b',True] |

### List slicing

listName[indexOfStartitem: indexOfEnditem:StepOver]

It is print [‘Toys’,’Books’]

|  |
| --- |
| amazon\_cart = ['Toys',  'Books',  'Grocery',  'Mobiles']  print(amazon\_cart[0:2]) |

It will print [‘Toys’,’Grocery’] as we use stepover

|  |
| --- |
| amazon\_cart = ['Toys',  'Books',  'Grocery',  'Mobiles']  print(amazon\_cart[0::2]) |

Lists are immutable means, we can replace a element with anything.

|  |
| --- |
| amazon\_cart = ['Toys',  'Books',  'Grocery',  'Mobiles']  amazon\_cart[0] = 'Cloths'  print(amazon\_cart) |

### Copying vs modifying list from another list

|  |
| --- |
| amazon\_cart = ['Toys',  'Books',  'Grocery',  'Mobiles']  amazon\_cart[0] = 'Cloths'  #Here below we just copying amazon\_cart to newcart  new\_cart = amazon\_cart[:];  #Here we assign the amazon\_cart to new\_cart2, means if the new\_cart2 changes then amazon\_cart  # will alsi change  new\_cart2 = amazon\_cart;  new\_cart[0] ='Gum';  print(new\_cart)  print(amazon\_cart)  new\_cart2[0] ='Shirts'  #Below both new\_cart2 and amazon\_cart will print the same value, as new\_cart2 is updated,  #amazon\_cart will also get updated, as they allocated same memory  print(new\_cart2);  print(amazon\_cart); |

### Matrix

List inside another list

|  |
| --- |
| matrix = [  [1,2,3],  [4,5,6],  [7,8,9]  ]  print(matrix[0][2]); |

### Reverse list using list slicing

Also remember it creates a new version.

|  |
| --- |
| list1 = ['a','z','b','y','d','e'];  print(list1[::-1]); |

### List Unpacking

|  |
| --- |
| a,b,c = [1,2,3];  print(a);  print(b);  print(c); |

Also placing others remaining element in another list

|  |
| --- |
| a,b,c, \*other = [1,2,3,4,5,6,7,8,9];  print(a);  print(b);  print(c);  print(other); |

### List Methods

<https://www.w3schools.com/python/python_ref_list.asp>

#### append(value)

Append basically adding new item at the end of the list

|  |
| --- |
| list1 = [1,2,3,4,5];  list2 = list1.append(7);  print(list1);  ## list2 will print none as list1 it is just modifying the list1 not copying to list2  print(list2);  list3 = list1;  ## list3 will print the same as list1 as we copying without appending in the same line  print(list3); |

#### insert(index, value)

Insert basically used to add new item wherever we want based on the index.

|  |
| --- |
| list1 = [1,2,3,4,5];  list1.insert(2,9)  print(list1); |

#### extend(list)

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

|  |
| --- |
| list1 = [1,2,3,4,5];  list1.extend([7,8,9])  print(list1); |

#### pop()

It remove the last element of the list and also we can give index in the pop(index)

|  |
| --- |
| list1 = [1,2,3,4,5];  list1.pop();  print(list1); |

#### remove(value)

Its help remove the element based on the value.

|  |
| --- |
| list1 = [1,2,3,4,7];  list1.remove(7);  print(list1); |

#### clear()

It just clears all the elements from the list.

|  |
| --- |
| list1 = [1,2,3,4,7];  list1.clear();  print(list1); |

#### sort(), reverse()

|  |
| --- |
| list1 = ['a','z','b','y','d','e'];  list1.sort();  list1.reverse();  print(list1); |

#### range()

It helps to create a list

Below will create a list of 0-99 numbers

|  |
| --- |
| print(list(range(100))); |

#### join()

It helps to join multiple string in one sentence.

|  |
| --- |
| new\_sentence = ' '.join(['Hello' ,'Everyone','How','are', 'you','?']);  print(new\_sentence); |

#### set()

We remove the duplicate from the list.

|  |
| --- |
| list = [1,2,3,3,4,5]  print(set(list)); |

## Dictionary

Dictionary is another datatype or data structure, where we have key and value pair storage. It’s basically an unordered (**means the storage is sequential in memory, but randomly stored**) key value pair.

Recently the python 3.7 and above, dictionary are ordered.

<https://softwaremaniacs.org/blog/2020/02/05/dicts-ordered/en/>

Based on the key, we get the value from the memory.

|  |
| --- |
| dictionary = {  'a': 1,  'b': 2  }  print(dictionary['a']); |

|  |
| --- |
| dictionary = {  'a': [1,2,3],  'b': "Hello",  'x': True  }  print(dictionary['a']); |

Also, we can store dictionary inside a list, as shown below.

|  |
| --- |
| my\_list = [  {  'a': [1,2,3],  'b': "Hello",  'x': True  },  {  'a': [5,6,7],  'b': "Hi",  'x': False  }  ]  print(my\_list[0]['a']); |

### Dictionary Keys

Dictionary keys needs to be immutable, means a key cannot change. Therefore, we can’t use list as a key

Below will give error.

|  |
| --- |
| dictionary = {  'a': [1,2,3],  True: "Hello",  1: "Hello",  [1,2,3]:"abc"  }  print(dictionary[1]) |

**Also, key should be unique**.

### Dictionary Methods

<https://www.w3schools.com/python/python_ref_dictionary.asp>

#### get()

To check whether the key is present or not. If not present it will return none.

It will return none, as age is not present and also we can set default in the second parameter of get

|  |
| --- |
| user = {  'hobbies': 'Trading',  'Job':'Software Engineer',  }  print(user.get('age',’29’)); |

#### dict()

Built in function for creating dictionary.

|  |
| --- |
| user1 = dict(name="Chiranjit");  print(user1); |

#### in

Same as in list, we can check whether the key is present or not in the dictionary. Based on the existence it will return true or false.

|  |
| --- |
| user = {  'hobbies': 'Trading',  'Job':'Software Engineer',  }  print('hobbies' in user); |

#### keys() and values()

To check specifically on keys and values.

|  |
| --- |
| user = {  'hobbies': 'Trading',  'Job':'Software Engineer',  }  print('hobbies' in user.keys());  print('Software Engineer' in user.values()); |

#### pop(), popitem(), clear(), copy(), update()

## Tuples

Tuples are like list, but they are immutable (means once created, it can’t change, sort etc.

