## Core Python Syllabus

1. **Python introduction**

* **What is python**

Guido van Rossum begin in 1980 and released the first version of 1991. Python is High level general purpose computer programming language. It is simple as like to learn but has vast application

* **Hello world in python**

print(‘Hello world’)

or

for max\_leaf\_nodes **in** [5, 50, 500, 5000]:

my\_mae = get\_mae(max\_leaf\_nodes, train\_X, val\_X, train\_y, val\_y)

print("Max leaf nodes: **{}** **\t\t** Mean Absolute Error: **{}**".formate(max\_leaf\_nodes, my\_mae))

or different way

for max\_leaf\_nodes **in** [5, 50, 500, 5000]:

my\_mae = get\_mae(max\_leaf\_nodes, train\_X, val\_X, train\_y, val\_y)

print("Max leaf nodes: **%d** **\t\t** Mean Absolute Error: **%d**" %(max\_leaf\_nodes, my\_mae))

type

abs(-32) 32

* **Type of object and advantages**

Two type of object user defined and built in object (it is also known function )

Built-in objects make programs easy to write

Built-in objects are components of extensions

Built-in objects are often more efficient than custom data structures

Built-in objects are a standard part of the language.

* **Indentation rules and comment**

Indentation: Code depends on the how to write, don’t depends any {} is requires to way of execution

for I in range(100):

print(i)

* **Variable and data type**

Variable Naming convention: Variable is container to store the data in python language and stored data is called value and is written in a to z any letter and by \_ but don’t start special character or number

A=110

A is variable and 110 is value

Variable is that thing though identity of value

Some useful keyword of python

False def if raise

None del import return

True elif in try

and else is while

as except lambda with

assert finally nonlocal yield

break for not

class from or

continue global pass

Reserved Keyword

dunder names. These can be functions or properties such as \_\_new\_\_(), \_\_init\_\_(), \_\_name\_\_, \_\_main\_\_, etc.

\_\*, \_\_\*\_\_, \_\_\*

* **Different data type of Python**

**Scalar Types**

Int : 1,3,5

Float : 2.3

Complex: 2+3j

Bool : True / False

None: represent null values

**Sequence Type**

String : ‘itnesh’

(print(bool(1)) *# all numbers are treated as true, except 0*

print(bool(0))

print(bool("asf")) *# all strings are treated as true, except the empty string ""*

print(bool(""))

True

False

True

False)

(

x = 12

*# x is a real number, so its imaginary part is 0.*

print(x.imag)

*# Here's how to make a complex number, in case you've ever been curious:*

c = 12 + 3j

print(c.imag

0

3.0

There is no use in real life coding

)

List: [1,2,'itnesh']

Tuple: (1,2,'itnesh')

**Mapping Type**

Dictionary: {"name": "itnesh", "age":25}

**Set Types**

Sets: {1,2,'itnesh'}

perform mathematical set operations like union, intersection, difference, etc.

* **Type conversion**

int(2.3) convert into integer =2

str(2) convert into string ='2'

float(5) convert into 5.0

tuple('s') convert into ('s',)

list('1') convert into ['1']

set('d') convert into {'d'}

dict([(1,'apple'), (2,'ball')]): {1: 'apple', 2: 'ball'}

* **STRING MANIPULATION**

*a ="Hello World"* # this is string

**Accessing**

*letter=a[0]*

*print (letter*)

*H*

*Length*

*len(a)*

*11*

*print(a.count('l'))*

*3*

*print(a.find("H"))*

*0*

*print(a.index("World"))*

*6*

**Slicing**

*print(a[0:3])*

*Hel*

*print(a[:3])*

*Hel*

*print(a[3:])*

*lo World*

*print(a[:-3])*

*Hello Wo*

**Splitting**

*list(a.split(' '))*

# Split on whitespace

*['Hello', 'World']*

**Startswith / Endswith**

*a.startswith("H")*

*True*

*a.endswith("d")*

*True*

*a.endswith("w")*

*False*

**Changing Upper and Lower Case Strings**

*print(a.lower())*

*hello world*

*print(a.title())*

*Hello World*

*print(a.upper())*

*HELLO WORLD*

*print(a.swapcase())*

*hELLO wORLD*

**Replacing**

*a.replace("Hello", "Goodbye")*

*'Goodbye World*

**Strip**

*j=" kumar "*

*j.strip()*

*#removes from both ends*

*'kumar'*

***rstrip() Method***

1. ***First use***

*txt = " banana "*

*x = txt.rstrip()*

*banana*

1. *Seconde use to Remove the trailing characters if they are commas, s, q, or w*

