# CHAPTER 1

## PYTHON BASICS

## 

**Introduction:**

* Python is an open-source programming language.
* Cross-platform compatible means you can run python programming language on any operating system.
* Python also provides a large standard library. For example, python provides packages for data manipulation, web development, data visualization, etc.
* It is an object-oriented language so whatever data you have around you, you can represent all of that data in the form of an object so that you can find a solution for whatever problem you are headed to.

## IDE:

* **Pycharm**

- Integrated Development Environment for Python.

**- Anaconda**

- It acts as IDE and content distribution.

- [Install anaconda on fedora35](https://tecadmin.net/how-to-install-anaconda-on-fedora/)

**- Jupyter Notebook**

- It is an browser-based interpreter

| **Purpose** | **Command** |
| --- | --- |
| create env | $ conda create -n cosmic\_coding python=3.9 |
| activate env | $ conda activate cosmic\_coding |
| deactivate env | $ conda deactivate |
| Open GUI | $ export PATH=/home/rothakur/anaconda3/bin/:$PATH $ anaconda-navigator |
| jupyter-notebook | $ jupyter notebook |

## Variables and Data-Types:

* To store data like name, age, salary, etc you can use define variables in python.
* example employee\_name = "Rohit".
* Python is known as a dynamical type language which means you don't have to initialize variables with a specific data type.
* So whatever data type value has, the same data type will be associated with the variable as well.

## Basic Data-Types

| **Data Type** | **Example** |
| --- | --- |
| int | 1,30000, -1, 0 |
| float | 1.24, 5.2 |
| boolean | True, False |
| string | “Rohit”, “Blake” |
| complex | 3 + 5j |

### Operators:

* With the help of these operators various operations can be performed on the data

| **Operator** | **Example** |
| --- | --- |
| Arithmetic-Operators | * Allows arithmetic or mathematical operations on top of data * + - \* / * Example=> a=10 , b=10   a+b |
| Relational-Operators | * >, < , ==, != |
| Logical-Operators | * $, | * & only return True when both operands are True * | only returns False when both operands are False |

## 

### **Python Tokens:**

Python tokens are the smallest meaningful components in a program.

There are the following tokens:

| Token | Description |
| --- | --- |
| Keywords | * Keywords are special reserved words that give meaningful information to the compiler or interpreter. * When you type a keyword python interpreter or compiler treats it as a special word and it knows that it performs a special function. * You can not have a function name, class name, or a variable name from reserved words. |
| Identifiers | * Identifiers are names used for variables, functions, or objects. * For example in real life if you want to call someone you can call them by their name, the same goes for the variable, function, objects, etc. * Rules -> No special character except underscore. * Identifiers are case-sensitive. * The first letter can not be a digit. |
| Literals | * Literals are constant in python. * They do not change * Basically you can have numerical literals, string literals,   Boolean literals, etc.   * So whatever values are stored inside variables, those values are nothing but literals. * Example a1 = “Rohit”, so here “Rohit” is a string literal. |
| Operators | * With the help of these operators various operations can be performed on the data |

### **Python Strings:**

**Introduction:**

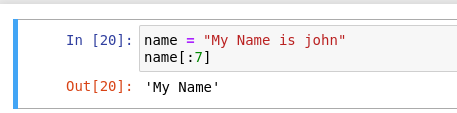
* Python strings are sequences of characters enclosed within single quotes(‘’), double quotes(“”) or triple quotes(‘’’ ‘’’)
* Example:

1. ‘Hello World’
2. “Hello World”
3. ‘’’Hello World could

Be a multiline string’’’

* Indexing starts from zero.
* Read a specific range that can be extracted.

It will read from the start index to one index before the end index is provided.



**String Functions:**

| **Function** | **Description** |
| --- | --- |
| len(my\_string) | * Finding the length of the string |
| my\_string.lower() | * To convert the string to lower case. |
| my\_string.upper() | * To convert the string to upper case. |
| Examples |  |
| my\_string.replace(‘s1’, ‘s2’) | * To replace a substring within a string * Example -> |
| my\_string.count(“Hello”) | * The count function gives the number of occurrences of a substring |
| my\_string.find(“Shimla”) | * Find function is used to find the first matched starting index of the provided substring. |
| cities.split(‘s1’) | * This function is used to split the string into substrings, it doesn’t include the provided splitter substring in the output list. |

## Data Structures In Python:

Till now we have only saved a singular value in a variable, but many times when we are dealing with a huge amount of data we would have to store multiple elements into a singular data structure, so that is where we will have this collection of variables. There are the following collections of variables.

