

Object-Oriented Programming(OOP)

Introduction to Object-oriented Programming(OOP)

- OOP is programming paradigm that involves designing the solution using objects where attributed and behaviour are defined.
- The OOP approach to programming well-suited for large, complex programs and actively updated or maintained.
- The OOP is used in many applications, such as mobile application development, neural networking and parallel programming.
- The basic building blocks of OOP are
 1. Class
 2. Objects
 3. Attributes
 4. Methods

Pillars of OOP

- An OOP processes several additional characteristics: design principles that help to write a clean object oriented code. The four pillars of OOP are
 1. Inheritance
 2. Abstraction
 3. Encapsulation
 4. Polymorphism

Classes and Objects

- A class is a user defined blueprint for the objects and hold attributed and methods.
- An object is called an instance of a class.

Defining a Class

Creating a class is regarded as creating a new data type where its object belongs to this data type.

Creating a class does not mean allocation the memory space needed for the execution of the program.


Class is defined using the keyword “class”

Illustrating the class creation by creating a class named Biological Family without definition

- `>>> #Creating a class called BiologicalFamily`
- `>>> class BiologicalFamily:`
- `... pass`
- `...`
- `>>>`
- In the mentioned above, a class with “BiologicalNames” is declared, but instead of a functional block of the class, the “pass” keyword is used.

Object Creation

- The objects that are instances of a class belong to its data type.
- There are two types of object creation in python: without parameters and with parameters.
- The syntax for creating an object for a defined class without parameters in python is as follows:

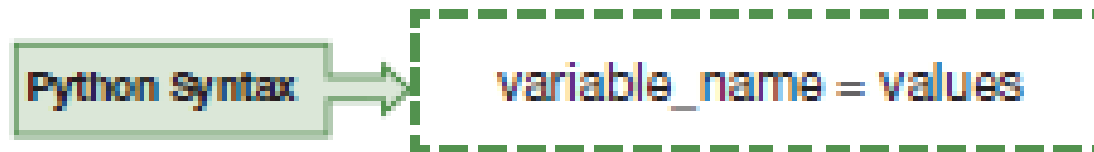


The diagram illustrates the Python syntax for creating an object without parameters. It features a light green rectangular box on the left containing the text "Python Syntax". A green arrow points from this box to a dashed green rectangular box on the right. Inside the dashed box is the code `object_name = ClassName()`, where `object_name` is in blue, `=` is in red, `ClassName` is in blue, and `()` is in blue.

```
Python Syntax → object_name = ClassName()
```

Class Attributes

- Class attributes are the variable defines inside a class, and all of its objects share the defined values for the objects inside the class.
- The values stored in the attributed are defined as the object's state.
- Class attributes are usually defined inside the class right after the first statement of the class declaration.
- Defining an attributes is an simple as a variable declaration. Th syntax for declaring an attribute inside the class statement is as follows:



Class Attributes

- Defined inside the class without the use of an object.
- Class attributed are shared across all objects of the class.
- Class attributed are accessed using a class name and an object with dot notation. E.g.,
class.Name or
object.class_attribute

Instance Variable

- Defined inside a constructor or many methods using the self Keyword.
- Instance variables are specific to some objects/
- Instance variables are accessed using object dot notation.

Types of Attributes

- Based on accessibility, class attributes are divided into two categories.

1. Public attributes

2. Private attributes

| Scope | Accessibility in | | |
|---------|------------------|-------------------|------------------------------|
| | Within Class | Within Subclasses | Outside Class and Subclasses |
| Public | ✓ | ✓ | ✓ |
| Private | ✓ | ✗ | ✗ |

Public attributes: The public scope of an attributes is the default scope in python. The kind of attributes declared, accessed, and modified.

Private variables: The private scope of the class attributes allows access to the class attributes only inside the class. The syntax for defining private members inside a class definition is as follows.

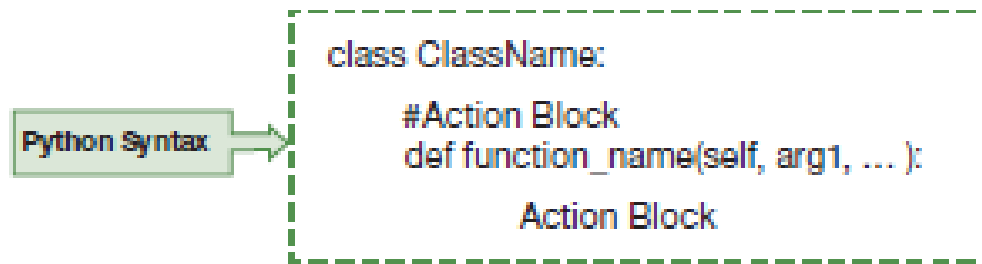
Python Syntax → `__class_variable = assigning_value`

Methods in Class

- Methods define the behaviour of the objects. Methods and functions seem to be the same but they are different in a few aspects and they are tabulated below.

| Methods | Functions |
|--|--|
| Methods are always defined inside a class. | The function does not need any class to be defined. |
| Methods are always associated with the objects of the belonging class. | Functions are not associated with any objects. |
| Invoking methods need the name of the respective objects. | The name of a function is enough to invoke. |
| Methods can operate on the data which the objects hold. | Functions can operate on the parameters that passed to it. |
| Methods require to have 'self' as their first argument | Functions do not need any 'self' argument. |

- Methods in the class are defined using the following syntax



A green box labeled "Python Syntax" has an arrow pointing to a dashed green box containing the following code:

```
class ClassName:  
    #Action Block  
    def function_name(self, arg1, ... ):  
        Action Block
```

- The self keyword is the first argument in the parameter list and refers to the object of the class.
- The function (actually, it should be called method) is defined under a class.

