CMPE 103
OBJECT-ORIENTED PROGRAMMING

MODULE 4

INTRO TO OOP

What is Object?

Pen

Board

Laptop

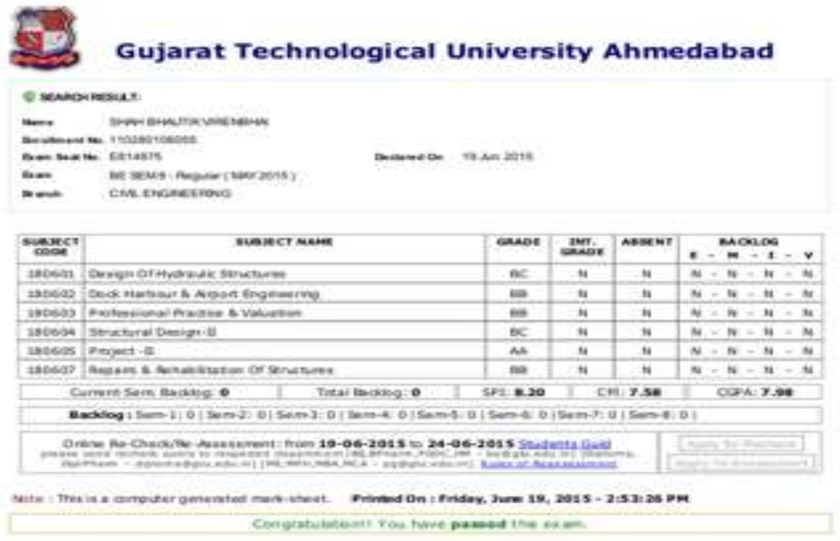
Bench

Student

Projector

Physical objects…

What is Object?



Gujarat Technological University Ahmedabad

SEARCH RESULT:

Name	SHAH BHAUTIK VIRENBHAI
Enrollment No.	110280106055
Exam Seat No.	E614575
Declared On	19 Jun 2015
Exam	BE SEM 8 - Regular (MAY 2015)
Branch	CIVIL ENGINEERING

SUBJECT CODE	SUBJECT NAME	GRADE	INT. GRADE	ABSENT	BACKLOG E - M - I - V
180601	Design Of Hydraulic Structures	BC	N	N	N - N - N - N
180602	Dock Harbour & Airport Engineering	BB	N	N	N - N - N - N
180603	Professional Practise & Valuation	BB	N	N	N - N - N - N
180604	Structural Design-II	BC	N	N	N - N - N - N
180605	Project -II	AA	N	N	N - N - N - N
180607	Repairs & Rehabilitation Of Structures	BB	N	N	N - N - N - N

Current Sem. Backlog: **0**	Total Backlog: **0**	SPI: **8.20**	CPI: **7.58**	CGPA: **7.98**

Backlog : Sem-1: 0 | Sem-2: 0 | Sem-3: 0 | Sem-4: 0 | Sem-5: 0 | Sem-6: 0 | Sem-7: 0 | Sem-8: 0 |

Online Re-Check/Re-Assessment : from **19-06-2015** to **24-06-2015** Students Guid
please send recheck query to respective department [illegible]
[illegible] Rules of Reassessment

Note : This is a computer generated mark-sheet. **Printed On : Friday, June 19, 2015 - 2:53:26 PM**

Congratulation!! You have **passed** the exam.

Result

Account

Bank Account

Logical objects...

What is a Class and Objects in Python?

•**Class**: The class is a user-defined data structure that binds the data members and methods into a single unit. Class is a **blueprint or code template for object creation**. Using a class, you can create as many objects as you want.

•**Object**: An **object is an instance of a class**. It is a collection of attributes (variables) and methods. We use the object of a class to perform actions.

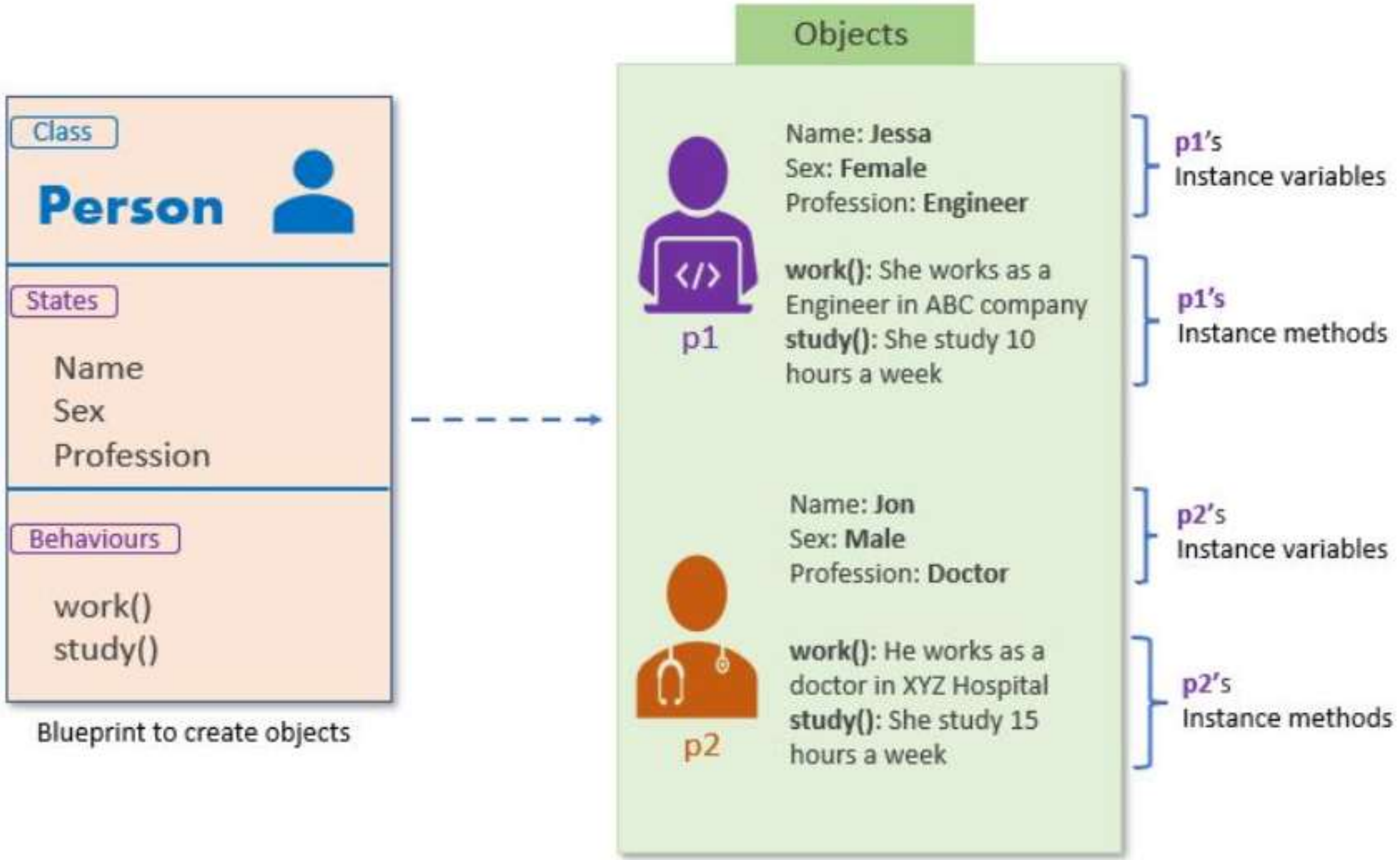
In short, Every **object** has the following property.

•**Identity**: Every object must be uniquely identified.

•**State**: An object has an attribute that represents a state of an object, and it also reflects the property of an object.

•**Behavior**: An object has methods that represent its behavior.

Python is an ***Object-Oriented Programming language***, so everything in ***Python is treated as an object***. An object is a real-life entity. It is the collection of various data and functions that operate on those data.