Below will give, for modify the value as we known that tuples are immutable in nature.

|  |
| --- |
| my\_tuble = ('a','b','c','d');  my\_tuble[1] = 's' |

Use tuples when we don’t need to change anything once assign.

Assigning values

|  |
| --- |
| x,y,z, \*other= ('a','b','c','d','e',1);  print(other); |

### Tuples Methods

<https://www.w3schools.com/python/python_ref_tuple.asp>

#### count(value)

Get the count of a value from the tuple.

|  |
| --- |
| tuple = ('a','b','c','d','e',1,'b');  print(tuple.count('b')); |

#### index(value)

Get the first index of a value from the tuple.

|  |
| --- |
| tuple = ('a','b','c','d','e',1,'b');  print(tuple.index('b')); |

## Sets

Sets are unordered collection of unique objects

Below it will remove the duplicate value as we know sets are unique objects

|  |
| --- |
| my\_set = {1,2,3,3,4,5}  print(my\_set); |

**How to check any item is present or not in the sets.?**

Using in

|  |
| --- |
| my\_set = {1,2,3,3,4,5}  print(1 in my\_set); |

### Sets Methods

<https://www.w3schools.com/python/python_ref_set.asp>

#### difference()

Return a set that contains the items that only exist in set x(my\_set), and not in set y(my\_set2):

|  |
| --- |
| my\_set = {1,2,3,3,4,5}  my\_set2 = {6,4,2,7,8,9}  print(my\_set.difference(my\_set2)); |

#### discard()

The discard() method removes the specified item from the set.

|  |
| --- |
| my\_set = {1,2,3,3,4,5}  my\_set.discard(5)  print(my\_set); |

#### difference\_update()

The difference\_update() method removes the items that exist in both sets.

|  |
| --- |
| my\_set = {1,2,3,3,4,5}  my\_set2 = {6,4,2,7,8,9}  my\_set.difference\_update(my\_set2);  print(my\_set); |

#### intersection()

The intersection() method returns a set that contains the similarity between two or more sets.

|  |
| --- |
| my\_set = {1,2,3,3,4,5}  my\_set2 = {6,4,2,7,8,9}  print(my\_set.intersection(my\_set2)); |

Shortcut of intersection using &

|  |
| --- |
| my\_set = {1,2,3,3,4,5}  my\_set2 = {6,4,2,7,8,9}  print(my\_set & my\_set2); |

#### isdisjoint

The isdisjoint() method returns True if none of the items are present in both sets, otherwise it returns False.

|  |
| --- |
| my\_set = {1,2,3,3,4,5}  my\_set2 = {6,4,2,7,8,9}  print(my\_set.isdisjoint(my\_set2)); |

#### union()

The union() method returns a set that contains all items from the original set, and all items from the specified sets.

|  |
| --- |
| my\_set = {1,2,3,3,4,5}  my\_set2 = {6,4,2,7,8,9}  print(my\_set.union(my\_set2)); |

**Shortcut of union using |**

|  |
| --- |
| my\_set = {1,2,3,3,4,5}  my\_set2 = {6,4,2,7,8,9}  print(my\_set | my\_set2); |

#### issubset()

The issubset() method returns True if all items in the set exists in the specified set, otherwise it retuns False.

|  |
| --- |
| my\_set = {4,2}  my\_set2 = {6,4,2,7,8,9}  print(my\_set.issubset(my\_set2)); |

#### issuperset()

The issuperset() method returns True if all items in the specified set exists in the original set, otherwise it retuns False.

|  |
| --- |
| my\_set = {4,2}  my\_set2 = {6,4,2,7,8,9}  print(my\_set2.issuperset(my\_set)); |

## Conditional Logic

If , elif, else

Remember one thing there is no {} for the conditional statement, in python it decided based on the indentation after the conditional statement.

|  |
| --- |
| age =int(input("Enter your age: "));  if age <= 18:  print("Your are not allowed to drive");  elif age > 18 and age < 60:  print("Hurray...Lets drive..")  else:  print("Sorry... your are not allowed..to drive");  print('Hello'); |

## Truthy and Falsey

<https://stackoverflow.com/questions/39983695/what-is-truthy-and-falsy-how-is-it-different-from-true-and-false>

|  |
| --- |
| is\_old = bool("Hello");  is\_licened = bool(10);  #It is print true means truthy  print(is\_old);  print(is\_licened);  is\_old = bool("");  is\_licened = bool(0);  #It is print False means falsy  print(is\_old);  print(is\_licened); |

## Ternary Operator

Shortcut of conditional statement

|  |
| --- |
| is\_Friend = True;  can\_message = "Message Allowed" if is\_Friend else "You are not allowed for messaging";  print(can\_message); |

## Logical Operator

and, or, >, < , >=, <=, == , !=, not()

not(True) = False, means not used just to opposite

## is vs ==

**==** checks for equality of values

**is** checks the location of memory where the value is stored

|  |
| --- |
| print(True == 1) # true  print('' == 1) # false (as '' is changed to false and 1 changed to True)  print([] == 1) # false  print(10 == 10.0) # True  print([] == []) # True |

|  |
| --- |
| print(True is True) #True  print([] is []) # false because both the stored in different location of memory |

## For loops

|  |
| --- |
| for item in 'Chiranjit Saha':  print(item);  for item in {1,2,3,'Chiranjit'}:  print(item);  for item in (1,2,3,'Chiranjit'):  print(item); |

Nested loops

|  |
| --- |
| for item in [1,2,3,4,5]:  for item2 in {'a','b','c','d','e'}:  print(item, item); |

**Iterables - can be String, list, dictionary, tuple, set. Because they are iterated means we can check one by one each item in the collection**.