*txt = "banana,,,,,ssqqqww....."*

*x = txt.rstrip(",.qsw")*

*print(x)*

*banana*

**Concatenation:** addition

*"Hello" + "World"*

*"HelloWorld"*

**Join**

*print(":".join(a))*

# #add a : between every char

*H:e:l:l:o: :W:o:r:l:d*

print("hello", end='')

print("pluto", end='')

*hellopluto*

claim = "Pluto is a planet!"

claim.startswith(planet)

*True*

datestr = '1956-01-31'

year, month, day = datestr.split('-')

'/'.join([month, day, year])

'01/31/1956'

* **Mutable and Immutable**: mutable though can be change on the basis of requirement that data type but immutable made at single time (no any change)

**Mutable** : List or Dictionary

**Immutable** : Numbers, strings, and Tuples

* **Python list**

**Indexing**

*L=[2,35,’rohan’,3]*

*Print(L[1])*

*35*

*O=[3,5,6,[‘man’,’sohan’]]*

*O[2][0]*

*Man*

*O[-1]*

*[‘man’,’sohan’]*

*['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']*

planets.index('Earth')

2

*2*

**Slicing**

*My\_list=[‘p’,’q’,’r’,’t’,’d’]*

*My\_list[1:3]*

*Q,r.t*

**Appending add in last in list**

*My\_list.append(2)*

*My\_list=[‘p’,’q’,’r’,’t’,’d’,7]*

**Extend**

*My\_list.extend([3,5,6])*

*[‘p’,’q’,’r’,’t’,’d’,7,3,5,6]*

**Concatenation :** addition of two data type except for number

*Odd=[7,51,3]*

*Even=[2,4,6]*

*Print(Odd+Even)*

*[7,51,3,2,4,6]*

**Insert**

*Odd.insert(1,3)*

*[7,3,51,3]*

**Delete**

*Del odd[51]*

*[7,3,3]*

**Remove**

*My\_list.remove(‘p’)*

*[‘q’,’r’,’t’,’d’,7]*

**Remove from last element pop**

*Print(My\_self.pop)*

*7*

.**count(), .sort(), .clear(),.reverse()**

* **Python tupple**

*tup1 = ('physics', 'chemistry', 1997, 2000)*

*print(tup1[0]): physics*

*del tup1*

**len, concatenation**

**repetation**

*(1,2,3)\*3=(1,2,3,1,2,3,1,2,3)*

*menbership1 in (1,2,3):*

*True*

**Iteration**

Calling the values one by one

*for x in (1,2,3):*

*print(x)*

*1*

*2*

*3*

*Find the numerator and denominator in tuple*

x = 0.125

x.as\_integer\_ratio()

*(1, 8)*

*Q.* We're using lists to record people who attended our party and what order they arrived in. For example, the following list represents a party with 7 guests, in which Adela showed up first and Ford was the last to arrive:

party\_attendees = ['Adela', 'Fleda', 'Owen', 'May', 'Mona', 'Gilbert', 'Ford']

A guest is considered 'fashionably late' if they arrived after at least half of the party's guests. However, they must not be the very last guest (that's taking it too far). In the above example, Mona and Gilbert are the only guests who were fashionably late.

Complete the function below which takes a list of party attendees as well as a person, and tells us whether that person is fashionably late.

