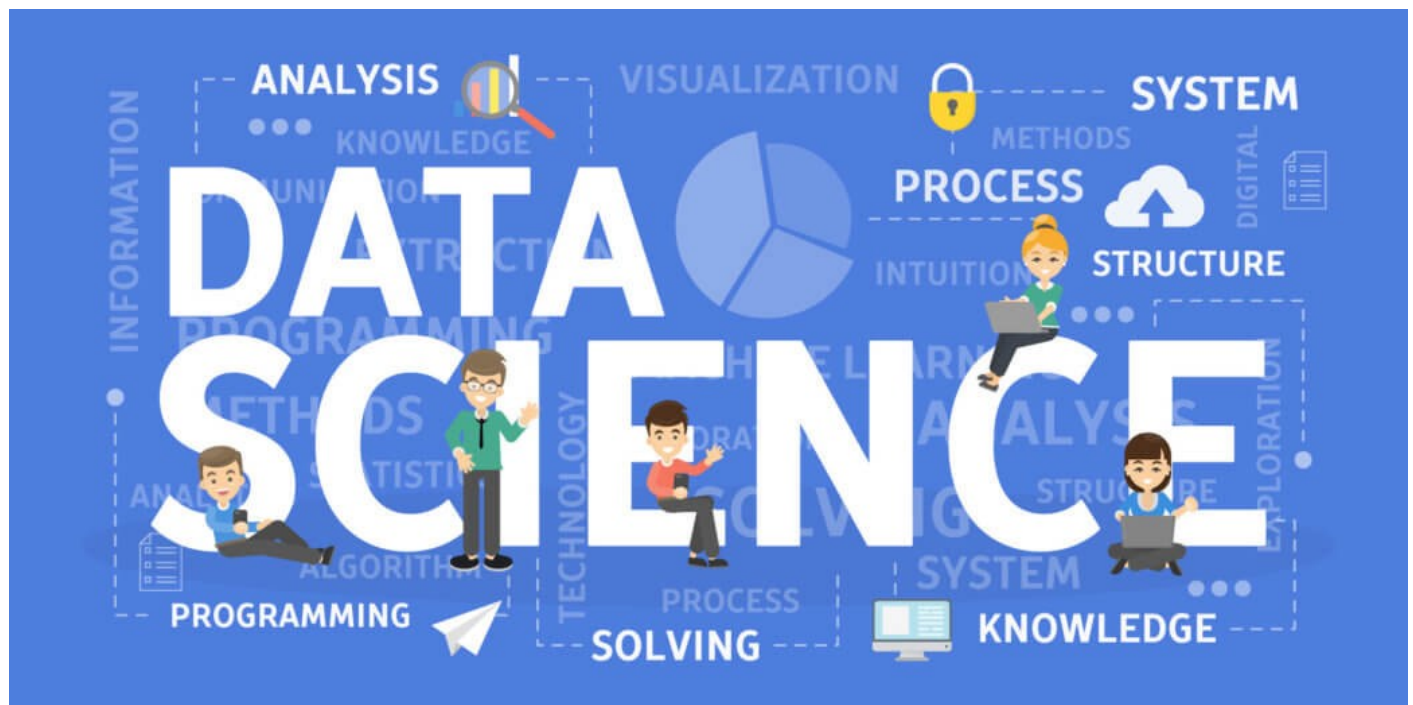


# Road to Data Science in 50 Days - Day 10

## Python for Data Science



# User Defined Functions

A function that you define yourself in a program is known as user defined function. You can give any name to a user defined function, however you cannot use the Python keywords as function name.

In python, we define the user-defined function using **def** keyword, followed by the function name.

Function name is followed by the parameters in parenthesis, followed by the colon

**For example:**

```
def function_name(parameter_1, parameter_2, ...) :  
    statements  
    ....
```

## Calling a user-defined function

You can call a user defined function by using function name followed by the arguments in the parenthesis.