**Looping in dictionary**

|  |
| --- |
| user = {'name': 'Chiranjit', 'age': 28, 'job': 'Software Engineer'}  for key, value in user.items():  print(key, value)  for item in user.values():  print(item)  for item in user.keys():  print(item) |

## range

Range() creates a special type of object which helps in looping/iterating.

**range(start , end , stepover)**

|  |
| --- |
| for item in range(0,20):  print(item); |

And remember if you don’t want the item variable, we can put underscore (**\_**) , it just a indicator for the other team members

|  |
| --- |
| for \_ in range(0,20):  print('I am changing my life'); |

**Reversing the iteration.**

|  |
| --- |
| for item in range(10,0,-1):  print(item); |

**Also, we can use range in list creation.**

**Below will create a list from 0 to 9**

|  |
| --- |
| print(list(range(10))); |

## enumerate()

Enumerate() takes the Iterables objects and gives us the index counter.

It is very useful if we need the index counter for the item we loop through.

|  |
| --- |
| for index, item in enumerate('Chiranjit'):  print(index,item); |

## While loop

|  |
| --- |
| i = 0;  while i < 50:  print(i);  i = i +1;  else:  print('Done with the loop'); |

Iterating list using while loop

|  |
| --- |
| my\_list = [1,2,3,4];  i = 0;  while i < len(my\_list):  print(my\_list[i]);  i += 1; |

While loop is extremely help where we don’t know, when the looping will end.

|  |
| --- |
| while True:  response = input("Say Something: ");  if(response.upper() == 'BYE'):  break;    print('Thank You for the conversion'); |

## break, continue, pass

break: - is used to break out of the current execution or loop.

continue: - is used to continue the execution.

pass: - is rarely used in python code.

## Developer Fundamental

How to write a good code?

1. Clean.
2. Readability.
3. Predictability.
4. DRY (Don’t Repeat yourself). Reusable.

## Functions

We create function, if we want some functionality is repeating means we can reuse the code, without writing the same functionality multiple times.

|  |
| --- |
| def Hello():  print("Hello");  Hello();  Hello(); |

### Function parameters and arguments

Arguments:- are used to pass actual value to a function.

Parameters:- name of the variable that we receive for function

|  |
| --- |
| #name is parameters  def Hello(name):  print(f"Hello {name}");  #'Chiranjit' is arguments  Hello('Chiranjit');  Hello('Arijit'); |

### Keyword Arguments

Keyword Arguments :- When we explicitly specify the arguments with the parameter name.

But remember keyword arguments is a bad practice.

|  |
| --- |
| #name is parameters  def Hello(name, welcomeMsg):  print(f"Hello {name} {welcomeMsg}");  return  #'Chiranjit' is arguments  Hello('Arijit', 'What you are doing?');  # keyword arguments  Hello(welcomeMsg = 'How are you?', name ='Chiranjit'); |

### Default Parameters

In the parameter variable, we can specify the default values. The default parameter will use when we don’t pass any arguments will invoking.

|  |
| --- |
| #name is parameters  #Sanjit and Welcome is default parameter values  def Hello(name='Sanjit', welcomeMsg='Welcome!!'):  print(f"Hello {name} {welcomeMsg}");  return  Hello() |

### Return and function inside another function (nested function)

**Return is last statement to execute in the function**

|  |
| --- |
| def sum(num1, num2):  def sum2(num1, num2):  return num1 + num2;  return sum2(num1, num2);  total = sum(10,20);  print(total); |

### Docstrings

Is used for adding any information for the function using ‘’’

|  |
| --- |
| def checkDriverAge(age=0):  '''  Information about the function  '''  if int(age) < 18:     return "Sorry, you are too young to drive this car. Powering off";  elif int(age) > 18:     return "Powering On. Enjoy the ride!";  elif int(age) == 18:     return "Congratulations on your first year of driving. Enjoy the ride!";  print(checkDriverAge(90)); |

### help()

using help, we can find what a function do, basically it will print the information message for that passed function name.

|  |
| --- |
| def checkDriverAge(age=0):  '''  Information about the function  '''  if int(age) < 18:     return "Sorry, you are too young to drive this car. Powering off";  elif int(age) > 18:     return "Powering On. Enjoy the ride!";  elif int(age) == 18:     return "Congratulations on your first year of driving. Enjoy the ride!";  help(checkDriverAge) |

### \*args and \*\*kwargs

\*args:- if we specify \* Infront of the parameter variable, it means it can except as many positional parameters.

|  |
| --- |
| def total(\*values):  return sum(values);  print(total(1,2,3,4,5)); |

\*\*kwargs:- if we specify \*\* Infront of the keyword parameter variable, it means it can except multiple keyword parameters.

|  |
| --- |
| def sum\_total(\*\*keyargs):  total =0;  for item in keyargs.values():  total = total + item;  return total;  print(sum\_total(num1 = 1, num2= 2, num3 = 3, num4 = 4)); |

**Rule of putting parameter in sequence as below:-**

1. Params.
2. \*args
3. Default params.
4. \*\*kwargs

### Scope Rules

|  |
| --- |
| a = 10;  def function():  def function2():  return a;  return function2();  print(function()); # 10 as if any variable value is not assign  # it will check on the parent then  # global  #local scope inside function  #-> parent scope inside parent function  #-> global scope  #-> built in python function.    # Start with local  # Parent global  # global  # built in python functions |

### global keyword

global keyword is used when we want to use any global variable inside a function.

|  |
| --- |
| total = 0;  def sum():  global total  total += 1;  return total;  sum();  sum();  print(sum()); |

### nonlocal keyword

nonlocal keyword is used when we want to any parent function variable inside a child function.

|  |
| --- |
| # Scope - what variables do I have access to?  def outer():  x = "local"  def inner():  nonlocal x  x = "nonlocal"  print("inner:", x)  inner()  print("outer:", x)  outer()  #1 - start with local  #2 - Parent local?  #3 - global  #4 - built in python functions |

## OOP

OOP is a paradigm the way to think of structing the code by using class instead of procedural code.