*def fashionably\_late(arrivals, name):*

*"""Given an ordered list of arrivals to the party and a name, return whether the guest with that*

*name was fashionably late.*

*"""*

*order= int(arrivals.index(name))*

*return order >= len(arrivals)/2 and order != len(arrivals)-1*

*list comprehension*

*planets = ['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']*

short\_planets = [planet for planet **in** planets if len(planet) < 6]

*['Venus', 'Earth', 'Mars']*

*Any function*

*When any one exist and a*

*All funciton*

*When all condition is true*

* *When defining a function return something then function end so you have to return again then it will use as else statement.*

*for num in nums:*

*if num % 7 == 0 and num != 0:*

*return True*

*return False*

* Remember that return causes a function to exit immediately, So our original implementation always ran for just one iteration.

**indexing, slicing,**

* **Python sets**

It gives the ouput in order form

*Days={"Mon","Tue","Wed","Thu","Fri","Sat"}*

*Days.add("Sun")*

*print(Days)*

*Days.discard('Tue')*

*{'Thu', 'Fri', 'Mon', 'Sun', 'Sat', 'Wed'}*

*{'Thu', 'Fri', 'Tue', 'Mon', 'Sun', 'Sat', 'Wed'}*

**UNION**

*s1={1,2,3,4,5}*

*s2={4,5,6,7,8}*

*s1.union(s2)*

*{1, 2, 3, 4, 5, 6, 7, 8}*

[*set.pop()*](https://www.tutorialsteacher.com/python/set-pop)*,* [*set.remove()*](https://www.tutorialsteacher.com/python/set-remove)

* **Python dictionaries**

It is form of keys and values pair

*dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}*

*dict['Name']*

*Zara*

*dict['Age'] = 8;* # update existing entry

*del dict['Name'];* # remove entry with key 'Name'

*dict.clear();* # remove all entries in dict

*pop\_ele = dict.pop(‘Name’)*

*{'Age': 7, 'Class': 'First'}*

*d = {'a': 10, 'b': 20, 'c': 30}*

*d.items()*

*dict\_items([('a', 10), ('b', 20), ('c', 30)])*

*list(d.keys())*

*['a', 'b', 'c']*

*list(d.values())*

*[10, 20, 30]*

*d1 = {'a': 10, 'b': 20, 'c': 30}*

*d2 = {'b': 200, 'd': 400}*

*d1.update(d2)*

*d1*

*{'a': 10, 'b': 200, 'c': 30, 'd': 400}*

* **Python operators**

**Arithmetic Operators:**

+ : addition: 2+3: 5

- : Subtraction: 5-2:3

\* :Multiplication: 5\*6: 30

/ :division: 4/5: 0.8

% :modulus: 5/2: 1

\*\* :Exponentiation: 2\*\*2:4

// :Floor division: 5//2: 2

**Assignment Operators:**

= x=5 assignment operators

+= x += 3 x = x + 3

-= x -= 3 x = x – 3

\*= x\*=3 x=x\*3

Etc

**Comparison Operators:**

== Equal x == y

!= Not equal x != y

Greater than x > y

< Less than x < y

>= Greater than or equal to x>=y

<= less than or equal to x<=y

**Logical Operators:**

and: both statement correct like then true : x>2 and x<9

or : at least one condition is true then true : x < 5 or x < 4

not : no any condition is correct then true: not(x < 5 and x < 10)

and firstly operate rather than or

**Identity Operators:**

is : Returns True if both variables are the same object: x is y

is not: Returns True if both variables are not the same object: x is not y

Membership Operators

in : return true if x present in y: Y in x

not in : return true if x is not present in y: Y not in x

# **Select the best from the data comming**

def get\_mae(max\_leaf\_nodes, train\_X, val\_X, train\_y, val\_y):

model = DecisionTreeRegressor(max\_leaf\_nodes=max\_leaf\_nodes, random\_state=0)

model.fit(train\_X, train\_y)

preds\_val = model.predict(val\_X)

mae = mean\_absolute\_error(val\_y, preds\_val)

return(mae)

candidate\_max\_leaf\_nodes = [5, 25, 50, 100, 250, 500]

# Write loop to find the ideal tree size from candidate\_max\_leaf\_nodes

scores= {leaf\_size: get\_mae(leaf\_size, train\_X, val\_X, train\_y, val\_y) for leaf\_size in candidate\_max\_leaf\_nodes}

# Store the best value of max\_leaf\_nodes (it will be either 5, 25, 50, 100, 250 or 500)

best\_tree\_size = min(scores, key=scores.get)