Blueprint to create objects

Attributes and operations

Attributes:
Name
Age
Weight

Operations:
Eat
Sleep
Walk

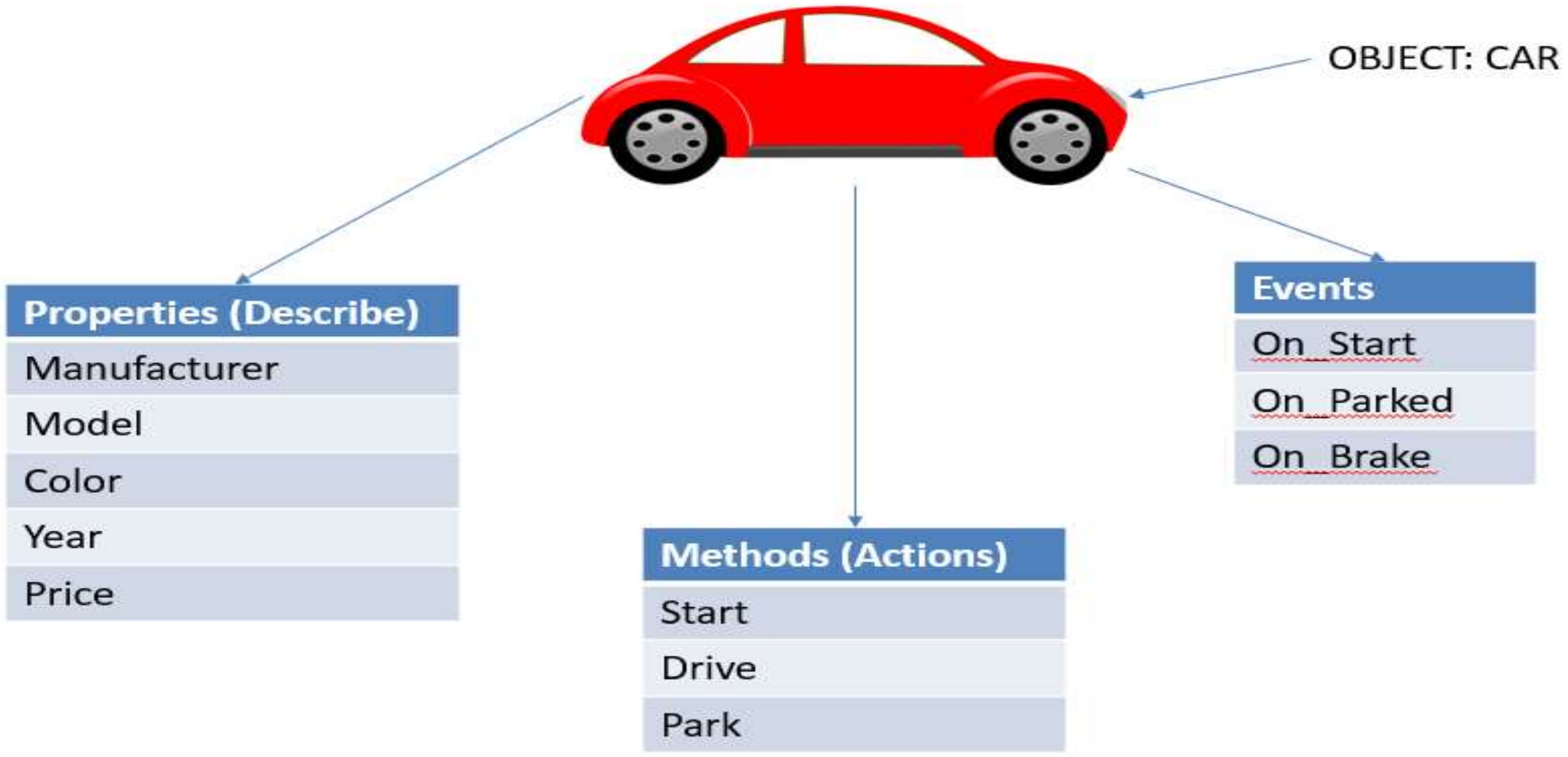
Attributes:
Company
Model
Weight

Operations:
Drive
Stop
FillFuel

Attributes:
AccountNo
HolderName
Balance

Operations:
Deposit
Withdraw
Transfer

What is Object ?



Properties (Describe)
Manufacturer
Model
Color
Year
Price

Methods (Actions)
Start
Drive
Park

Events
On_Start
On_Parked
On_Brake

Class

Pattern of pen

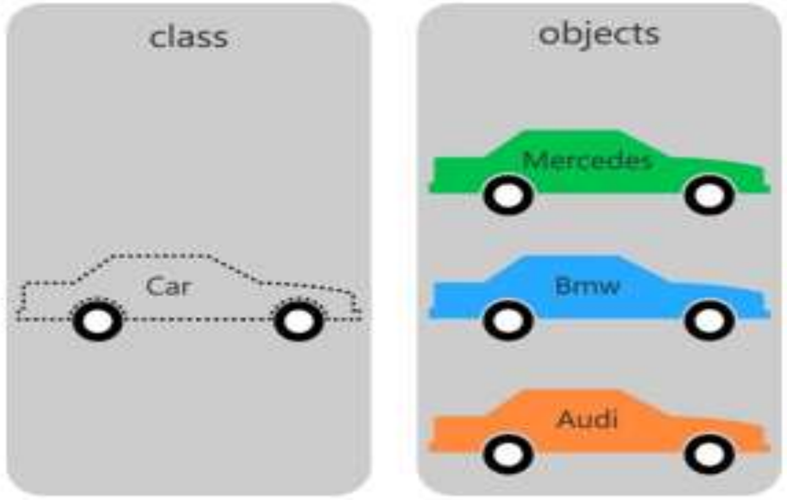
Constructor

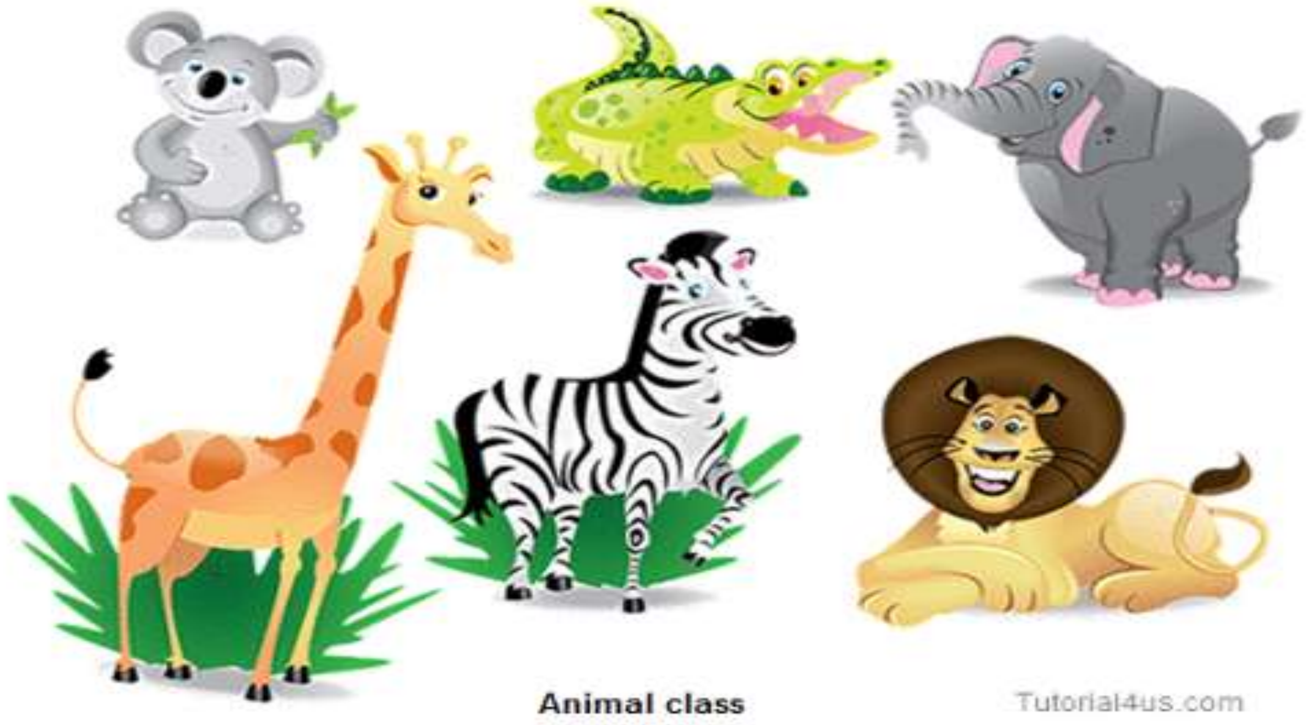
Machine which takes list of actions to produce pen