* Tuples
* List
* Dictionary
* Set

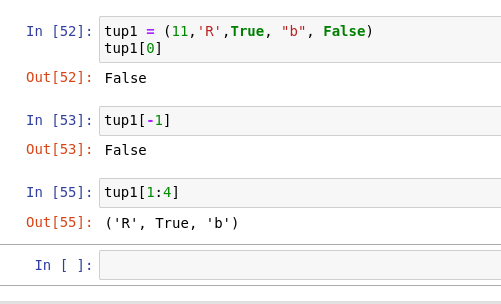
### **Tuple:**

* A tuple is basically an ordered collection of elements enclosed within ( ).
* Tuples are immutable means once you create a tuple you can not change the value present inside the tuple.
* Example: define a tuple with values of different data types.
* **tup1 = (11, ‘R’, False)**
* So tuple could be a mixture of heterogeneous elements and this is an ordered collection of those heterogeneous elements.
* Once you have created a tuple you can’t go ahead and change the tuple.

#### **Create a Tuple:**

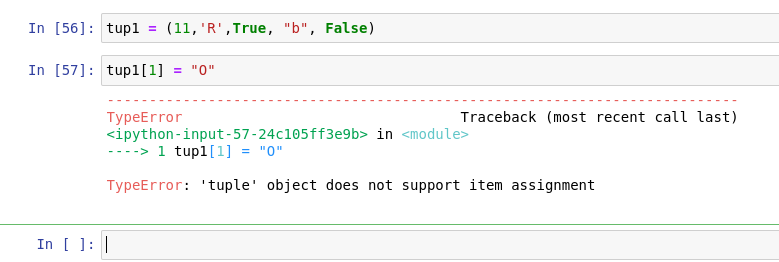
#### **Extracting Individual Elements:**

* In tuples indexing starts from the 0

****

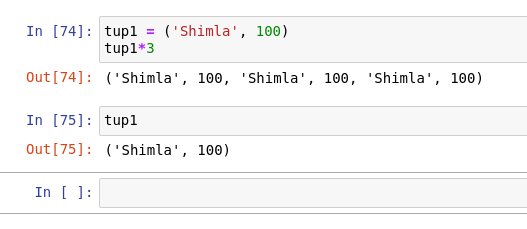
#### Immutable:

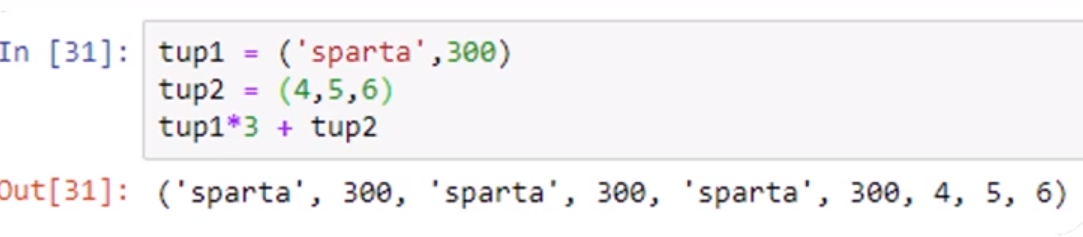
* You can not modify a tuple as it is immutable.