Note

If the "self" keyword is not used in the argument of a method, one cannot use the data/class attributes/instance variables associated with the object inside the method.

The "self" keyword in the argument of the method denotes the corresponding object. Also, it is used to denote the object along with the "dot" operator for accessing the variables of the corresponding object.

The syntax for calling the methods using the object is,



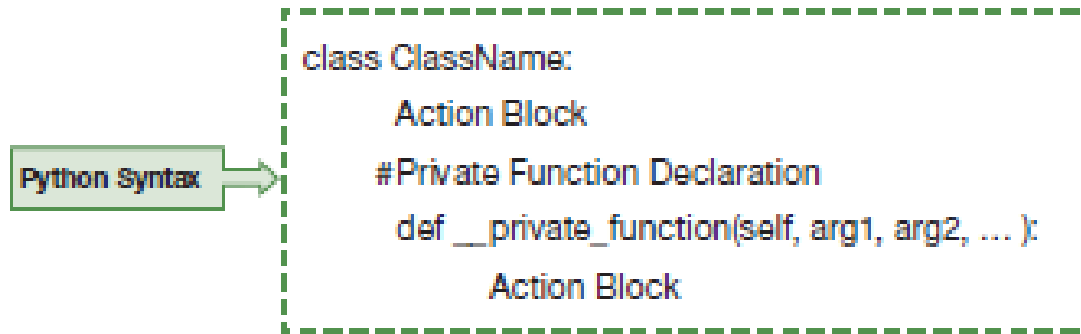
A green box labeled "Python Syntax" has an arrow pointing to a dashed green box containing the following code:

```
Object_name.method_name(arg1, arg 2, ...)
```

For invoking the method of an object, the name of the object is followed by the "dot" notation, the name of the method, and arguments inside the parathesis. The following example shows how to define a method and invoking.

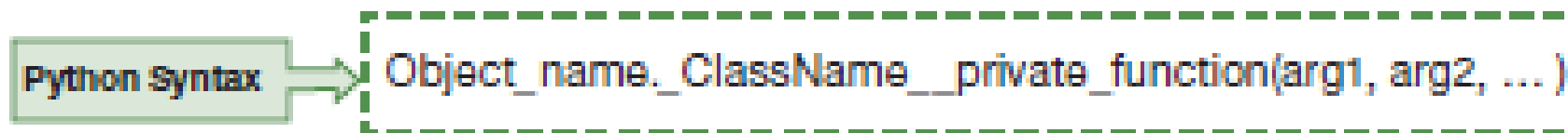
Private Methods

- With the private variables, private methods get access only within the class and are accompanied by “__” (double underscore) as the prefix.
- The syntax for defining a private function is as follows.



```
class ClassName:  
    Action Block  
    #Private Function Declaration  
    def __private_function(self, arg1, arg2, ... ):  
        Action Block
```

- The syntax for invoking private function from the outside the class is,



```
Object_name._ClassName__private_function(arg1, arg2, ... )
```

Constructor

- A constructor is a special method of a class that is generally used to instantiate an object.
- The constructor allocated the memory space for storing the class variables and executing the methods of the object of a class.
- One of the major tasks of the constructor is to initialise the values of the class variables(class attributes) when an object of the class is created.

Python Syntax

```
class ClassName:  
    ...  
    Action Block  
    ...  
    #Cosntructor of the Class  
    def __init__(self):  
    ...  
        #Action Block for Constructor  
    ...
```

Destructor

- Destructors are called when an object gets destroyed.
- In Python, destructors are not needed as much as in C++ since python has a garbage collector that handles memory management automatically.
- The `del__()` method is a destructor method in python.
- The syntax for deleting the object of the class is as follows:



```
Python Syntax → del(object_name)
```

Garbage Collection

- Garbage collection is a memory recovery feature that is used in many programming language to free-up memory space that has been allocated to the variables or objects which are no longer demanded by the program.
- In addition to the automatic garbage collector in python, a separate module is used to control the process of garbage collection.
- Manual Garbage collection is performed on the following basis.
 1. time-based garbage collection
 2. Event-based garbage collection

Built-ins

- When an object is created for a class, there are attributed and methods created along with it by default.

Built-in Methods in class

- Special methods in the class are a set of built-in methods to enrich the classes and are usually prefixed and suffixed with “__”(double score).
 1. `__repr__()`: `__repr__` method is used to represent the object of a class as a string.
 2. `__Str__()`: returns the string representation of the object of a class.

Built-in Attributes in Objects

| Attributes Name | Purpose of Usage |
|-------------------------|---|
| <code>__dict__</code> | Gives a dictionary containing the class's or object's namespace. |
| <code>__doc__</code> | Returns documentation of the object as the string. If there is no documentation, then "None" is returned. |
| <code>__class__</code> | Returns a string that holds the module of the program and the name of the class. |
| <code>__module__</code> | Gives the name of a module in which the class is defined. |
| <code>__bases__</code> | Return the base classes in order of their occurrence in the base class list. |