Object

Creates many pen using pattern and machine

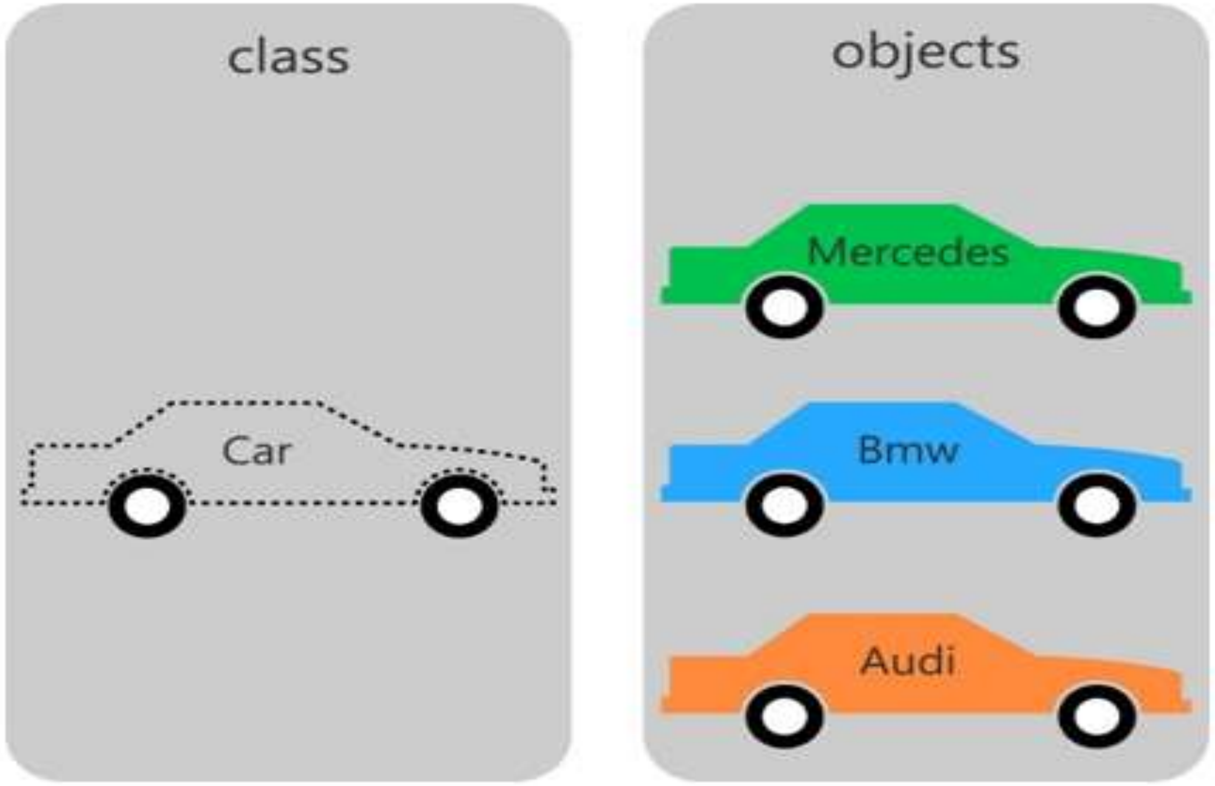
Classes…

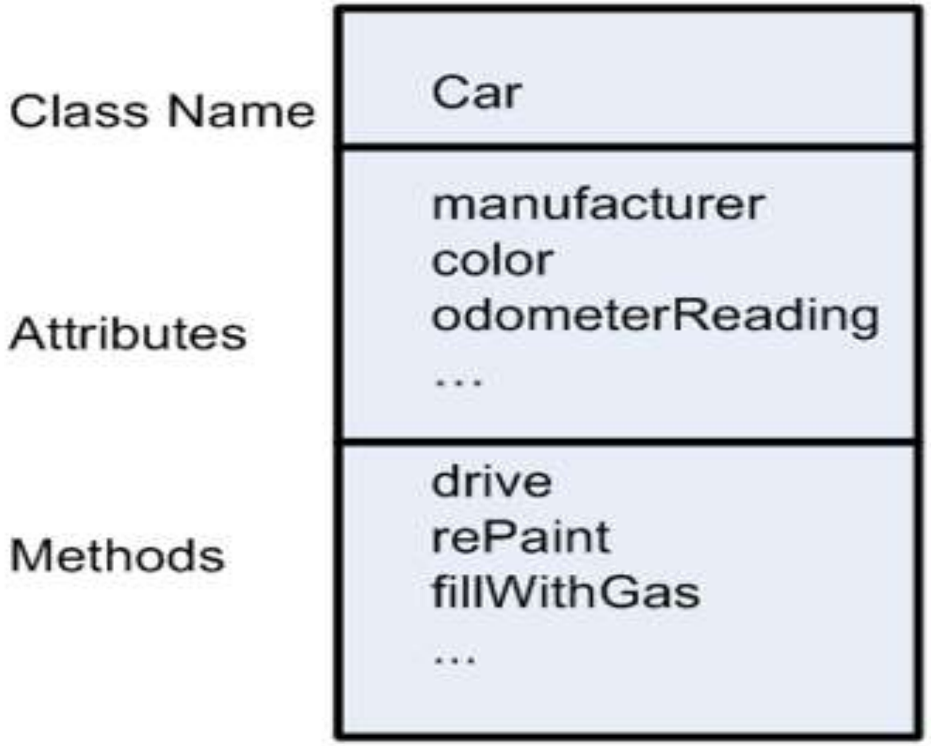




Class: Blueprint (template) for object.
Object: Instance of class.

Class


class
Car
objects
Mercedes
Bmw
Audi


Class Name
Attributes
Methods
Car
manufacturer
color
odometerReading
...
drive
rePaint
fillWithGas
...

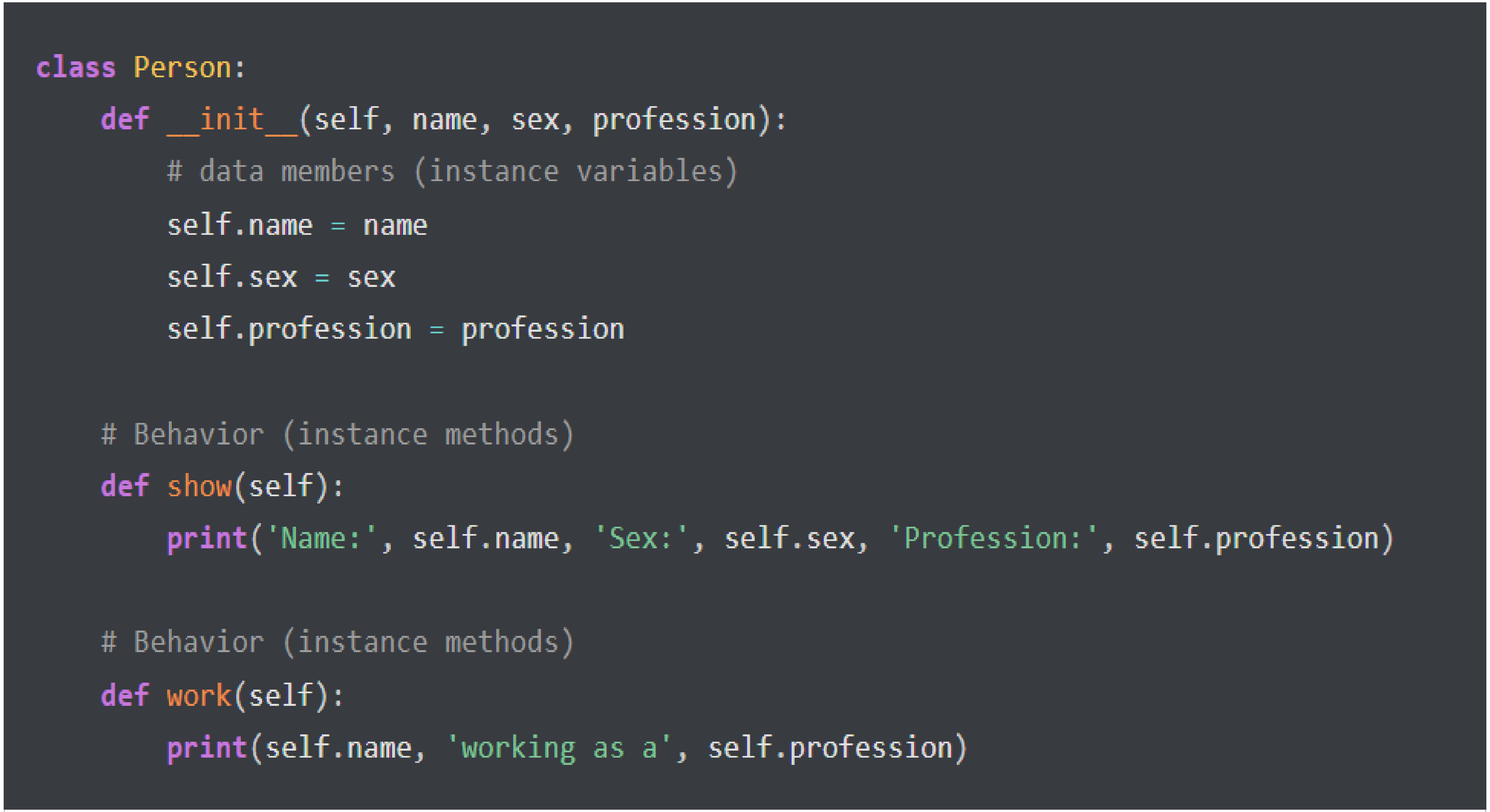
Create a Class in Python

In Python, class is defined by using the `class` keyword. The syntax to create a class is given below.

Syntax:

```
class class_name:
    '''This is a docstring. I have created a new class'''
    <statement 1>
    <statement 2>
    .
    .
    <statement N>
```

- `class_name` : It is the name of the class
- `Docstring` : It is the first string inside the class and has a brief description of the class. Although not mandatory, this is highly recommended.
- `statements` : Attributes and methods



```python
class Person:
    def __init__(self, name, sex, profession):
        # data members (instance variables)
        self.name = name
        self.sex = sex
        self.profession = profession

    # Behavior (instance methods)
    def show(self):
        print('Name:', self.name, 'Sex:', self.sex, 'Profession:', self.profession)

    # Behavior (instance methods)
    def work(self):
        print(self.name, 'working as a', self.profession)
```

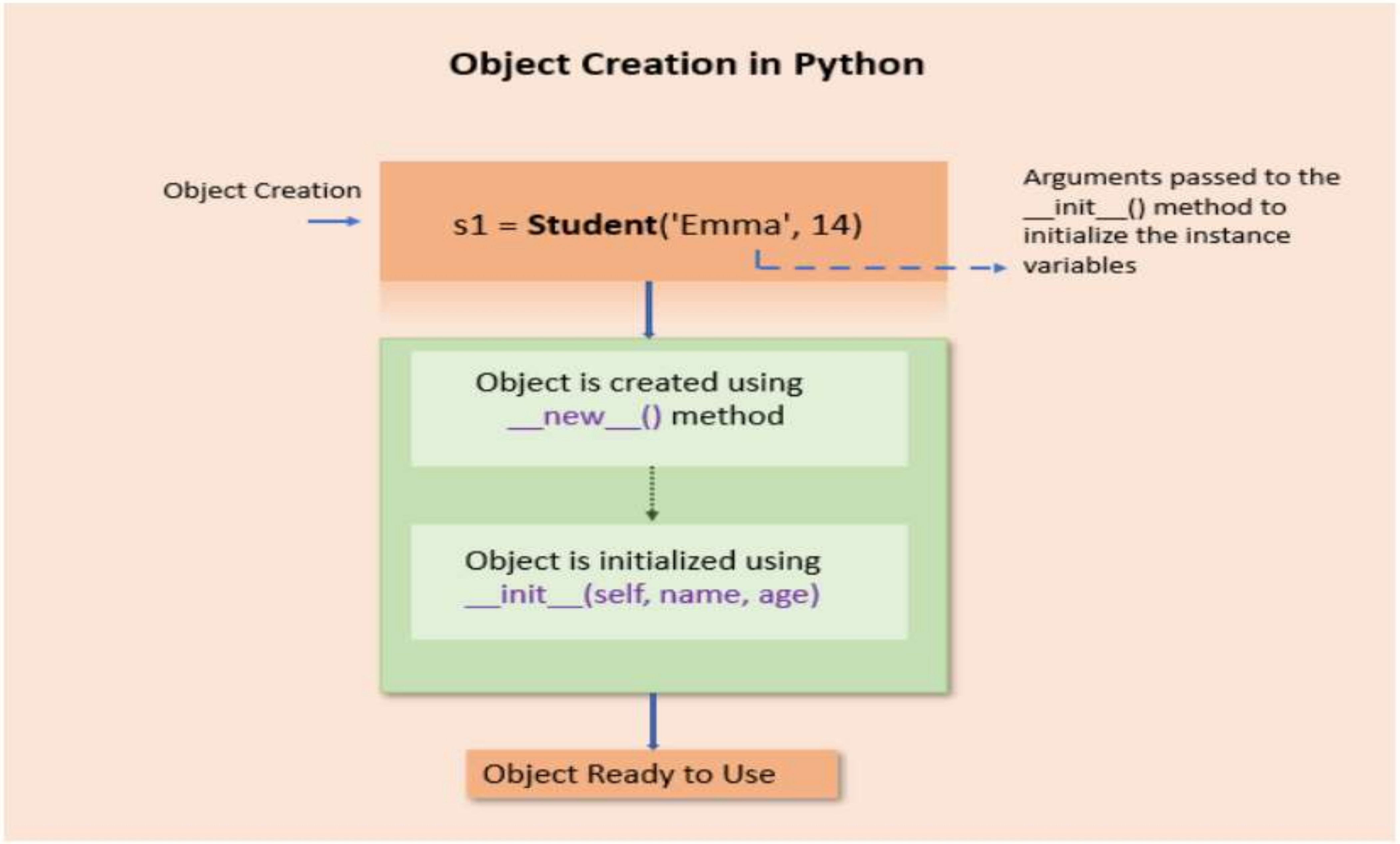
Create Object of a Class

An object is essential to work with the class attributes. The object is created using the class name. When we create an object of the class, it is called instantiation. The object is also called the instance of a class.