#### Basic Operation with Tuples:

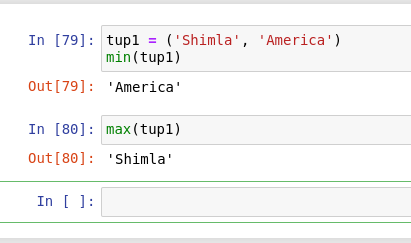
| **Operation** | **Example** |
| --- | --- |
| len(tup1) | * Finds the length of tuple |
| Tup1 + tup2 | * This concatenates two tuples and returns a new tuple |
|  | * **Sequence of tuples matter** |

**Repeating Elements in tuples:  
**

**Repeating and Concatenating:  
**

**Tuples Function:**

1. **Min:** gives the minimum value for the same data type
2. **Max** gives the maximum value of the same data type

****

### **List:**

* The list is an ordered collection of elements enclosed in square brackets.
* Difference between list and tuple is that tuples are immutable. That means values inside can not be modified once that is created.

|  |
| --- |

#### Extracting Individual Elements:

* In tuples indexing starts from the 0
* The last element can be extracted with [-1]
* Sequence of elements can be provided [0:6], the last element will be excluded.

|  |
| --- |

**Reverse Indexing:**

|  |
| --- |

#### Mutable:

##### **Change Value**

|  |
| --- |

##### **Append Value:** this will append the value at the end of the list.

|  |
| --- |

##### **Pop last element:** This will remove the last element from the list or any provided indexed value.

|  |
| --- |

#### Modifying a List:

##### **Reverse:** Reversing elements of a list:

|  |
| --- |

##### **Insert at specific:** you can insert a new element at a specific position. So

* For the rest of the index of the element will be incremented with 1.

|  |
| --- |

* **Sort a list:** you can sort the list in ascending as well as descending order.
* What happens when we perform sorting on **heterogeneous data.**

|  |
| --- |

* **Sorting on homogeneous data.**

|  |
| --- |

* Sorting in **reverse** order

|  |
| --- |

#### Concatenate and Repeat Elements of List:

* Concatenation:

|  |
| --- |

* Repeating Elements:

|  |
| --- |

### **Dictionary:**

* Dictionary is an unordered collection of key-value pairs enclosed with {}
* Dictionary is mutable (means if you have elements inside the dictionary you can always edit them)
* It has key-value pairs inside the dictionary.

#### Create Dictionary:

|  |
| --- |

#### 

#### Extract keys and values from the dictionary:

##### **keys():** Extract only the keys.

|  |
| --- |

##### **values()**: Extract only the values.

|  |
| --- |

##### **items():** It will extract all the key-value pairs

|  |
| --- |

#### 

#### 

#### 

#### 

#### Modifying a Dictionary:

##### **Adding new element /Changing existing element**

Add new element.

|  |
| --- |

Change existing element:

|  |
| --- |

#### 

#### Dictionary Functions:

##### **Update():** update one dictionary elements with another.

|  |
| --- |

##### **pop(<key-val>):** Popping an element.

|  |
| --- |

### **Set:**

* Set is an unordered and unindexed collection of elements enclosed with {}.
* You can’t access the individual element using an index.
* Duplicates are not allowed inside the set
* It is a heterogeneous data structure

#### Create Set:

|  |
| --- |

#### 

#### Set Operations:

##### **add():** add a value to the set.

##### 

|  |
| --- |

##### **remove(‘element name):** add a value to the set.

|  |
| --- |

##### **update(‘elements):** this will add multiple elements.

|  |
| --- |

#### Set Functions:

##### **Union:** Union of two sets, returns addition of two sets.

* Intersection: return common of two sets.