### Class and Objects

Class is a blueprint and then it can be instantiating to create instance which is known as Objects.

|  |
| --- |
|  |

### Constructor, Attributes, Methods

Attributes is dynamic, means for different instances we can changes the value.

|  |
| --- |
| class PlayerCharacter:  #Constructor method or \_\_init\_\_ method  # it is automatically called when we instance  def \_\_init\_\_(self,name,age):  #attributes  self.name = name;  self.age = age;    def run(self): # methods  print('run');  return 'done';  # Creating the object  player1 = PlayerCharacter('chiranjit','29');  player2 = PlayerCharacter('arijit','14');  print(player1.name);  print(player1.age);  print(player1.run());  print(player2.name);  print(player2.age); |

### Class object attribute

Class object attribute it not dynamic, it is a static means it does not change across instances.

|  |
| --- |
| class PlayerCharacter:  # Class object attribute  memberShip = True;  #Constructor method or \_\_init\_\_ method  # it is automatically called when we institante  def \_\_init\_\_(self,name,age):  if(self.memberShip):  #attributes  self.name = name;  self.age = age;    def run(self): # methods  print(f'My name is {self.name}');  return 'done';  # Creating the object  player1 = PlayerCharacter('chiranjit','29');  player2 = PlayerCharacter('arijit','14');  print(player1.name);  print(player1.age);  print(player1.memberShip);  print(player1.run());  print(player2.name);  print(player2.age); |

### @classmethod and @staticmethod

If a method is specific as @classmethod then we can call the method directly using the classname instead of instating.

If a method is specific as @staticmethod then we can call the method directly using the classname instead of instating. Difference we cannot access the class as a parameter

|  |
| --- |
| class PlayerCharacter:  # Class object attribute  memberShip = True;  #Constructor method or \_\_init\_\_ method  # it is automatically called when we institante  def \_\_init\_\_(self,name,age):  if(self.memberShip):  #attributes  self.name = name;  self.age = age;    def run(self): # methods  print(f'My name is {self.name}');  return 'done';  @classmethod  def adding\_things(cls,num1,num2):  return num1 + num2;  @staticmethod  def adding\_things2(num1,num2):  return num1 + num2;  # Creating the object  player1 = PlayerCharacter('chiranjit','29');  player2 = PlayerCharacter('arijit','14');  print(PlayerCharacter.adding\_things(2,3));  print(PlayerCharacter.adding\_things2(2,3));  print(player1.name);  print(player1.age);  print(player1.adding\_things(1,2));  print(player1.memberShip);  print(player1.run());  print(player2.name);  print(player2.age); |

### Four Pillars of OPP

#### Encapsulation

 It describes the idea of bundling data and methods that work on that data within one unit. This concept is also often used to hide the internal representation, or state, of an object from the outside.

#### Abstraction

Abstraction refers to the concept of hiding the complexities of a system from the users of that system.

#### Public vs Private variables

There is no private and public access specifier in python, we just use single underscore( \_ ) Infront of the variable name to specify private, so that the other developer just aware on that, means we should not modified.

And for the method we use double (\_\_) to specify private, so that other developer should not modified.

|  |
| --- |
| class PlayerCharacter:  def \_\_init\_\_(self,name,age):  if(self.memberShip):  #private variable use \_ , just to aware to the other developer  self.\_name = name;  self.\_age = age;    def run(self): # methods  print(f'My name is {self.name}');  return 'done';    # Creating the object  player1 = PlayerCharacter('chiranjit','29');  player2 = PlayerCharacter('arijit','14'); |

#### Inheritance

Inheritance is a mechanism in which one class acquires the property of another class. For example, a child inherits the traits of his/her parents. With inheritance, we can reuse the fields and methods of the existing class. Hence, inheritance facilitates Reusability and is an important concept of OOPs

|  |
| --- |
| # parent class  class User():  def signIn(self):  print('Logged in');  #Inheritance the child class by passing the [parent class name.  class Wizard(User):  def \_\_init\_\_(self,name,power):  self.name = name;  self.power = power;    def attack(self):  print(f'attacking with power of {self.power}');  class Archer(User):  def \_\_init\_\_(self,name,num\_arrows):  self.name = name;  self.num\_arrows = num\_arrows;    def attack(self):  print(f'attacking with arrows: arrow left {self.num\_arrows}');  wizard1 = Wizard('John',20);  archer1 = Archer('Shelly',30);  wizard1.attack();  archer1.attack(); |

#### isinstance method

Is used to check any instance is a part of that class or not, based on the status it will return True or False.

|  |
| --- |
| wizard1 = Wizard('John',20);  print(isinstance(wizard1,Wizard)); |

#### Polymorphism

Polymorphism is a OOPs concept where one name can have many forms. For example, same method name acts different for different object

|  |
| --- |
| # parent class  class User():  def signIn(self):  print('Logged in');  #Inheritance the child class by passing the [parent class name.  class Wizard(User):  def \_\_init\_\_(self,name,power):  self.name = name;  self.power = power;    def attack(self):  print(f'attacking with power of {self.power}');  class Archer(User):  def \_\_init\_\_(self,name,num\_arrows):  self.name = name;  self.num\_arrows = num\_arrows;    def attack(self):  print(f'attacking with arrows: arrow left {self.num\_arrows}');  wizard1 = Wizard('John',20);  archer1 = Archer('Shelly',30);  # here we are calling the same method name but the functionality different based on the object, example of polymorphism  wizard1.attack();  archer1.attack(); |