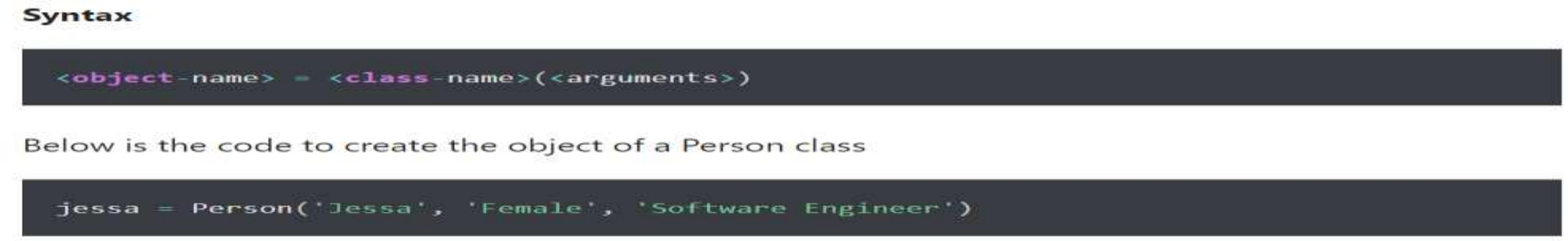
A constructor is a special method used to create and initialize an object of a class. This method is defined in the class.

In Python, Object creation is divided into two parts in **Object Creation** and **Object initialization**

- Internally, the `__new__` is the method that creates the object
- And, using the `__init__()` method we can implement constructor to initialize the object.


Object Creation in Python
Object Creation
s1 = Student('Emma', 14)
Arguments passed to the __init__() method to initialize the instance variables
Object is created using __new__() method
Object is initialized using __init__(self, name, age)
Object Ready to Use

Create an object in Python using a constructor



Syntax

```
<object-name> = <class-name>(<arguments>)
```

Below is the code to create the object of a Person class

```
jessa = Person('Jessa', 'Female', 'Software Engineer')
```

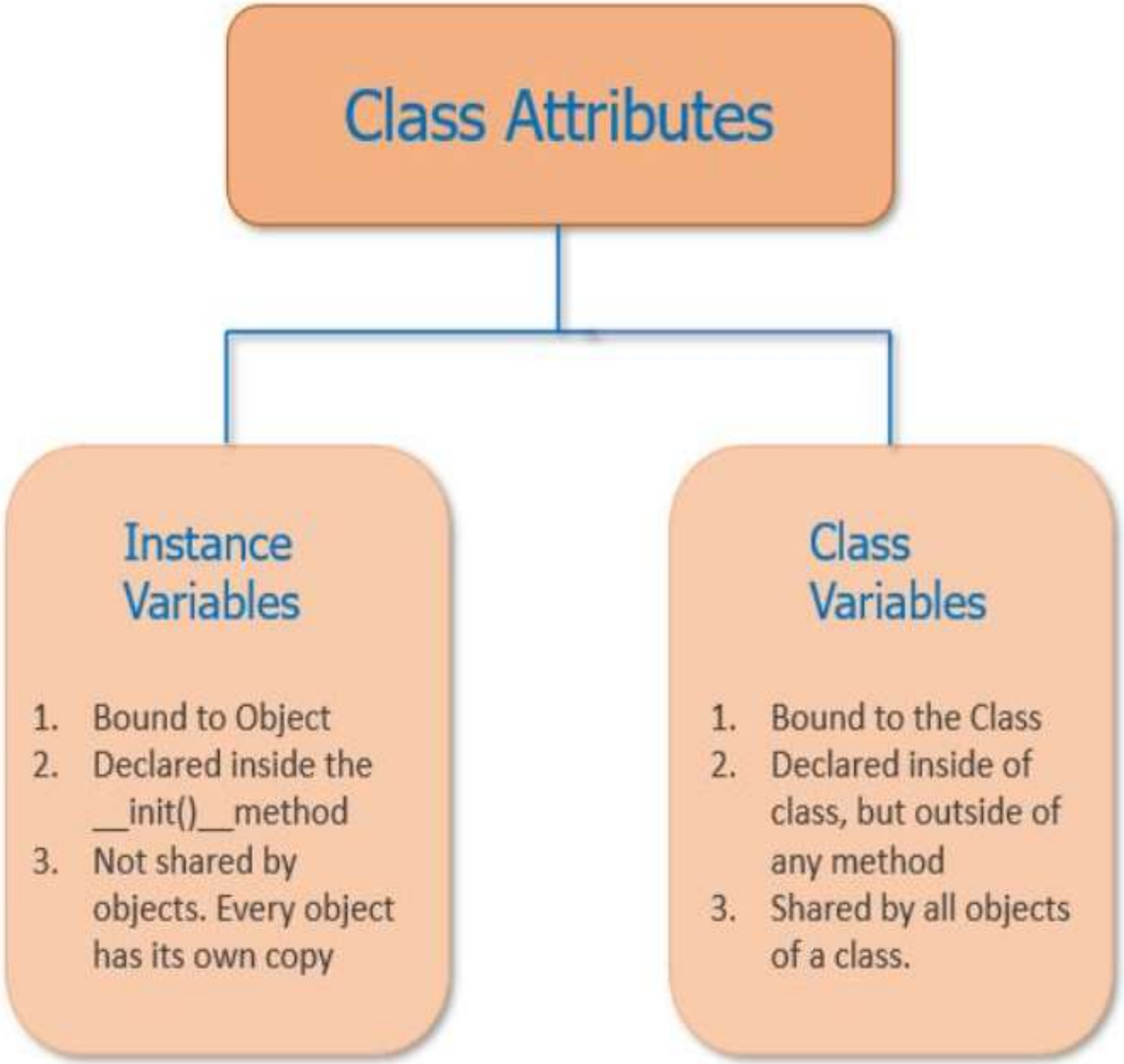
```
class Person:
    def __init__(self, name, sex, profession):
        # data members (instance variables)
        self.name = name
        self.sex = sex
        self.profession = profession

    # Behavior (instance methods)
    def show(self):
        print('Name:', self.name, 'Sex:', self.sex, 'Profession:', self.profession)

    # Behavior (instance methods)
    def work(self):
        print(self.name, 'working as a', self.profession)

# create object of a class
jessa = Person('Jessa', 'Female', 'Software Engineer')

# call methods
jessa.show()
jessa.work()
```

Class Attributes

When we design a class, we use instance variables and class variables.

In Class, attributes can be defined into two parts:

- **Instance variables**: The instance variables are attributes attached to an instance of a class. We define instance variables in the constructor (the `__init__()` method of a class).
- **Class Variables**: A class variable is a variable that is declared inside of class, but outside of any instance method or `__init__()` method.


Class Attributes
Instance Variables
1. Bound to Object
2. Declared inside the __init()__method
3. Not shared by objects. Every object has its own copy
Class Variables
1. Bound to the Class
2. Declared inside of class, but outside of any method
3. Shared by all objects of a class.

Class Attributes in Python

Objects do not share instance attributes. Instead, *every object has its copy of the instance attribute and is unique to each object.*