|  |
| --- |

## Conditional and Looping In Python:

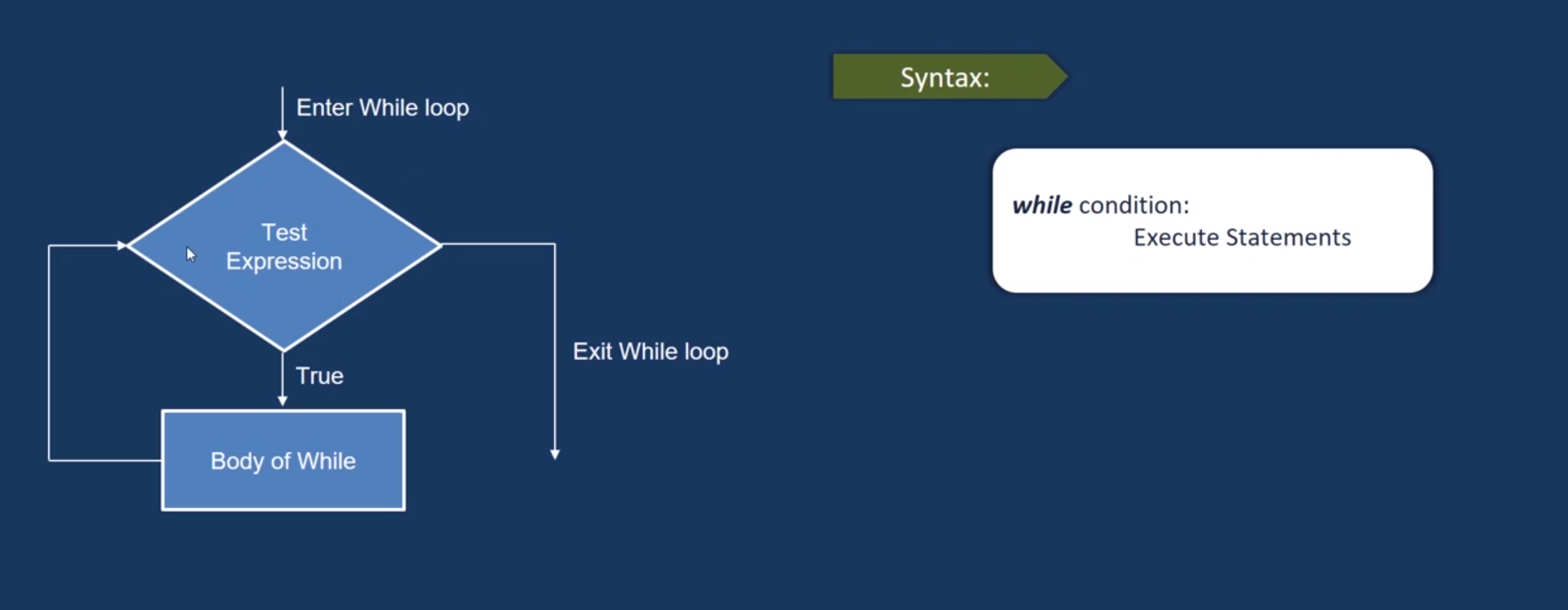
### **IF Statement**

This is a conditional statement.

|  |
| --- |

### **While Loop**

* Until the condition is false will keep executing all the statements.



|  |
| --- |

### **For Loop -** For loop is used to iterate over a sequence(tuple, list, dictionary)

|  |
| --- |

### 

## Functions in Python:

### **Introduction**

* A function is a block of code that performs a specific task.
* Let's say you have to do an online transaction and you have three options, either deposit money, withdraw money, or Balance a check.
* All of these functions' lines of code could be in thousands, so every time you are depositing cash you can just write thousands of lines of code.
* That's why you can create one function for a deposit, one function for withdrawing money, and one function for checking your balance.
* Whenever you want to deposit money you will invoke the specific function.

|  |
| --- |

### **Types of Function in Python**

- In python there are two types of functions. Normal and lambda.

#### Normal function:

| def greeting(name):  print(“Hello ”, name) |
| --- |

#### Lambda function:

Lamda Function with filter()

|  |
| --- |

|  |
| --- |

Lamda Function with map()

|  |
| --- |

# CHAPTER 2

## OOPs

## Introduction:

* Object-Oriented Programming is a programming paradigm that imitates the real world.
* For example, let's say I am recording a session, so what components will I require?
* Maybe a camera, laptop, or microphone, so here every component is a real-world object that possesses some properties and functionality, and using these we can achieve our task/ problem of recording a session.
* If we have to implement the above problem in the computer world then we need to map these physical objects into a virtual object and this mapping can be achieved by OOPs.

| **Note:**   * OOPs basically revolves around objects, as in our real world everything is an object, it tries to imitate the same fashion. * So OOPs revolves around the object and because of this imitation it becomes really easy to solve the real-world problems using OOPs. |
| --- |

## Example:

* Let's say you want to make a bowl of noodles for yourself, so what are the steps you are going to follow:

1. Turn on the heat.
2. Put the pan
3. Add water and keep it to boil
4. Add noodles and spices
5. Wait for a few minutes
6. Switch off the heat
7. Remove the Pan
8. Serve

* Now if you are going to implement the above real-world process into Object-Oriented Programming, every entity here will be an object, like Pan, water, noodles, etc.