#### Exercise

|  |
| --- |
| class Pets():  animals = []  def \_\_init\_\_(self, animals):  self.animals = animals  def walk(self):  for animal in self.animals:  print(animal.walk())  class Cat():  is\_lazy = True  def \_\_init\_\_(self, name, age):  self.name = name  self.age = age  def walk(self):  return f'{self.name} is just walking around'  class Simon(Cat):  def sing(self, sounds):  return f'{sounds}'  class Sally(Cat):  def sing(self, sounds):  return f'{sounds}'  #1 Add nother Cat  class John(Cat):  def sing(self, sounds):  return f'{sounds}'  #2 Create a list of all of the pets (create 3 cat instances from the above)  my\_cats = [Simon('Simon',2),Sally('Sally',3),John('John',4)];  #3 Instantiate the Pet class with all your cats use variable my\_pets  my\_pets = Pets(my\_cats);  #4 Output all of the cats walking using the my\_pets instance  my\_pets.walk(); |

#### super()

Using super() we can call the attribute of a parent class inside child class. Same we can use of the parent method to call also.

|  |
| --- |
| class User():  def \_\_init\_\_(self,email):  self.email = email;  def signIn(self):  print('Logged in');  #Inheritance the user class by passing the class name.  class Wizard(User):  def \_\_init\_\_(self,name,power,email):  #using super() we call the attribute of the parent class  super().\_\_init\_\_(email);  self.name = name;  self.power = power;    def attack(self):  print(f'attacking with power of {self.power}');  wizard1 = Wizard('John',20,'chiranjitsaha9955@gmail.com');  print(wizard1.email);  wizard1.attack(); |

### Introspection

Ability to determine the type of an object at runtime , using dir() we can find the details of that object.

|  |
| --- |
| class User():  def \_\_init\_\_(self,email):  self.email = email;  def signIn(self):  print('Logged in');  #Inheritance the user class by passing the class name.  class Wizard(User):  def \_\_init\_\_(self,name,power,email):  #using super() we call the attribute of the parent class  super().\_\_init\_\_(email);  self.name = name;  self.power = power;    def attack(self):  print(f'attacking with power of {self.power}');  wizard1 = Wizard('John',20,'chiranjitsaha9955@gmail.com');  print(dir(wizard1)); |

### Dunder method (private method)

<https://docs.python.org/3/reference/datamodel.html#special-method-names>

Dunder methods are the methods which creates using double underscore (\_\_) as before and after name.

Generally we should not modify the dunder method (private method) , but we can modify if required.

|  |
| --- |
| class Toy():  def \_\_init\_\_(self, color, age):  self.color = color  self.age = age  self.my\_dict = {  'name':'Yoyo',  'has\_pets': False,  }  def \_\_str\_\_(self):  return f"{self.color}"  def \_\_len\_\_(self):  return 5  def \_\_del\_\_(self):  return "deleted"  def \_\_call\_\_(self):  return('yes??')  def \_\_getitem\_\_(self,i):  return self.my\_dict[i]  action\_figure = Toy('red', 0)  print(action\_figure.\_\_str\_\_())  print(str(action\_figure))  print(len(action\_figure))  print(action\_figure())  print(action\_figure['name']) |

### isssubclass()

Also below we are using list inbuilt class as a parent for the SuperList class.

|  |
| --- |
| class SuperList(list):  def \_\_len\_\_(self):  return 100;  SuperList1 = SuperList();  print(len(SuperList1));  SuperList1.append(10);  print(SuperList1[0]);  print(issubclass(SuperList,list)); |

### Multiple Inheritance

Child class inherits from multiple parent class.

Remember multiple inheritance is very complicate, as it can leads to locking due to that many language don’t support multiple inheritance.

|  |
| --- |
| # parent class  class User():  def signIn(self):  print('Logged in');  #Inheritance the child class by passing the [parent class name.  class Wizard(User):  def \_\_init\_\_(self,name,power):  self.name = name;  self.power = power;    def attack(self):  print(f'attacking with power of {self.power}');  class Archer(User):  def \_\_init\_\_(self,name,num\_arrows):  self.name = name;  self.num\_arrows = num\_arrows;    def checkarrows(self):  print(f'attacking with arrows: arrow left {self.num\_arrows}');  #Multiple inheritance  class HybridBorg(Wizard, Archer):  def \_\_init\_\_(self,name, power, arrows):  #For multiple inheritance, to use the parent attribute as both are different  Archer.\_\_init\_\_(self,name,arrows);  Wizard.\_\_init\_\_(self,name,power);  HybridBorg1 = HybridBorg('Borg',30,20);  print(HybridBorg1.attack());  print(HybridBorg1.checkarrows());  print(HybridBorg1.signIn()); |

#### MRO – Method Resolution order

<http://www.srikanthtechnologies.com/blog/python/mro.aspx>

Method Resolution Order (MRO) is the order in which Python looks for a method in a hierarchy of classes. Especially it plays vital role in the context of multiple inheritance as single method may be found in multiple super classes.

MRO uses the Depth First Search Algorithm.

## Functional Programming

Basically, separation of concern which we can say same as OOP, but they separate data and function, means instead of combining methods and attributes of OOP they separate them in functional programming. Also, we don’t have any concept of Class and Object in functional programming.

But the goal of Functional programming and OOP is same as below: -

1. Clear and under stable.
2. Easy to extend.
3. Easy to maintain.
4. Memory efficient means we are not storing information all over the place.

Main pillar of Functional programming is function should be pure, means there is a separation of data and behavior of a program.

### Pure Function

Pure function should have two rules

1. Same input should always give the same output.
2. It should not produce any side effect to the outside of the function, for example if a function accessing a variable that present outside of a different scope.