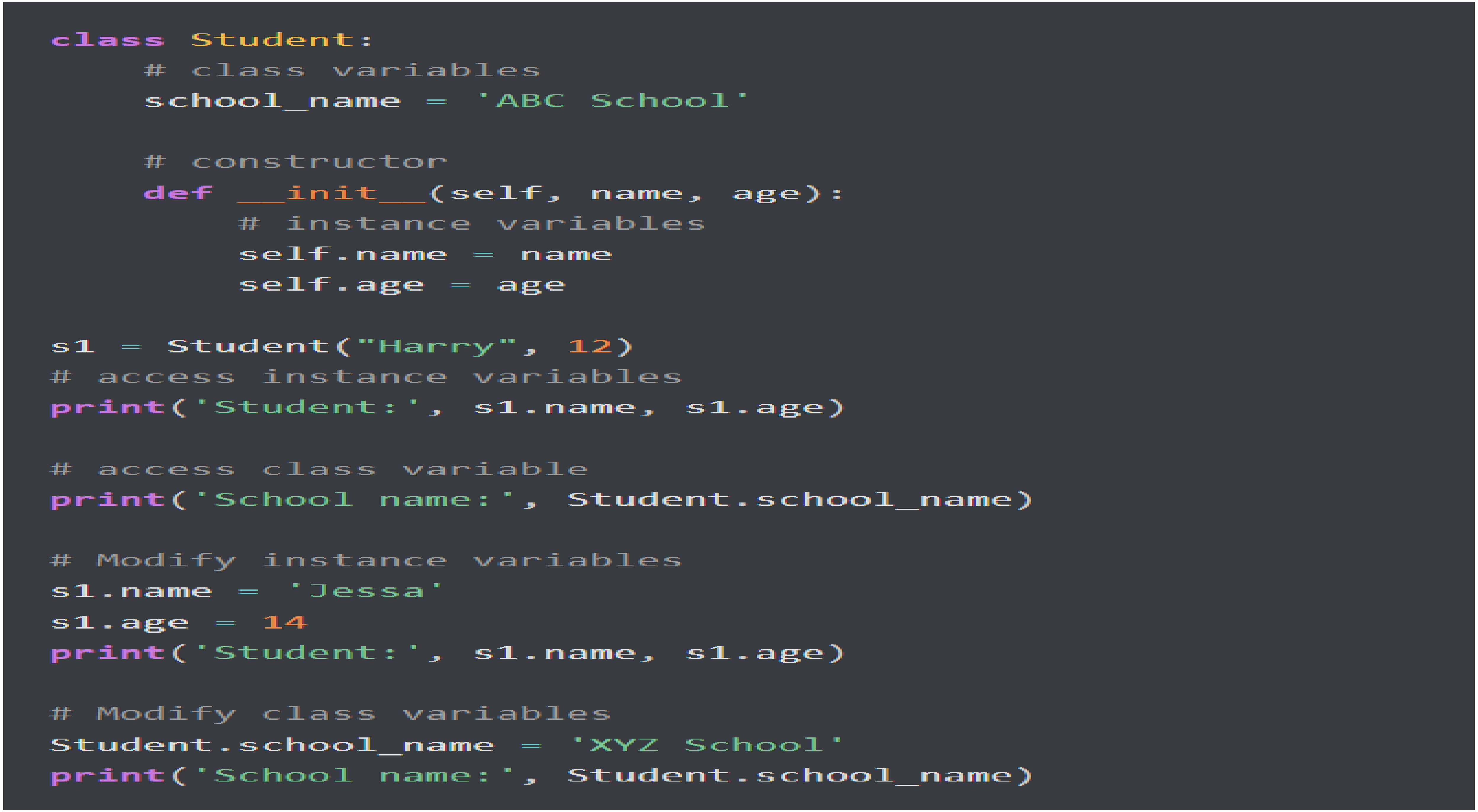
All instances of a class share the class variables. However, unlike instance variables, the value of a class variable is not varied from object to object.

Only one copy of the static variable will be created and shared between all objects of the class.

Accessing properties and assigning values

•An instance attribute can be accessed or modified by using the dot notation: ***instance_name.attribute_name*****.**

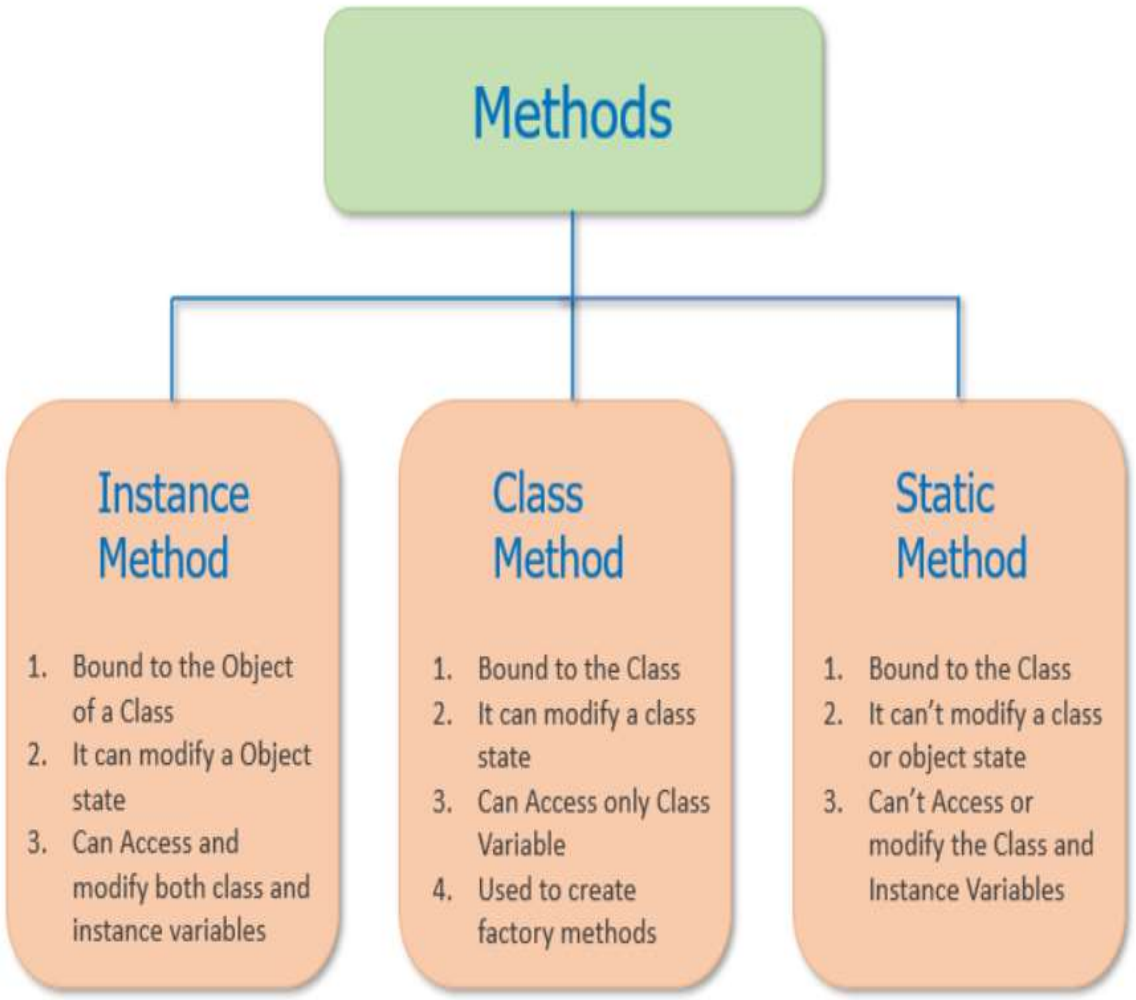
•A class variable is accessed or modified using the class name



```python
class Student:
    # class variables
    school_name = 'ABC School'

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

s1 = Student("Harry", 12)
# access instance variables
print('Student:', s1.name, s1.age)

# access class variable
print('School name:', Student.school_name)

# Modify instance variables
s1.name = 'Jessa'
s1.age = 14
print('Student:', s1.name, s1.age)

# Modify class variables
Student.school_name = 'XYZ School'
print('School name:', Student.school_name)
```

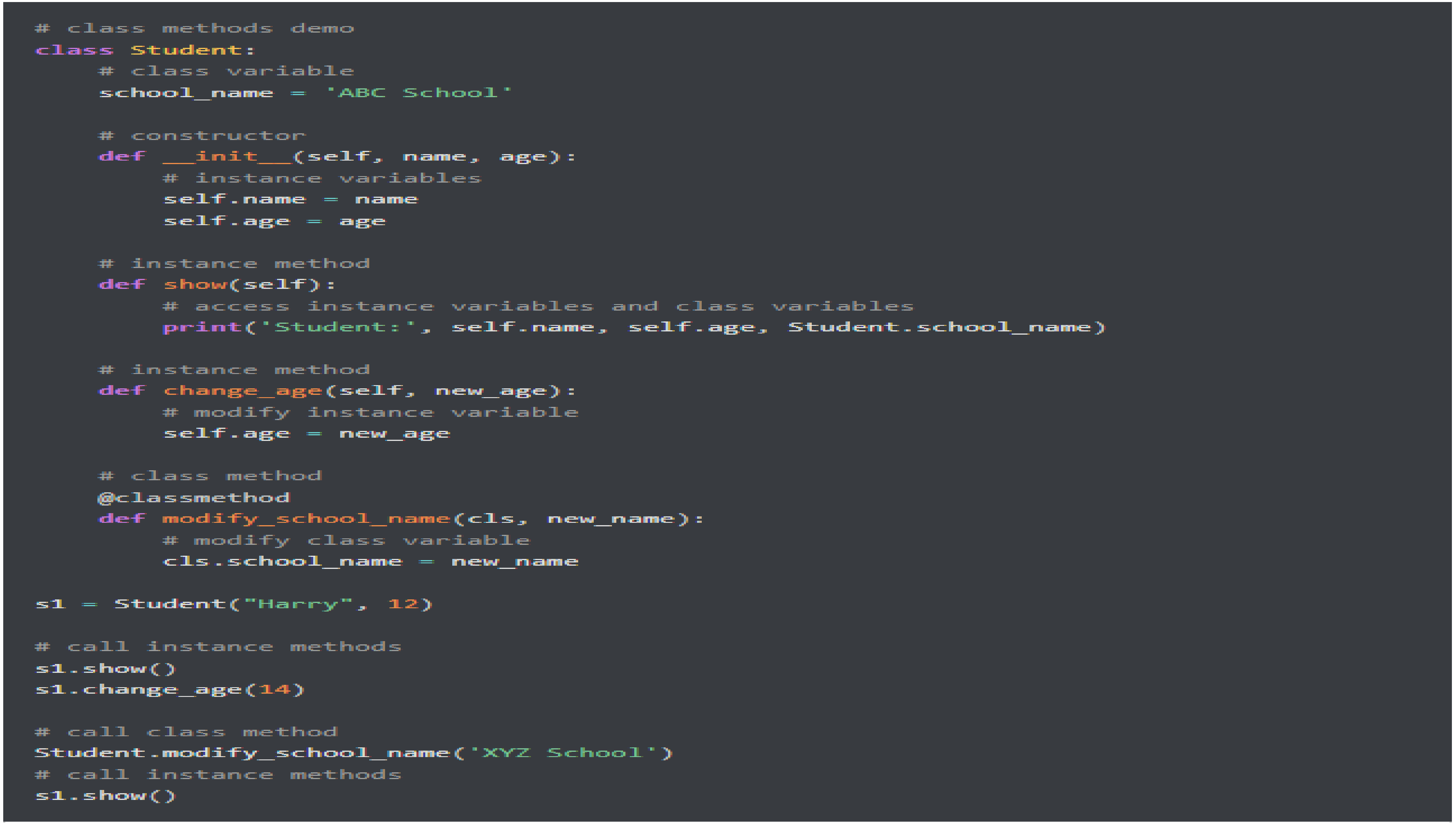
Class Methods

In Object-oriented programming, Inside a Class, we can define the following three types of methods.