**For example:**

```
function_name(argument_1, argument_2)
```

## Example of a user-defined function

In the following example we have defined a user-defined function **sum**. This function can accept two arguments as we have defined this function with two parameters. Inside the `print()` function we are calling the function **sum** and passing the numbers `x` & `y` as arguments.

In the **sum()** function we have a return statement that returns the sum of two parameters, that are passed to the function as arguments.

As you can see, we have used a **print()** function in the following example without even defining that function, this is because `print()` is a built-in function, which is already available to use and we can just call it.

```
In [2]: # Program to demonstrate the  
        # use of user defined functions  
  
def sum(a,b):  
    total = a + b  
    return total  
  
x = 10  
y = 20  
  
print("The sum of",x,"and",y,"is:",sum(x, y))
```

The sum of 10 and 20 is: 30

## Let's try some examples of User Defined Functions

### Example 1:- Arithmetic Progression

Write a function to return a list with arithmetic progression defined as follows :-

$a, a+d, a+2d, a+3d, \dots, a+(n-1)d$

```
In [3]: # Creating the function
def arithmetic_progression(a,d,n):
    ap_list = []
    for i in range(0,n,1):
        value = a+i*d
        ap_list.append(value)
    return ap_list
```

```
In [4]: ## Calling the function
a = arithmetic_progression(100,4,10)
print('Here the parameters are a = 100, d = 4 and n=10')
print("The list obtained after execution of function arithmetic_progression is ",a)
```

Here the parameters are a = 100, d = 4 and n=10

The list obtained after execution of function arithmetic\_progression is [100, 104, 108, 112, 116, 120, 124, 128, 132, 136]

### Example 2:- Greetings

Ask the user to enter his/her name and then say "Hi" to them

```
In [5]: def Greetings():
        name = input('Please enter your name: ')
        return 'Hi! '+name
```

```
In [6]: ## CALLING THE FUNCTION
Greetings()
```

Please enter your name: Saif

Out[6]: 'Hi! Saif'

### Example 3:- Palindrome or not

Enter a number/string and check if it is palindrome or not

```
In [13]: def Palindrome_check(value):
          value= value.lower()
          if value == value[::-1]:
              return('Palindrome')
          else:
              return('Non Palindrome')
```

```
In [14]: value = input('Enter a value to check if it is a Palindrome or not:- ')
          print('The text ',value,' is ',Palindrome_check(value))
```

Enter a value to check if it is a Palindrome or not:- abba  
The text abba is Palindrome

## Example 4:- Normalising an array

```
In [18]: import numpy as np
          def normalize_array(list_entered):
              array = np.float_(np.array(list_entered))
              return array/np.sqrt(array.dot(array))
```

```
In [19]: a = input('Enter list elements separated by space ')
          a = a.split(" ")
          print('The norm of ',a,' will be ',normalize_array(a))
          print('The dot product with itself for a normalized array must be 1.')
          print('We have ',round(normalize_array(a).dot(normalize_array(a)),2))
```

Enter list elements separated by space 1 2 4 2 6 2 64 2 7 35 24 6 3  
The norm of ['1', '2', '4', '2', '6', '2', '64', '2', '7', '35', '24', '6', '3'] will be [0.01284587 0.02569175 0.0513835 0.02569175 0.07707525 0.02569175  
0.82213599 0.02569175 0.08992112 0.44960562 0.308301 0.07707525  
0.03853762]  
The dot product with itself for a normalized array must be 1.  
We have 1.0

---

# Python Object Oriented Programming

---



## Object Oriented Programming

Python is a multi-paradigm programming language. It supports different programming approaches.

One of the popular approaches to solve a programming problem is by creating objects. This is known as Object-Oriented Programming (OOP).

An object has two characteristics:

- attributes
- behavior

Let's take an example:

A parrot is can be an object,as it has the following properties:

- name, age, color as attributes
- singing, dancing as behavior

The concept of OOP in Python focuses on creating reusable code. This concept is also known as DRY (Don't Repeat Yourself).