1. **Python conditional statements**

* **if statement**

*food = 'spam'*

*if food == 'spam':*

*print('Ummmm, my favorite!')*

* **if else statement**

*if food == 'spam':*

*print('Ummmm, my favorite!')*

*else:*

*print("No, I won't have it. I want spam!")*

* **elif statement**

*var=input(‘Enter your char from a to c’)*

*if choice == 'a':*

*print("You chose 'a'.")*

*elif choice == 'b':*

*print("You chose 'b'.")*

*elif choice == 'c':*

*print("You chose 'c'.")*

*else:*

*print("Invalid choice.")*

**input function is used to get the values from user side**

**def to\_smash(total\_candies):**

**if** total\_candies **==** 1:

print("Splitting 1 candy")

**else**:

print("Splitting", total\_candies, "candies")

Here's a slightly more succinct solution using a conditional expression:

print("Splitting", total\_candies, "candy" **if** total\_candies **==** 1 **else** "candies")

* **Nested conditionals**

*if x < y:*

*STATEMENTS\_A*

*else:*

*if x > y:*

*STATEMENTS\_B*

*else:*

*STATEMENTS\_C*

* **Recursion** : repeat the same task until and unless the desired condition don’t meet

*def factorial(n):*

*if n in [0,1]:*

*return 1*

*else:*

*return n\*factorial(n-1)*

* **The for loop**

*for friend in ['Margot', 'Kathryn', 'Prisila']:*

*print(friend)*

*Margot*

*Kathryn*

*Prisila*

***Enumerate*** *function*

It allows us to loop over something and have an automatic counter, It shows the index and element both of them of same

my\_list = ['apple', 'banana', 'grapes', 'pear']

**for** counter, value **in** enumerate(my\_list):

print counter, value

*Output:*

*0 apple*

*1 banana*

*2 grapes*

*3 pear*

**Zip**

Itrate two list at a time

**for** i, c **in** zip(indices, clf\_names):

[plt.text](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.text.html#matplotlib.pyplot.text)(-.3, i, c)

ref: https://www.saltycrane.com/blog/2008/04/how-to-use-pythons-enumerate-and-zip-to/

* **The while statement**

*i=1*

*While i<6:*

*Print (i)*

*i+=1*

* **The break statement**

To break the statement at condition, after that no any execution will be proceed

*for i in [12, 16, 17, 24, 29]:*

*if i % 2 == 1: # if the number is odd*

*break*

* **The continue statement**

It skip that value given in condition

*for i in [12, 16, 17, 24, 29, 30]:*

*if i % 2 == 1: # if the number is odd, don't process it*

*continue*

*print(i)*

* **List comprehensions**

*numbers = [1, 2, 3, 4]*

*[x\*\*2 for x in numbers]*

*[1, 4, 9, 16]*

***Nested List comprehension***

results = [[x[i] **for** x **in** results] **for** i **in** range(4)]

* **Range function**

range(3) : range(0,1) I can iterate this value by looping

*for i in range(2,5):*

*print(i)*

*2*

*3*

*4*

1. **File and exception handling**

* **File**

**File open**

*open(filename, mode)*

*“ r “,* for reading.

*“ w “,* for writing.

*“ a “,* for appending.: it append the text from last time cursor is present in file

*“ r+ “,* for both reading and writing

**read**

*file = open("file.text", "r")*

*print (file.read())*

**write**

*file = open('file.txt','w')*

*file.write("This is the write command")*

*file.write("It allows us to write in a particular file")*

*file.close()*

**append the text**

*file = open('file.txt','a')*

*file.write("This will add this line")*

*file.close()*

***Python – Functions***

**Defining a Function**

*def function\_name(parameters):*

*"""docstring"""*

*statement1*

*statement2*

*...*

*...*

*return [expr]*

**ARG Function**

*def greet(\*arg):*

*print ('Hello ', names[0], ', ', names[1], ', ', names[3])*

*greet('Steve', 'Bill', 'Yash')*

**Keyword Argument \*\*kwarg**

*def greet(\*\*person):*

*print('Hello ', person['firstname'], person['lastname'])*

*greet(firstname='Steve', lastname='Jobs')*

*greet(lastname='Jobs', firstname='Steve')*

*greet(firstname='Bill', lastname='Gates', age=55)*

*greet(firstname='Bill') # raises KeyError*

paramter\_name['keyword\_argument']