Methods
Instance Method
1. Bound to the Object of a Class
2. It can modify a Object state
3. Can Access and modify both class and instance variables
Class Method
1. Bound to the Class
2. It can modify a class state
3. Can Access only Class Variable
4. Used to create factory methods
Static Method
1. Bound to the Class
2. It can't modify a class or object state
3. Can't Access or modify the Class and Instance Variables

class method vs static method vs instance method

- **Instance method**: Used to access or modify the object state. If we use instance variables inside a method, such methods are called instance methods.
- **Class method**: Used to access or modify the class state. In method implementation, if we use only class variables, then such type of methods we should declare as a class method.
- **Static method**: It is a general utility method that performs a task in isolation. Inside this method, we don't use instance or class variable because this static method doesn't have access to the class attributes.



```python
# class methods demo
class Student:
    # class variable
    school_name = 'ABC School'

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

    # instance method
    def show(self):
        # access instance variables and class variables
        print('Student:', self.name, self.age, Student.school_name)

    # instance method
    def change_age(self, new_age):
        # modify instance variable
        self.age = new_age

    # class method
    @classmethod
    def modify_school_name(cls, new_name):
        # modify class variable
        cls.school_name = new_name

s1 = Student("Harry", 12)

# call instance methods
s1.show()
s1.change_age(14)

# call class method
Student.modify_school_name('XYZ School')
# call instance methods
s1.show()
```

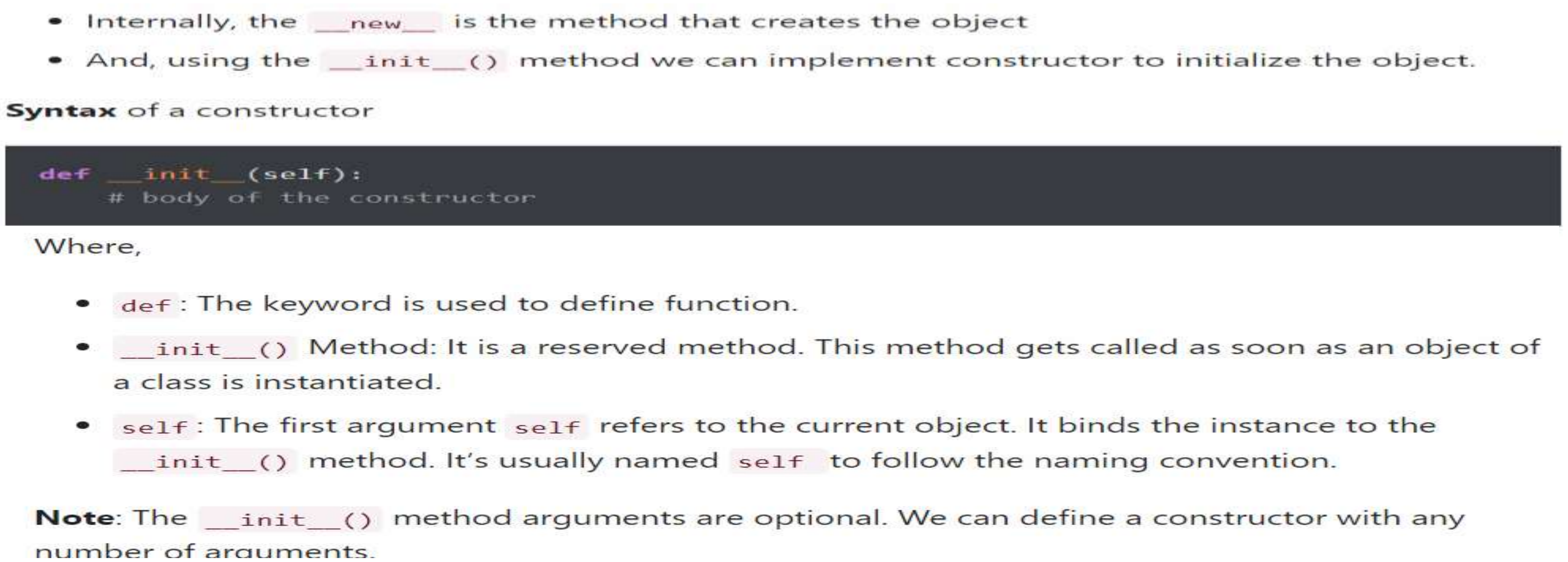
What is Constructor in Python?

In object-oriented programming, **A constructor is a special method used to create and initialize an object of a class**. This method is defined in the class.

•The constructor is executed automatically at the time of object creation.

•The primary use of a constructor is to declare and initialize data member/ instance variables of a class. The constructor contains a collection of statements (i.e., instructions) that executes at the time of object creation to initialize the attributes of an object.

In Python, Object creation is divided into two parts in **Object Creation** and **Object initialization**



- Internally, the `__new__` is the method that creates the object
- And, using the `__init__()` method we can implement constructor to initialize the object.

Syntax of a constructor

```
def __init__(self):
    # body of the constructor
```

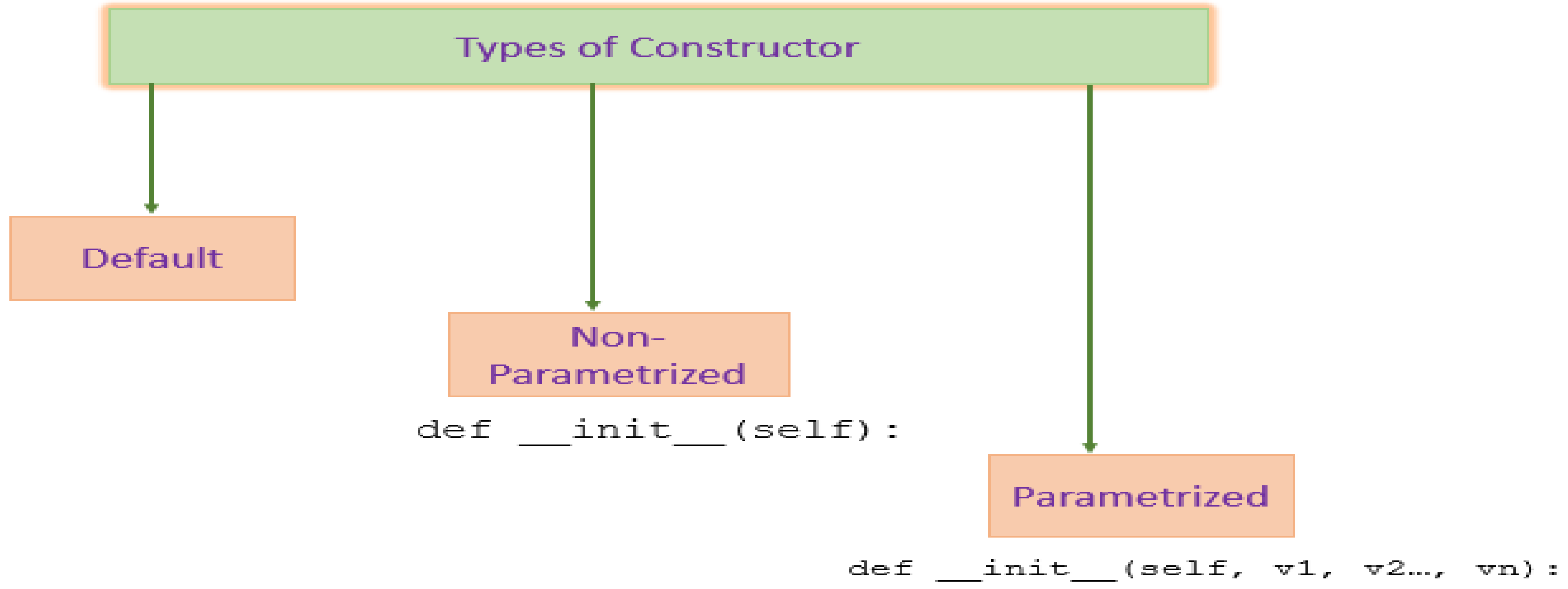
Where,

- `def` : The keyword is used to define function.
- `__init__()` Method: It is a reserved method. This method gets called as soon as an object of a class is instantiated.
- `self` : The first argument `self` refers to the current object. It binds the instance to the `__init__()` method. It's usually named `self` to follow the naming convention.

Note: The `__init__()` method arguments are optional. We can define a constructor with any number of arguments.

In Python, we have the following three types of constructors.