In Python, the concept of OOP follows some basic principles:

## Class

---



A class is a blueprint for the object.

We can think of class as a sketch of a parrot with labels. It contains all the details about the name, colors, size etc. Based on these descriptions, we can study about the parrot. Here, a parrot is an object.

The example for class of parrot can be :

```
class Parrot:  
    pass
```

Here, we use the **class** keyword to define an empty class **Parrot**. From class, we construct instances. An instance is a specific object created from a particular class.

## Object

An object (instance) is an instantiation of a class. When class is defined, only the description for the object is defined. Therefore, no memory or storage is allocated.

The example for object of parrot class can be:

```
obj = Parrot()
```

Here, **obj** is an object of class **Parrot**.

Suppose we have details of parrots. Now, we are going to show how to build the class and objects of parrots.

### Example 1: Creating Class and Object in Python

```
In [20]: class Parrot:

    # class attribute
    species = "bird"

    # instance attribute
    def __init__(self, name, age):
        self.name = name
        self.age = age

# instantiate the Parrot class
blu = Parrot("Blu", 10)
woo = Parrot("Woo", 15)

# access the class attributes
print("Blu is a {}".format(blu.__class__.species))
print("Woo is also a {}".format(woo.__class__.species))

# access the instance attributes
print("{} is {} years old".format( blu.name, blu.age))
print("{} is {} years old".format( woo.name, woo.age))

Blu is a bird
Woo is also a bird
Blu is 10 years old
Woo is 15 years old
```

In the above program, we created a class with the name **Parrot**. Then, we define attributes. The attributes are a characteristic of an object.

These attributes are defined inside the **init** method of the class. It is the initializer method that is first run as soon as the object is created.

Then, we create instances of the **Parrot** class. Here, **blu** and **woo** are references (value) to our new objects.

We can access the class attribute using **class.species**. Class attributes are the same for all instances of a class. Similarly, we access the instance attributes using **blu.name** and **blu.age**. However, instance attributes are different for every instance of a class.

To learn more about Class and Object, watch: <https://www.youtube.com/watch?v=8O5kX73OkIY&list=PLsyebzWxl7poL9JTVyndKe62ieoN-MZ3&index=53> (<https://www.youtube.com/watch?v=8O5kX73OkIY&list=PLsyebzWxl7poL9JTVyndKe62ieoN-MZ3&index=53>)

## Methods

Methods are functions defined inside the body of a class. They are used to define the behaviors of an object.

### Example 2 : Creating Methods in Python

```
In [21]: class Parrot:

    # instance attributes
    def __init__(self, name, age):
        self.name = name
        self.age = age

    # instance method
    def sing(self, song):
        return "{} sings {}".format(self.name, song)

    def dance(self):
        return "{} is now dancing".format(self.name)

# instantiate the object
blu = Parrot("Blu", 10)

# call our instance methods
print(blu.sing('Happy'))
print(blu.dance())

Blu sings 'Happy'
Blu is now dancing
```