## Advantages of Using OOPs:

### **DRY (Do Not Repeat Yourself)**

- Let's say now I want to make another dish and I need Pan, water then I don't have to create a different object for that.I can reuse those which were already created.

### **Reusability**

* As we can reuse the object.

### **Modularization**

* OOPs also enforces modularization as every object is a separate entity and can be handled separately.

## Object-Oriented Vs Procedural:

| **Top-Down vs Bottom-Up Approach:**   * PP follows a top-down approach but OOPs follow bottom-up. * To achieve a particular task you will have a sequence of instructions that has to be executed in order to achieve it. IN PP this flow moves from top-bottom. * In OOPs you have objects to represent entities, so let's say we have a student, maybe the employee, etc. |
| --- |

| * PP works with functions but OOPs divide the program objects designated to real-world entities * So whatever task you want to achieve in procedural programming can be implemented in terms of functions, so you can have different functions to perform subtasks. * But with OOPs we have different objects, which have some properties and behavior, and we are going to put that behavior and properties on an employee(object) together as a single entity when we are going to implement this problem in an OOP. |
| --- |

| * In PP you can have very less security by just keeping things in functions, etc * IN OOPs you have one piller known as encapsulation and in encapsulation, there are ways where you can hide the data from the outside world. * So you have access to modifiers that can be used to actually modify the access rights of a particular member. * So in OOPs, you can enable security by using encapsulation. |
| --- |

| * PP doesn’t emphasize code reusability but OOPs do. * In PP the only reusability we have is through functions. * In OOPs let's say we have an entity employee, let's say we are working on an institute problem, and let's say we have four departments IT, Admin, Finance, and Faculty. * So in all these four departments who are working? Employee Tada !!! * So everybody in these departments is also an employee. * So rather than having the attributes of employees and functionalities in each of the departments separately, we can have a common entity and we are going to inherit the things in this particular employee object in these four departments. * So every person in the IT department is going to possess properties of the IT department and employee., so here the code reusability happening. |
| --- |

| * OOPs offers modularization, different developer can work on different modules, for example, dev1 can work on the student module, dev2 can work on the teacher module, dev3 can work on the course module, etc. * Also helps in the maintenance of the code. |
| --- |

| * C, VB follies PP * Most HLL follies OOPS, Java, Python, C++, etc |
| --- |

## What is Class:

* CLASS is a skeleton, so CLASS basically is a blueprint that defines all the functionalities and properties of a particular entity.
* Let's say we are dealing with Animal Class, now what all properties Animals could have:

- type: what type of animal it is.

- color: the color of the animal.

- name: Name of the animal.

- the number of legs

- it can fly or not.

- plant eater or meat eater, etc

* SO what are the functionalities Animal Class could have:

- It is going to run.

- sleep

- eat

- hunt, etc

* Now if we were to create the Animal class then that needs to have all the above-mentioned properties and functionalities. They are going to club together in a code block which will together represent an ANimal entity.

### CLASS can also have constructors which are special functions and are used to initialize the properties.

* You can also have a nested class that is a class within the class.
* Class is USER DEFINED data structure, so store number we have INT in python, similarly, for strings we have string, but store Animal we don’t have anything that's why we need user-defined DATA STRUCTURE.

### **CLASS Enables Modularization:**

* Everything will be in a modularized way. This makes debugging, development, etc easy.

### **CLASS Implementation:**

|  |
| --- |

## What is an Object:

### Object is an instance of a CLASS. Class SUV could have instances as actual SUV cars.

### Object is REAL ENTITY, which means it exists physically and possesses all the properties and functionalities that are defined inside CLASS.

### MEMORY ALLOCATION takes place when an object is created. It means class which is just a logical idea is not going to occupy any memory.

## 

### **Object Implementation:**

|  |
| --- |

## 

## What is an Attribute:

* Attributes are properties and features of an entity, for example, Animals can have properties like type, color, name, etc. So all these properties are termed as attributes in OOPs world.