- Default Constructor
- Non-parametrized constructor
- Parameterized constructor



```
def __init__(self):
```

```
def __init__(self, v1, v2…, vn):
```

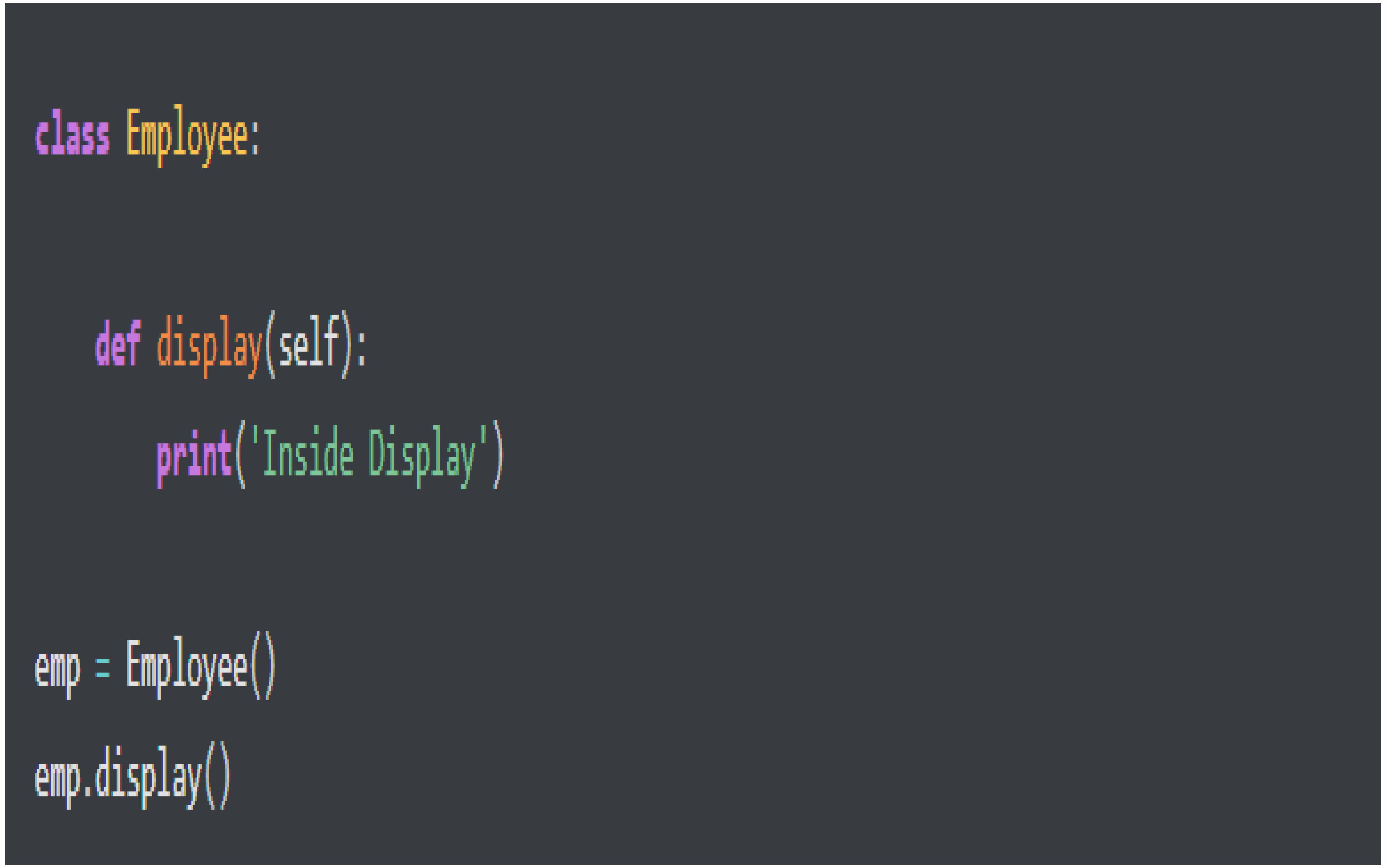
Types of constructor

Default Constructor

Python will provide a default constructor if no constructor is defined. Python adds a default constructor when we do not include the constructor in the class or forget to declare it. It does not perform any task but initializes the objects. It is an empty constructor without a body.

If you do not implement any constructor in your class or forget to declare it, the Python inserts a default constructor into your code on your behalf. This constructor is known as the default constructor. It does not perform any task but initializes the objects. It is an empty

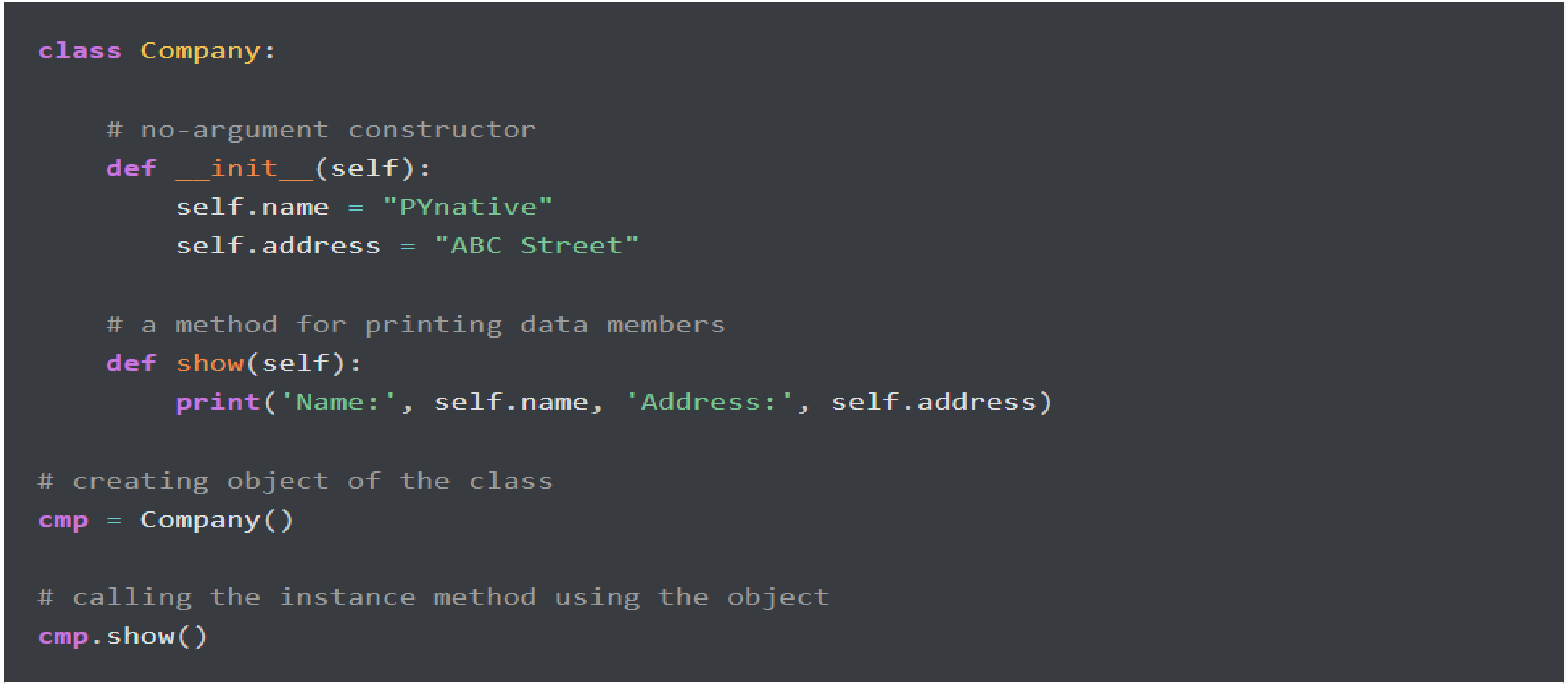
Example:



```
class Employee:

    def display(self):
        print('Inside Display')

emp = Employee()
emp.display()
```

Non-Parametrized Constructor

A constructor without any arguments is called a non-parameterized constructor. This type of constructor is used to initialize each object with default values. This constructor doesn't accept the arguments during object creation. Instead, it initializes every object with the same set of values.

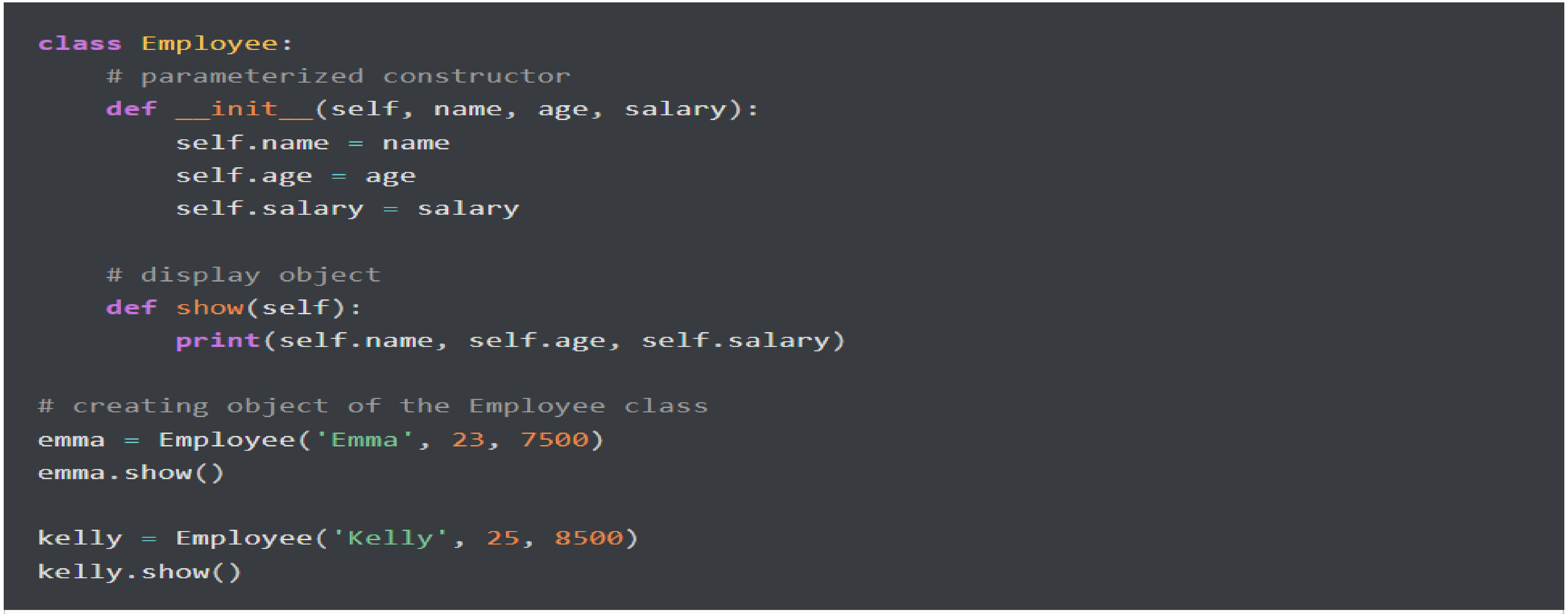


```python
class Company:

    # no-argument constructor
    def __init__(self):
        self.name = "PYnative"
        self.address = "ABC Street"

    # a method for printing data members
    def show(self):
        print('Name:', self.name, 'Address:', self.address)

# creating object of the class
cmp = Company()

# calling the instance method using the object
cmp.show()
```

Parameterized Constructor

A constructor with defined parameters or arguments is called a parameterized constructor. We can pass different values to each object at the time of creation using a parameterized constructor.