**Variable scope**

***Global Variable***

name='John'

def greet():

print ("Hello ", name)

output

greet()

Hello Steve

name

'Steve'

**Local Variable**

def greet():

name = 'Steve'

print('Hello ', name)

output

greet()

Hello Steve

>>> name

Traceback (most recent call last):

File "<pyshell#4>", line 1, in <module> name

NameError: name 'name' is not defined

**Local and Global Variables**

name = 'Steve'

def greet():

name = 'Bill'

print('Hello ', name)

output

>>> greet()

Hello Bill

>>> name

'Steve'

**Access Global Variables**

name = 'Steve'

def greet():

global name

name = 'Bill'

print('Hello ', name)

output

>>> name

'Steve'

>>> greet()

Hello Bill

>>> name

'Bill'

**Global Variables**

name = 'Steve'

def greet():

globals()['name'] = 'James'

name='Steve'

print ('Hello ', name)

output

>>> name

'Steve'

>>> greet()

Hello Steve

>>> name

'James'

Check <https://www.tutorialsteacher.com/python/local-and-global-variables-in-python>

* **Os module**

*import os*

# Get the current working

*cwd = os.getcwd()*

**Change**

*os.chdir('path')*

**Present directory**

*pwd()*

**Directory making**

*os.makedirs(path)*

**Directory listing**

*os.listdir(path)*

**Remove the Directory**

*os.rmdir(path)*

**Remove the file**

*os.remove(path)*

[*https://www.tutorialsteacher.com/python/collections-module*](https://www.tutorialsteacher.com/python/collections-module)

**Random**

**Generate Random Floats**

import random

>>> random.random()

0.645173684807533

**Generate Random Integers**

>>> import random

>>> random.randint(1, 100)

95

>>> random.randint(1, 100)

49

**Generate Random Numbers within Range**

>>> random.randrange(1, 10)

2

>>> random.randrange(1, 10, 2)

5

>>> random.randrange(0, 101, 10)

80

**Select Random Elements**

>>> import random

>>> random.choice('computer')

't'

>>> random.choice([12,23,45,67,65,43])

45

>>> random.choice((12,23,45,67,65,43))

67

**Random order the list**

>>> numbers=[12,23,45,67,65,43]

>>> random.shuffle(numbers)

>>> numbers

[23, 12, 43, 65, 67, 45]

>>> random.shuffle(numbers)

>>> numbers

[23, 43, 65, 45, 12, 67]

**Recursive Function**

def factorial(n):

if n == 1:

print(n)

return 1

else:

print (n,'\*', end=' ')

return n \* factorial(n-1)

| Exception | Description |
| --- | --- |
| AssertionError | Raised when the assert statement fails. |
| AttributeError | Raised on the attribute assignment or reference fails. |
| EOFError | Raised when the input() function hits the end-of-file condition. |
| FloatingPointError | Raised when a floating point operation fails. |
| GeneratorExit | Raised when a generator's close() method is called. |
| ImportError | Raised when the imported module is not found. |
| IndexError | Raised when the index of a sequence is out of range. |
| KeyError | Raised when a key is not found in a dictionary. |
| KeyboardInterrupt | Raised when the user hits the interrupt key (Ctrl+c or delete). |
| MemoryError | Raised when an operation runs out of memory. |
| NameError | Raised when a variable is not found in the local or global scope. |
| NotImplementedError | Raised by abstract methods. |
| OSError | Raised when a system operation causes a system-related error. |
| OverflowError | Raised when the result of an arithmetic operation is too large to be represented. |
| ReferenceError | Raised when a weak reference proxy is used to access a garbage collected referent. |
| RuntimeError | Raised when an error does not fall under any other category. |
| StopIteration | Raised by the next() function to indicate that there is no further item to be returned by the iterator. |
| SyntaxError | Raised by the parser when a syntax error is encountered. |
| IndentationError | Raised when there is an incorrect indentation. |
| TabError | Raised when the indentation consists of inconsistent tabs and spaces. |
| SystemError | Raised when the interpreter detects internal error. |
| SystemExit | Raised by the sys.exit() function. |
| TypeError | Raised when a function or operation is applied to an object of an incorrect type. |
| UnboundLocalError | Raised when a reference is made to a local variable in a function or method, but no value has been bound to that variable. |
| UnicodeError | Raised when a Unicode-related encoding or decoding error occurs. |
| UnicodeEncodeError | Raised when a Unicode-related error occurs during encoding. |
| UnicodeDecodeError | Raised when a Unicode-related error occurs during decoding. |
| UnicodeTranslateError | Raised when a Unicode-related error occurs during translation. |
| ValueError | Raised when a function gets an argument of correct type but improper value. |
| ZeroDivisionError | Raised when the second operand of a division or module operation is zero. |

