Python features:

- Simple syntax
- Oops
- Dynamic typing
- Cross-platform support
- Huge community
- Easy to understand

Typecasting: process of changing datatype from one to another

String: are sequence of chars enclosed between '', " ", ''' '''

- " ", ' ' used to represent single line as string
- ''' ''' used to rep multiple line as string
- Can be indexed by positive(from front) negative(from end)

Datastructure:

- Built in: Tuple(), list[], Dict{},set
- User defined: Stack, queue, tree, linked list,graph

Tuple: ordered collection of elements enclosed within (), order of indexing is maintained as in array

- Can store heterogenous elements
- Tuples are immutable(once the values are set it cannot be changed)
- Two tuples can be joined using join() function, eg str="".join(tup1)

List: ordered collection of elements stored within []

- Can be heterogeneous
- List are mutable
- Elements can be inserted in a list using append(), eg list.append(cse)
- List.sort() can be used to sort elements in a list
- Follows positive and negative indexing methods

Dictionary: unordered collection of key-value pair enclosed within {}

- Heterogenous in nature
- Mutable in nature
- Updating one dictionary with another using update(), eg: x.update(y)
- Poping an element form dictionary using pop(), x.pop(xyz)

Sets: unordered and unique collection of elements enclosed within {}

- Can stored heterogenous elements

- Union and intersection operations can be performed
  union: x.union(y), intersection: x.intersection(y)
- Creating a set from a tuple, t=(1,2,3), s=set(t)

Function: block of code that does a particular job

Lambda function: special function in python that does not need to be defined. It is an anonymous function a function with no name, eg s=”hello”, b={lambda string:len(string)}

Lambda with filter: it is used to eliminate unwanted values filter()
eg: l=[1,2,3], l2=list(filter(lambda X:x>2,l)), print(l2), output:[3]

Lambda with map: used along with lambda fun to manipulate value of a list map()
eg: l=[1,2,3], l2=list(map(lambda x:x+2,l)), print(l2), output:[3,4,5]

Lambda with reduce: reduce functiontools can be used if we want to reduce the elements of the list to a consolidated value.
eg from functiontools import reduce, l=[1,3,2], l2=reduce(lambda x,y:x+y,l), print(l2), output:[6]

OOP in python:

Class in python:

- Class name starts with capital letters
- Self is required to allow the object to call the method
- Self is an inbuilt parameter, whenever we create a method inside the class, the first parameter will always be class

Public Member Function:

```
Class A:
    def start(self):                          output:
        print(“Car started”)                  Car Started
    def  stop(self):                          Car Stopped
        print(“Car Stopped”)
obj=A()
obj.start()
obj.stop()
```

For Private Member Function: add two underscore before member function, and how to access private member function

```
Class A:
      def __start(self):                          output:
          print("Car started")                     Inside Show
      def  __stop(self):                           Car Started
          print("Car Stopped")                      Car Stopped

       def show(self):
          print("Inside Show")
          self.__start()
          self__stop()


obj=A()
obj.show()
```

Protected Member Function: Accessed Through Inheritance

```
Class Car:                                          output:
         def _start(self):                          Car Started
           print("Car Started")                     Car Stopped
          def _stop(self):                          Bike Started
           print("Car Stopped")                     Bike Stopped
Class Bike(Car):
         def show():
           print("Bike Started")
           print("Bike Stopped")

b=Bike()
b._start()
b._stop()
b.show()
```

using __init()__ with class:

- Works as constructor in python
- Used to initialized attributes of the class
- Preceded and succeeded by two underscores , __init(self)__
- Called automatically when object is declared

```
class Student:

        def __init__(self,name,roll):
                self.name=name
                self.roll=roll
        def show():
                print(“Student Details:”)
                print(“Name”,self.name)
                print(“Roll”,self.roll)

s=Student(“XYZ”,20)
s.show()
```

output

```
Student Details:
Name: XYZ
Roll: 20
```

overriding in init() in base/super class: using super()

```
class A:
    def __init__(self,x):
        self.x=x

    def showA(self):
        print(“Inside A”)
        print(“Value of x:”self.x)

class B(A):

    def __init__(self,x):
        super().__init__(x)

    def showB(self):
        print(“Inside B”)
        print(“Value of x:”,self.x)

obj=B(20)
obj.showA()
obj.showB()
```

output:

```
Inside A
Value of x:20

Inside B
Value of x:20
```

Inheritance:

- Single Inheritance
- Multiple Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

Graphics User Interface(GUI) in Python: Tkinter is a library to developed GUI in python. It is embedded in the standard python installation

Container: component used to store and organize interface objects, in our the objects we want to store are termed widgets

Widgets: Represent any screen elements it can be button, label, textbox etc

Event Handler: action, routine, or function executed when we click a button

```
From tkinter import*
#create a window

r=Tk()
r.geometry(500x500)
r.title("GUI")
#create screen variables
a=Stringvar()
res=Stringvar()
#button function
def display():
        b=a.get()
        res.set("Welcome"+a+"!")

#create Entry details
lbl=Label(r,text="Enter Your name",font=10,bd=1)
lbl.pack(pady=10)
txt=Entry(r,textvariable=a,font=10,bd=1)
txt.pack(pady=10)
btn=Button(r,text="Enter",font=10,bd=1,command=dis)
btn.pack(pady=10)
#display variables
lbl1=Label(r,textvariable=res,font=10,bd=1)
lbl.pack(pady=10)
#call main loop
r.mainloop()
```