Example of pure function:-

|  |
| --- |
| def multiply\_by2(li):  result = [];  for value in li:  result.append(value\*2);  return result;  print(multiply\_by2([1,2,3])); |

Benefits of Pure function: -

1. Less bug.
2. Testing makes easy.

Also, it is impossible to have all function as pure, but wherever can we should apply the pure functions rules

### map()

The map() function executes a specified function for each item in a Iterables. The item is sent to the function as a parameter.

map(*function*, *iterables*)

|  |  |
| --- | --- |
| function | Required. The function to execute for each item |
| iterable | Required. A sequence, collection or an iterator object.  You can send as many iterables as you like, just make sure the function has one parameter for  each iterable. |

|  |
| --- |
| my\_list = [1,2,3]  def multiply\_by2(li):  return li \* 2;  #the map object needs to convert to list  print(list(map(multiply\_by2, my\_list)));  #my\_list value will not change  print(my\_list); |

### filter()

The filter() function returns an iterator were the items are filtered through a function to test if the item is accepted or not.

filter(function, iterable)

|  |
| --- |
| my\_list = [1,2,3]  def check\_odd(li):  return li %2 != 0;  print(list(filter(check\_odd,my\_list)));  #my\_list value will not change  print(my\_list); |

### zip()

The zip() function returns a zip object, which is an iterator of tuples where the first item in each passed iterator is paired together, and then the second item in each passed iterator are paired together etc

zip(iterator1, iterator2, iterator3 ...)

|  |
| --- |
| my\_list = [1,2,3];  your\_list = [4,5,6];  print(list(zip(my\_list,your\_list))); |

### reduce()

We don’t have inbuilt reduce() function in python, but we can import from **functools**

|  |
| --- |
| my\_list = [1,2,3];  from functools import reduce;  def accumulator(acc,item):  return acc + item; # acc will assign based on the return value in each iteration  print(reduce(accumulator,my\_list,0)); |

### Lambda Expression

Lambda Expression are onetime **anonymous functions (means they just use for only once and also no name)**, there is no name attach, that we don’t need to run more than once.

lambda arguments : expression

|  |
| --- |
| from functools import reduce;  my\_list = [1,2,3];  print(list(map(lambda item : item \* 2, my\_list)));  print(list(filter(lambda item : item % 2!= 0, my\_list)));  print(reduce(lambda acc, item : acc + item, my\_list)); |

Sorting using Lambda based on second key

|  |
| --- |
| a = [(0,2),(4,4),(9,9),(10,-1)];  a.sort(key = lambda x:x[1]);  print(a); |

### Comprehensions

Comprehensions in Python provide us with a short and concise way to construct new sequences (such as lists, set, dictionary etc.) using sequences which have been already defined.

**List Comprehensions**

***output\_list = [*output\_exp *for*var *in*input\_list *if (*var *satisfies this condition)]***

|  |
| --- |
| my\_list = [item for item in 'Hello'];  my\_list2 = [item for item in range(0,100)];  my\_list3 = [item \*\* 2 for item in range(0,100)];  my\_list4 = [item for item in my\_list3 if item % 2 == 0 ];  print(my\_list);  print(my\_list2);  print(my\_list3);  print(my\_list4); |

**Set Comprehensions**

Set Comprehension is same a list comprehension just changes the [] to {}

***output\_set = {*output\_exp *for*var *in*input\_set *if (*var *satisfies this condition)}***

|  |
| --- |
| my\_set = {item for item in 'Hello'};  my\_set2 = {item for item in range(0,100)};  my\_set3 = {item \*\* 2 for item in range(0,100)};  my\_set4 = {item for item in my\_set3 if item % 2 == 0 };  print(my\_set);  print(my\_set2);  print(my\_set3);  print(my\_set4); |

**Dictionary Comprehensions**

***output\_dict = {key:value for (key, value) in*iterable *if (key, value satisfy this condition)}***

|  |
| --- |
| my\_simpledict = {  'a':1,  'b':2  };  my\_dict = {key:value\*\*2 for key,value in my\_simpledict.items() if value % 2 == 0 };  my\_dict2 = {key:key\*2 for key in [1,2,3] };  print(my\_dict);  print(my\_dict2); |

***Exercise :- find the duplicates elements from a list***

|  |
| --- |
| some\_list = ['a', 'b', 'c', 'b', 'd', 'm', 'n', 'n']  # solution using loop  duplicates = []  for value in some\_list:  if some\_list.count(value) > 1:  if value not in duplicates:  duplicates.append(value)  print(duplicates);  #Solution using comprehensions  # Here we use set as we know set removes duplicates character for the list  my\_list =list(set([ item for item in some\_list if some\_list.count(item) > 1]));  print(my\_list); |

### Higher Order function

If a function accepts another function as a parameter or another way is if a function returns another function.

Also map, filter etc. is the example of inbuilt higher order function

|  |
| --- |
| #1st way  def greet(func):  func();  #2nd way  def greet2():  def func():  return 'Hello';  return func; |

## Decorators

In simple term decorators add extra features to a function. Also in the programming the decorators follow the decorator pattern principle for creating any decorator.

|  |
| --- |
| #Creating the custom decorator  # Here func is the function whoever will use this decorator.  def my\_decorators(func):  def wrap\_fun():  print('\*\*\*\*\*\*\*\*\*\*');  func();  print('\*\*\*\*\*\*\*\*\*\*');  return wrap\_fun;  #Using the custom decorator  @my\_decorators  def Hello():  print('Hellooo');  Hello();  #Briefly we can say that the above code can be respected as below, without using decorator: -  a = my\_decorators(Hello);  #or  my\_decorators(Hello)();  print(a()); |

**Passing parameter from the function to decorator.**

|  |
| --- |
| #Creating the custom decorator  # Here func is the function whoever will use this decorator.  def my\_decorators(func):  #passing parameters from the function whoever will use  def wrap\_fun(x,y):  print('\*\*\*\*\*\*\*\*\*\*');  func(x,y);  print('\*\*\*\*\*\*\*\*\*\*');  return wrap\_fun;  #Using the custom decorator  @my\_decorators  #passing parameters  def Hello(greeting, emoji):  print(greeting,emoji);  Hello('Hello',':)'); |

Example of Decorator like performance check, logging, etc.

|  |
| --- |
| from time import time;  def performance\_check(fnc):  def wrapper\_func(\*args):  t1 = time();  result = fnc(\*args);  t2 = time();  print(f'time taken {t2 - t1} ms');  return result;  return wrapper\_func;  @performance\_check  def long\_time():  for i in range(500000):  i\*5;  long\_time(); |

## Error Handling

#### Types of Errors: -

<https://docs.python.org/3/library/exceptions.html>

#### try, except, else and finally block

finally :- runs regardless at the end of everything.