* **Exception handling**

**Error**

1. ZeroDivisionError: Occurs when a number is divided by zero.
2. NameError: It occurs when a name is not found. It may be local or global.
3. IndentationError: If incorrect indentation is given.
4. IOError: It occurs when Input Output operation fails.
5. EOFError: It occurs when the end of the file is reached, and yet operations are being performed

**Exception**

*try:*

*a = int(input("Enter a:"))*

*b = int(input("Enter b:"))*

*c = a/b*

*except:*

*print("Can't divide with zero")*

try:

print('try block')

x=int(input('Enter a number: '))

y=int(input('Enter another number: '))

z=x/y

except ZeroDivisionError:

print("except ZeroDivisionError block")

print("Division by 0 not accepted")

else:

print("else block")

print("Division = ", z)

finally:

print("finally block")

x=0

y=0

print ("Out of try, except, else and finally blocks." )

output

try block

Enter a number: 10

Enter another number: 2

else block

Division = 5.0

finally block

Out of try, except, else and finally blocks.

1. **Object oriented concepts**

* **Oops concepts**

***Class***

*class Employee:*

*def \_\_init\_\_(self, name, id):*

*self.name = name*

*self.id = id*

**Inheritance ( object )**

*r=Employee(‘ram’,abc)*

* **Generators**

It is used for memory efficiency

*def infinite\_sequence():*

*num = 0*

*while True:*

*yield num*

*num += 1*

*gen = infinite\_sequence()*

*next(gen)*

*0*

And so on get upto want, withough gen it will go upto infinfinite and system will crass.

* **Inheritance**

It is the process to make another type of class by using of another class , that class is called parent class

*class Dog:*

*species = "Canis familiaris"*

*def \_\_init\_\_(self, name, age):*

*self.name = name*

*self.age = age*

*def \_\_str\_\_(self):*

*return f"{self.name} is {self.age} years old"*

*parenting the class*

*class Bulldog(Dog):*

*pass*

**Public Attributes**

class Student:

schoolName = 'XYZ School' # class attribute

def \_\_init\_\_(self, name, age):

self.name=name # instance attribute

self.age=age # instance attribute

**Access Public Members**

>>> std = Student("Steve", 25)

>>> std.schoolName

'XYZ School'

>>> std.name

'Steve'

>>> std.age = 20

>>> std.age

20

**Protected Attributes**

class Student:

\_schoolName = 'XYZ School' # protected class attribute

def \_\_init\_\_(self, name, age):

self.\_name=name # protected instance attribute

self.\_age=age # protected instance attribute

**Access Protected Members**

>>> std = Student("Swati", 25)

>>> std.\_name

'Swati'

>>> std.\_name = 'Dipa'

>>> std.\_name

'Dipa'

class Student:

def \_\_init\_\_(self,name):

self.\_name = name

@property

def name(self):

return self.\_name

@name.setter

def name(self,newname):

self.\_name = newname

>>> std = Student("Swati")

>>> std.name

'Swati'

>>> std.name = 'Dipa'

>>> std.name

'Dipa'

>>> std.\_name # still accessible

**Private Attributes**

class Student:

\_\_schoolName = 'XYZ School' # private class attribute

def \_\_init\_\_(self, name, age):

self.\_\_name=name # private instance attribute

self.\_\_salary=age # private instance attribute

def \_\_display(self): # private method

print('This is private method.')