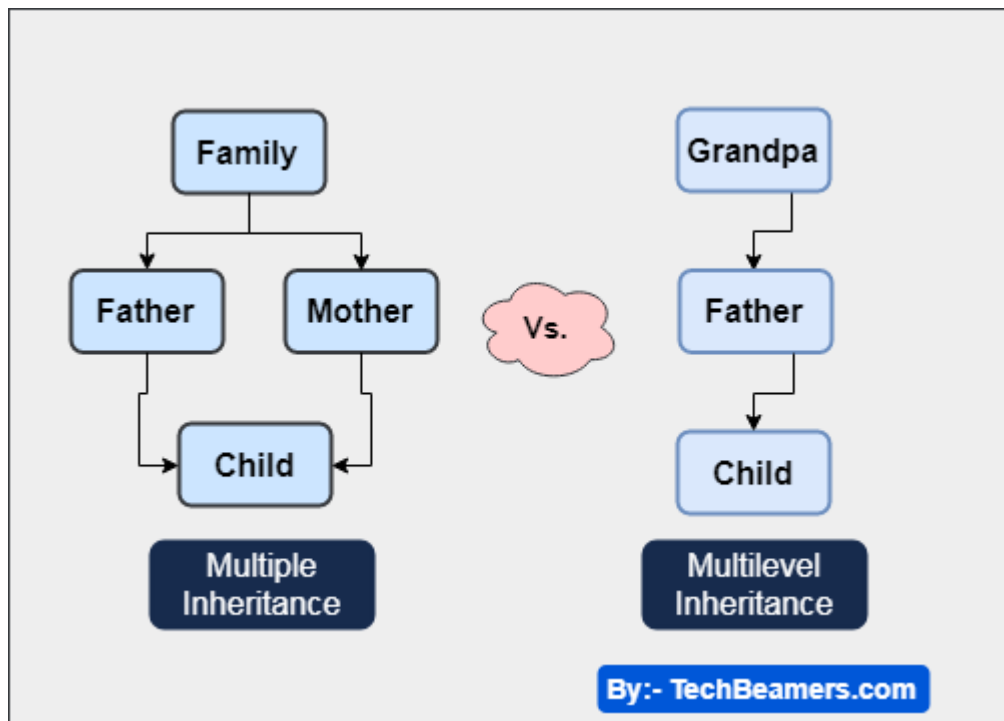


In the above program, we define two methods i.e **sing()** and **dance()**. These are called instance methods because they are called on an instance object i.e **blu**.

To learn more about Methods, watch: <https://www.youtube.com/watch?v=IVfGQOzzRCM&list=PLsyebzWxl7poL9JTVyndKe62ieoN-MZ3&index=57>

(<https://www.youtube.com/watch?v=IVfGQOzzRCM&list=PLsyebzWxl7poL9JTVyndKe62ieoN-MZ3&index=57>)

## Inheritance



Inheritance is a way of creating a new class for using details of an existing class without modifying it. The newly formed class is a derived class (or child class). Similarly, the existing class is a base class (or parent class).

### Example 3: Use of Inheritance in Python

```
In [22]: # parent class
class Bird:

    def __init__(self):
        print("Bird is ready")

    def whoisThis(self):
        print("Bird")

    def swim(self):
        print("Swim faster")

# child class
class Penguin(Bird):

    def __init__(self):
        # call super() function
        super().__init__()
        print("Penguin is ready")

    def whoisThis(self):
        print("Penguin")

    def run(self):
        print("Run faster")

peggy = Penguin()
peggy.whoisThis()
peggy.swim()
peggy.run()
```

```
Bird is ready
Penguin is ready
Penguin
Swim faster
Run faster
```

In the above program, we created two classes i.e. **Bird** (parent class) and **Penguin** (child class). The child class inherits the functions of parent class. We can see this from the **swim()** method.

Again, the child class modified the behavior of the parent class. We can see this from the **whoisThis()** method. Furthermore, we extend the functions of the parent class, by creating a new **run()** method.

Additionally, we use the **super()** function inside the **init()** method. This allows us to run the **init()** method of the parent class inside the child class.

To learn more about inheritance, watch: <https://www.youtube.com/watch?v=Cn7AkDb4pIU&list=PLsyebzWxl7poL9JTVyndKe62ieoN-MZ3&index=59> (<https://www.youtube.com/watch?v=Cn7AkDb4pIU&list=PLsyebzWxl7poL9JTVyndKe62ieoN-MZ3&index=59>)

## Encapsulation



Using OOP in Python, we can restrict access to methods and variables. This prevents data from direct modification which is called encapsulation. In Python, we denote private attributes using underscore as the prefix i.e single `_` or double `__`.