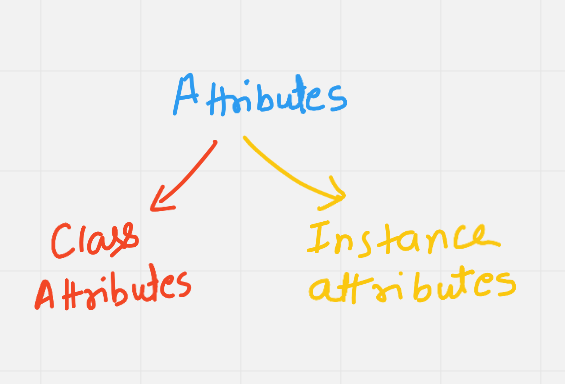
### State of an Object: Values of the attributes define the state of an object. So the state of an object refers to the current values in the attributes of this particular object, for example, type = ‘mammal’, color = ‘black’

### Build-In-Attributes: Python provides built-in attributes which hold special information about a particular class.

| **Attribute** | **Description** |
| --- | --- |
| \_\_dict\_\_ | * This holds the class namespace. |
| \_\_doc\_\_ | * This hols the documentation string of the class. |
| \_\_module\_\_ | * This defines the module name in which the class is defined. |
| \_\_name\_\_ | * This attribute holds the class name. |
| \_\_bases\_\_ | * If there is some inheritance happening then all the classes from which the inheriting is done are going to be present in this. |

### **Attribute Implementation:**

* Attribute is the property of a particular entity.
* Attributes are of two types in python class attributes and instance attributes.



### **Types of Attributes:**

| **INSTANCE ATTRIBUTES** | * Those attributes which are associated with an instance. * Whenever an object of the class is created then the instance properties will get assigned to each and every object. |
| --- | --- |
| **CLASS ATTRIBUTES** | * ALL the objects that are of this particular class * Will share that particular property. |

### 

### **INIT:**

* To have attributed as an instance attribute python has a special function known as def \_\_int\_\_().
* For \_\_init\_\_(): first param will be self and this self actually points to the current object. So this will hold the reference to the current instance and will assign these(color, type, legs, etc) instance attributes to it.

|  |
| --- |

* Class variable identity will be common to all the objects and can be accessed either through the object or class name itself.

|  |
| --- |

## What is a Method:

* Methods are functionalities that an entity can perform, so for an Animal entity when we talk about the behavior of an Animal or the functions it can perform then that will actually perform the methods of this Animal.
* Let's say we are defining three functionalities of an Animal it can perform, it can run, can eat, can hunt, sleep, etc. So these will actually get implemented in CLASS called Animal and will be termed as methods.
* These are actually special functions that are associated with CLASS Animal and they basically responsible for depicting or implementing the behavior of Entity Animal.
* These methods are also known as Instance Methods because they are also associated with instances.
* So if we have an object DOG then eat, sleep, hunt, etc functionality will be associated with Cats, and again CAT object will have its own associations.

### **Instance Methods Definition:**

* First parameter it takes is self and it allows 0 or more parameters apart from self.
* Whatever functionality your definition of a particular class then you have to have the first param as self.
* All parameters specified need to get passed as arguments in the correct order when they are called for execution.

### **Methods can be used to add new instance attributes:**

* Let’s say an Animal has attributes like group, legs\_count, color, etc.  
  And then you have eaten functionality which allows taking eatable\_names, quantity, and type. So we can define these attributes within methods as self. eatables, self.type, self. quantity, etc.

|  |
| --- |

## What are constructors:

* Constructor is a special function that is called implicitly when the object of the class is created.
* When you are creating an object of class Animal then your constructor which is a special method referred to by \_\_init\_\_ gets called.
* Cat = Animal(‘black’, 4, ‘mammals’) will call \_\_init\_\_() method implicitly and will create an object referred to by a cat.
* This implicitly \_\_init\_\_() calling will actually be responsible for the creation of an object.
* So a constructor is used to create and initialize the object and attributes.
* The first parameter in the constructor is always ‘self’ that holds the instance for which the constructor is called.
* Self is a must-have parameter when you are dealing with constructors.

### **Constructors Implementation:**

|  |
| --- |