>>> std = Student("Bill", 25)

>>> std.\_\_schoolName

AttributeError: 'Student' object has no attribute '\_\_schoolName'

>>> std.\_\_name

AttributeError: 'Student' object has no attribute '\_\_name'

>>> std.\_\_display()

AttributeError: 'Student' object has no attribute '\_\_display'

**instantiate class**

*jim = Bulldog("Jim", 5)*

*inherit*

*jim.speak("Woof")*

*'Jim says Woof'*

* **Polymorphism**

same names but carrying different functionalities

*class Audi:*

*def description(self):*

*print("This the description function of class AUDI.")*

*class BMW:*

*def description(self):*

*print("This the description function of class BMW.")*

*audi = Audi()*

*bmw = BMW()*

*for car in (audi,bmw):*

*car.description()*

This the description function of class AUDI

This the description function of class BM

* **Lambda functions**

lambda [arguments] : expression

*1.*

*x = lambda a, b : a \* b*

*print(x(5, 6))*

*30*

*2.*

*greet = lambda name: print('Hello ', name)*

*greet('Steve')*

*Hello Steve*

*3.*

*sum = lambda x, y, z : x + y + z*

*>>> sum(5, 10, 15)*

*30*

*4.*

(lambda x: x\*x)(5)

25

1. mapping

*sqrList = map(lambda x: x\*x, [1, 2, 3, 4]) # passing anonymous function*

*>>> next(sqrList)*

*1*

*>>> next(sqrList)*

*4*

*>>> next(sqrList)*

*9*

*>>> next(sqrList)*

*16*

*>>> next(sqrList)*

* **Debugging of code**

import pdb

*def functionB(first\_val=23, last\_val=72):*

*pdb.set\_trace()*

*response = funcA(first\_val, last\_vale)*

*result = response \* first\_val / 7*

*return result*

**Regex**

**re.match() function**

from re import match

mystr = "Welcome to TutorialsTeacher"

obj1 = match("We", mystr)

print(obj1)

obj2 = match("teacher", mystr)

print(obj2)

output

<re.Match object; span=(0, 2), match='We'>

None

**The match object has start and end properties.**

>>> print("start:", obj.start(), "end:", obj.end())

Output

start: 0 end: 2

**re.search() function**

from re import search

string = "Try to earn while you learn"

obj = search("earn", string)

print(obj)

print(obj.start(), obj.end(), obj.group())

7 11 earn

Output

<re.Match object; span=(7, 11), match='earn'>

**re.findall() Function**

from re import findall

string = "Try to earn while you learn"

obj = findall("earn", string)

print(obj)

output

['earn', 'earn']

**re.finditer() function**

from re import finditer

string = "Try to earn while you learn"

it = finditer("earn", string)

for match in it:

print(match.span())

output

(7, 11)

(23, 27)

**re.split() function**

from re import split

string = "Flat is better than nested. Sparse is better than dense."

words = split(r' ', string)

print(words)

**output**

['Flat', 'is', 'better', 'than', 'nested.', 'Sparse', 'is', 'better', 'than', 'dense.']

**re.compile() Function**

from re import \*

pattern = compile(r'[aeiou]')

string = "Flat is better than nested. Sparse is better than dense."

words = split(r' ', string)

for word in words:

print(word, pattern.match(word))

output

Flat None

is <re.Match object; span=(0, 1), match='i'>

better None

than None

nested. None

Sparse None

is <re.Match object; span=(0, 1), match='i'>

better None

than None

dense. None

* **Python module and user define module**

Like “os module” is python module but when made a function sum of two number in “test.py” then we want to use in different programming we just “import test” in that file.

We can see all the names of module by dir function

print(dir(module\_name))

1. **Tk inter library used for make GUI.**
2. **Beatifulsoup, tk inter and reauests libraruy**

**Algoritms**

**Searching**

**Sorting**

**Linkedlist**

**Stack(hip, pop)**

**Maps**

**Hashing**