### Example 4: Data Encapsulation in Python

```
In [23]: class Computer:

    def __init__(self):
        self.__maxprice = 900

    def sell(self):
        print("Selling Price: {}".format(self.__maxprice))

    def setMaxPrice(self, price):
        self.__maxprice = price

c = Computer()
c.sell()

# change the price
c.__maxprice = 1000
c.sell()

# using setter function
c.setMaxPrice(1000)
c.sell()

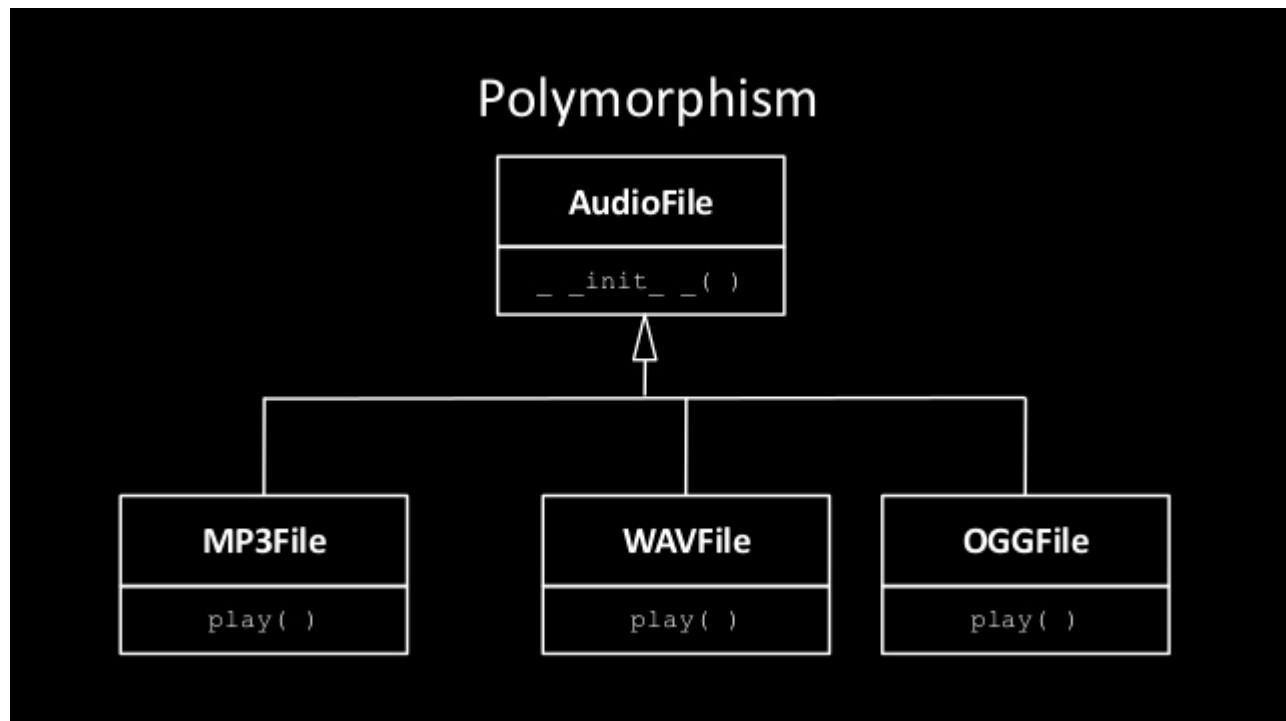
Selling Price: 900
Selling Price: 900
Selling Price: 1000
```

In the above program, we defined a **Computer** class.

We used **init()** method to store the maximum selling price of **Computer**. We tried to modify the price. However, we can't change it because Python treats the `__maxprice` as private attributes.

As shown, to change the value, we have to use a setter function i.e **setMaxPrice()** which takes price as a parameter.

# Polymorphism



Polymorphism is an ability (in OOP) to use a common interface for multiple forms (data types).