|  |
| --- |
| while True:  try:  age = int(input("Enter your age: "));  10/age;  except ValueError:  print('Please enter a number');  except ZeroDivisionError:  print('Please enter a age higher than zero');  else:  print('Thank You!!!');  break;  finally:  print('Ok I am finally done'); |

#### Handling based on error type and multiple except block

|  |
| --- |
| while True:  try:  age = int(input("Enter your age: "));  10/age;  except ValueError:  print('Please enter a number');  except ZeroDivisionError:  print('Please enter a age higher than zero');  else:  print('Thank You!!!');  break; |

#### How to print the exception message and catch multiple error in same catch block

|  |
| --- |
| def sum(num1, num2):  try:  return num1/num2;  except (TypeError, ZeroDivisionError) as ex:  print(f'Expection occured {ex}');  print(sum('1' , 2));  print(sum(1 , 0)); |

#### How to raise an error instead of catch block

|  |
| --- |
| while True:  try:  age = int(input("Enter your age: "));  10/age;  raise ValueError('some thing wrong');  finally:  print('Ok I am finally done'); |

## Generators

It allows us to generate a sequence of values overtime.

Generators are very easy to implement, but a bit difficult to understand.

Generators are used to create iterators, but with a different approach. Generators are simple functions which return an Iterables set of items, one at a time, in a special way.

When an iteration over a set of item starts using the for statement, the generator is run. Once the generator's function code reaches a "yield" statement, the generator yields its execution back to the for loop, returning a new value from the set. The generator function can generate as many values (possibly infinite) as it wants, yielding each one in its turn.

## Modules

In general, we will not have only python file, instead will have any multiple python files, each python files can be mentioned as one module.

Communication between two modules is

**Import modulename;**

Different ways to import module: -

1. **Import <modulename>;**
2. **from <modulename> import <functionname1fromthatmodule>,<functionname2> etc..;**

|  |
| --- |
| from utility import multiply,divide; from shopping.shopping\_cart import buy;  print(buy('mango')); print(divide(12,8)); print(multiply(12,8)); |

* **from <modulename> import \*;**

Here \* represents import everything from that module.

#### Packages

Packages is python mentioned as the folder containing the modules. So we import it as below in the other module outside that package.

Import <packagename/foldernam>.<modulename>

#### How to find the main python file?

|  |
| --- |
| if \_\_name\_\_ == "\_\_main\_\_":  print("hello"); |

Basically “**\_\_name\_\_** “returns the name of the module currently executing. And “**\_\_main\_\_**” return for the main module (or from where the code will start the execution).

#### Built-in Modules

Whenever we installed python, we get access to all the built in module.

<https://docs.python.org/3/py-modindex.html>

Example :-

Here **dir()** is used to check all the method available for random module.

|  |
| --- |
| import random;  print(dir(random));  print(random.random()); |

Example of random built-in module.

|  |
| --- |
| from random import randint;  answer = randint(1,10);  while True:  try:  guess = int(input('Guess a number 1 to 10: '));  if 0 < guess < 11:  if(guess == answer):  print(f'Yeah we found the number {guess}');  break;  else:  print('Please enter number between 1 to 10');  except ValueError:  print('Please enter a number.');  continue; |

##### Collection module

#### Pypi

<https://pypi.org/>

Using Pypi we can download or Shared code to the world. Good practice is to check first where the package is present in built-in module, if not then we can search pypi community.

**Pypi is just npm in JavaScript.**

##### Pip install

<https://www.makeuseof.com/tag/install-pip-for-python/>

pip install <packagename>

##### Pyvenv

It helps us to create different version of the same packages for different project.

## Debugging

#### Pdb

Pdb is the built-in module in python.

<https://docs.python.org/3/library/pdb.html>

|  |
| --- |
| import pdb;  def add(num1,num2):  pdb.set\_trace();  return num1 + num2;  print(add(8,'chkj')) |

## File I/O

|  |
| --- |
| my\_file = open('test.txt'); print(my\_file.read()); print(my\_file.readline()); print(my\_file.readlines()); my\_file.close(); |

With open , if we use with open then we no need to close the file.

|  |
| --- |
| with open('test.txt') as my\_file:  print(my\_file.readline()); |

#### Write

**By default, it will take as read, if we not specify the mode.**

Also, if you want to specify both read and write, then **mode = ‘r+’**

|  |
| --- |
| with open('test.txt',mode='w') as my\_file:  text = my\_file.write('Hello.....');  print(text); |

#### Append

Append use to add the text at the end of the file.

|  |
| --- |
| with open('test.txt',mode='a') as my\_file:  text = my\_file.write('How are you....');  print(text); |

#### Pathlib

Pathlib is a in-built module for working the path directory for both windows and linux operating system.

<https://docs.python.org/3/library/pathlib.html>

#### File IO Errors

|  |
| --- |
| try:  with open('test.txt',mode='r') as my\_file:  text = my\_file.read();  print(text);  except FileNotFoundError as err:  print('FIle not founf');  raise err;  except IOError as err:  print('IO Erro');  raise err; |

## Regular Expressions

|  |
| --- |
| import re;  pattern =re.compile('this')  string = 'Search inside of this text please!';  #search using regaular expression  a= re.search('this',string);  #search using pattern  b= pattern.search(string);  print(a.span()); |

<https://www.w3schools.com/python/python_regex.asp>

<https://regex101.com/>

<https://regexone.com/>

Email search

|  |
| --- |
| import re;  pattern =re.compile(r"(^[a-zA-Z0-9\_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$)");  string = 'chiranjitsaha9955@gmail.com';  #search using pattern  b= pattern.search(string);  print(b); |

Password match

|  |
| --- |
| import re;  pattern2 = re.compile( r"[A-Za-z0=9$%!&]{8,}\d");  password = 'chirnn&fffjj9'  check = pattern2.fullmatch(password);  print(check); |

## Testing

#### Unit Testing

For every python file, we will have one test python also.

**In python file (for example main.py)**

|  |
| --- |
| def sum (a,b):  try:  return a + b;  except TypeError as err:  return err; |

**In test file (for example test.py)**

|  |
| --- |
| #inbuilt module  import unittest;  #import the python file we want to test  import main;  class TestMain(unittest.TestCase):  def test\_sum1(self):  test\_param1 = 10;  test\_param2 = 20;  result = main.sum(test\_param1,test\_param2);  self.assertEqual(result,30);  def test\_sum2(self):  test\_param1 = 'abc';  test\_param2 = 20;  result = main.sum(test\_param1,test\_param2);  self.assertTrue(isinstance(result, TypeError));  #Here main doesnot means the python, but it tells to run  if \_\_name\_\_ == '\_\_main\_\_':  unittest.main(); |

Asserts method

<https://docs.python.org/3/library/unittest.html#assert-methods>

##### SetUp() method

If we want to run a method before every testcase, also it is an inbuilt method.

|  |
| --- |
| #inbuilt module  import unittest;  #import the python file we want to test  import main;  class TestMain(unittest.TestCase):  def setUp(self):  print('About to test a function');    def test\_sum1(self):  test\_param1 = 10;  test\_param2 = 20;  result = main.sum(test\_param1,test\_param2);  self.assertEqual(result,30);  def test\_sum2(self):  test\_param1 = 'abc';  test\_param2 = 20;  result = main.sum(test\_param1,test\_param2);  self.assertTrue(isinstance(result, TypeError));  unittest.main(); |

##### teardown()

If we want to run a method, at the end of each test cases. This method is basically used for cleaning some variable, data instance. We don’t often used frequently.

|  |
| --- |
| #inbuilt module  import unittest;  #import the python file we want to test  import main;  class TestMain(unittest.TestCase):  def setUp(self):  print('About to test a function');  def test\_sum1(self):  test\_param1 = 10;  test\_param2 = 20;  result = main.sum(test\_param1,test\_param2);  self.assertEqual(result,30);  def test\_sum2(self):  test\_param1 = 'abc';  test\_param2 = 20;  result = main.sum(test\_param1,test\_param2);  self.assertTrue(isinstance(result, TypeError));    def tearDown(self):  print('Cleaning up');  if \_\_name\_\_ == '\_\_main\_\_':  unittest.main(); |

## Scripting

#### What is Image Processing?

Image processing refers to the manipulation of digital images in order to extract more information than is visible on the original image. ... Image processing techniques use filters to enhance an image. Their main applications are to transform the contrast, brightness, resolution and noise level of an image.

In Python we have many modules, which we can use for image processing: -

pillow

<https://pillow.readthedocs.io/en/stable/>

## Scraping

#### Web Scraping

Web Scraping means programmatically using python or any languages grabbing data from a website.

<https://www.synerzip.com/blog/web-scraping-introduction-applications-and-best-practices/#:~:text=Web%20scraping%20typically%20extracts%20large,show%20data%20from%20a%20website>.

#### Library we use for Scraping

1. Beautiful Soup :- <https://www.crummy.com/software/BeautifulSoup/>

Pip4 install beautifulsoup4.

1. Request:- <https://pypi.org/project/requests/> It allows to access html file

|  |
| --- |
| # request to get the html response  import requests  # import Beautiful soup  from bs4 import BeautifulSoup  # import pprint to print the text in pritier way  import pprint  # Getting the response from html  res = requests.get('https://news.ycombinator.com/news')  res2 = requests.get('https://news.ycombinator.com/news?p=2')  # Parsing the html  soup = BeautifulSoup(res.text, 'html.parser')  soup2 = BeautifulSoup(res2.text, 'html.parser')  # Select the class from the parse html  links = soup.select('.storylink')  subtext = soup.select('.subtext')  links2 = soup2.select('.storylink')  subtext2 = soup2.select('.subtext')  mega\_links = links + links2  mega\_subtext = subtext + subtext2  def sort\_stories\_by\_votes(hnlist):  return sorted(hnlist, key=lambda k: k['votes'], reverse=True)  def create\_custom\_hn(links, subtext):  hn = []  for idx, item in enumerate(links):  title = links[idx].getText()  href = links[idx].get('href', None)  vote = subtext[idx].select('.score')  if len(vote):  points = int(vote[0].getText().replace(' points', ''))  if points > 99:  hn.append({'title': title, 'link': href, 'votes': points})  return sort\_stories\_by\_votes(hn)  pprint.pprint(create\_custom\_hn(mega\_links, mega\_subtext)) |

1. Scrapy <https://scrapy.org/>   
   An open source and collaborative framework for extracting the data you need from websites. In a fast, simple, yet extensible way.

## Web Deployment

#### Flask :-

<https://flask.palletsprojects.com/en/1.1.x/>

Flask is a tool in python that uses http server but instead make sure that the security, added tools and benefits are already prebuilt for us.

In python also have Django framework which is big framework. On the other hand Flask is small and clean framework.

**Flow the installation step for Flask as mentioned in the below URL**

<https://flask.palletsprojects.com/en/1.1.x/installation/>

**To get the dependency library files in the application**

pip3 freeze > requirements.txt

#### Deployment of Python web application

Pythonanywhere

<https://www.pythonanywhere.com/>

Step to deploy

<https://help.pythonanywhere.com/pages/Flask/>

## Testing/Automation

We use selenium for testing automation

<https://selenium-python.readthedocs.io/index.html>

<http://allselenium.info/python-selenium-commands-cheat-sheet-frequently-used/>

**Use the below website for testing in Selenium**

<https://www.seleniumeasy.com/test/>

## Machine Learning +Data Science

|  |
| --- |
|  |

### What is Machine learning?

We use Machine to predict result based on incoming data.

### Types of Machine Learning

|  |
| --- |
|  |

|  |
| --- |
| #1 - Import the data  #2 - Clean the data  #3 - Split data. Training Set/Test Set  #4 - Create a Model  #5 - Check the output  #6 - Improve |