Suppose, we need to color a shape, there are multiple shape options (rectangle, square, circle). However we could use the same method to color any shape. This concept is called Polymorphism.

## Example 5: Using Polymorphism in Python

```
In [24]: class Parrot:

    def fly(self):
        print("Parrot can fly")

    def swim(self):
        print("Parrot can't swim")

class Penguin:

    def fly(self):
        print("Penguin can't fly")

    def swim(self):
        print("Penguin can swim")

# common interface
def flying_test(bird):
    bird.fly()

# instantiate objects
blu = Parrot()
peggy = Penguin()

# passing the object
flying_test(blu)
flying_test(peggy)
```

```
Parrot can fly
Penguin can't fly
```

In the above program, we defined two classes **Parrot** and **Penguin**. Each of them have a common **fly()** method. However, their functions are different.

To use polymorphism, we created a common interface i.e **flying\_test()** function that takes any object and calls the object's **fly()** method. Thus, when we passed the blu and peggy objects in the **flying\_test()** function, it ran effectively.

To learn more about Polymorphism: <https://www.youtube.com/watch?v=P1vH3Pfw6BI&list=PLsyebzWxl7poL9JTVyndKe62ieoN-MZ3&index=61> (<https://www.youtube.com/watch?v=P1vH3Pfw6BI&list=PLsyebzWxl7poL9JTVyndKe62ieoN-MZ3&index=61>)

## Key Points to Remember:

- Object-Oriented Programming makes the program easy to understand as well as efficient.
- Since the class is sharable, the code can be reused.
- Data is safe and secure with data abstraction.
- Polymorphism allows the same interface for different objects, so programmers can write efficient code.

By now, we have got a good hold of Python and moving forward we will learn Python by implementing the statistics terms we learned earlier and will also learn/use 2 of the most used libraries in Data Analysis i.e.

**Pandas and Numpy.**

For those who are absolute beginners, i would suggest to follow the links to refer section, learn python for some more day and then come back to this series.

## References and Links to Refer

Python tutorials by TutorialsPoint: <https://www.tutorialspoint.com/python/index.htm>  
(<https://www.tutorialspoint.com/python/index.htm>)

Python Tutorials by Programiz: <https://www.programiz.com/python-programming/first-program>  
(<https://www.programiz.com/python-programming/first-program>)

## Video Tutorials:

Practical Tutorial for OOP Concepts in Python by Edureka - <https://www.youtube.com/watch?v=nrEhpPM9rS8>  
(<https://www.youtube.com/watch?v=nrEhpPM9rS8>)

Python Tutorials by Corey Schafer (I learned from him): <https://www.youtube.com/watch?v=YYXdXT2l-Gg&list=PL-osiE80TeTt2d9bFvYTiXJA-UTHn6WwU> (<https://www.youtube.com/watch?v=YYXdXT2l-Gg&list=PL-osiE80TeTt2d9bFvYTiXJA-UTHn6WwU>)

Python Tutorials by Sentdex: [https://www.youtube.com/watch?v=oVp1vrFL\\_w4&list=PLQVvva0QuDe8XSftW-RAxdo6OmaeL85M](https://www.youtube.com/watch?v=oVp1vrFL_w4&list=PLQVvva0QuDe8XSftW-RAxdo6OmaeL85M) ([https://www.youtube.com/watch?v=oVp1vrFL\\_w4&list=PLQVvva0QuDe8XSftW-RAxdo6OmaeL85M](https://www.youtube.com/watch?v=oVp1vrFL_w4&list=PLQVvva0QuDe8XSftW-RAxdo6OmaeL85M))

Python Tutorials by Telusko(Indian Author - Recommended): <https://www.youtube.com/watch?v=QXeEoD0pB3E&list=PLsyebzWxl7poL9JTVyndKe62ieoN-MZ3> (<https://www.youtube.com/watch?v=QXeEoD0pB3E&list=PLsyebzWxl7poL9JTVyndKe62ieoN-MZ3>)