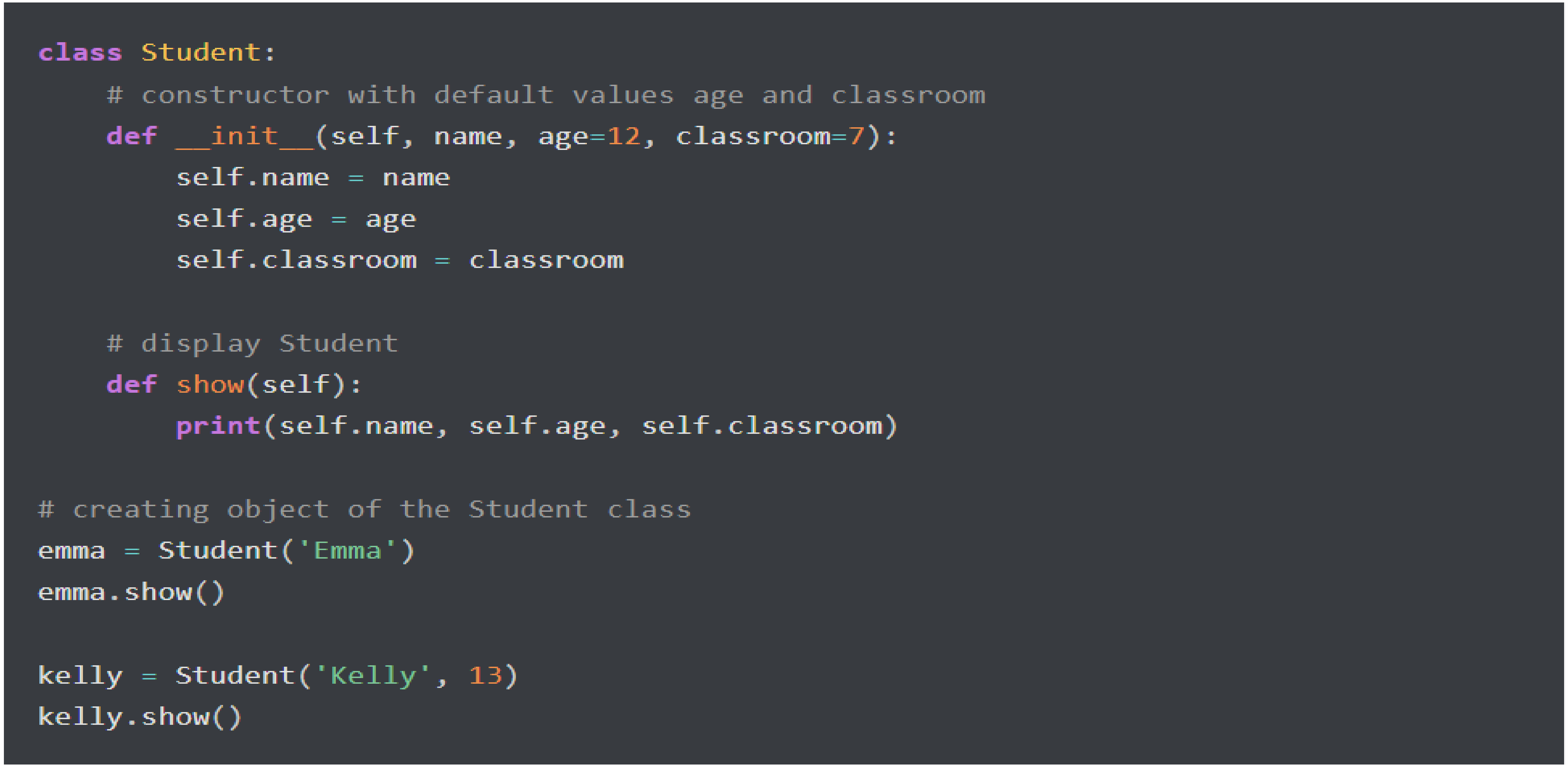
The first parameter to constructor is `self` that is a reference to the being constructed, and the rest of the arguments are provided by the programmer. A parameterized constructor can have any number of arguments.



```python
class Employee:
    # parameterized constructor
    def __init__(self, name, age, salary):
        self.name = name
        self.age = age
        self.salary = salary

    # display object
    def show(self):
        print(self.name, self.age, self.salary)

# creating object of the Employee class
emma = Employee('Emma', 23, 7500)
emma.show()

kelly = Employee('Kelly', 25, 8500)
kelly.show()
```

Constructor With Default Values

Python allows us to define a constructor with default values. The default value will be used if we do not pass arguments to the constructor at the time of object creation.



```python
class Student:
    # constructor with default values age and classroom
    def __init__(self, name, age=12, classroom=7):
        self.name = name
        self.age = age
        self.classroom = classroom

    # display Student
    def show(self):
        print(self.name, self.age, self.classroom)

# creating object of the Student class
emma = Student('Emma')
emma.show()

kelly = Student('Kelly', 13)
kelly.show()
```

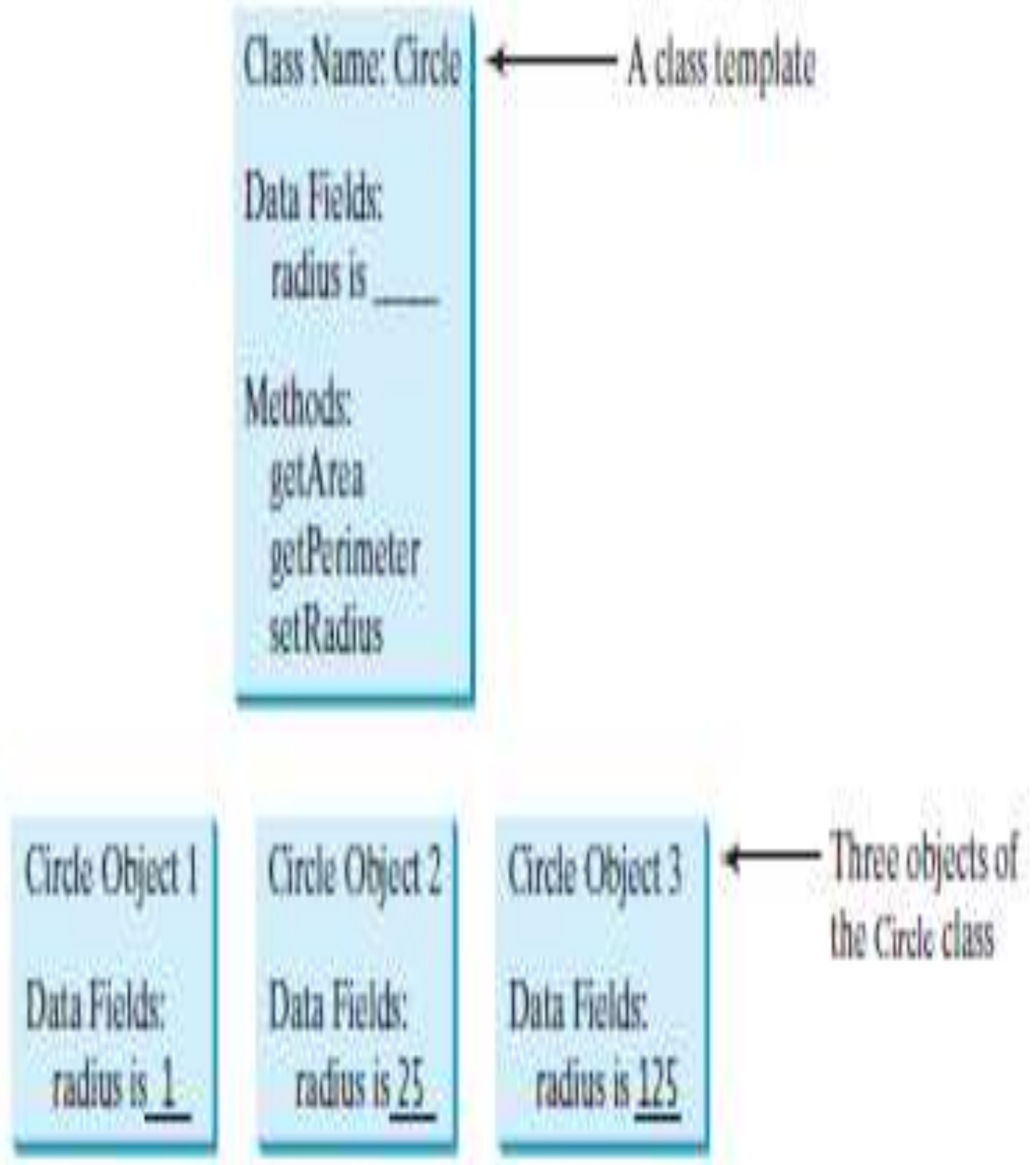
Self Keyword in Python

The first argument `self` refers to the current object.

Whenever we call an instance method through an object, the Python compiler implicitly passes object reference as the first argument commonly known as self.

It is not mandatory to name the first parameter as a `self`. We can give any name whatever we like, but it has to be the first parameter of an instance method.

```python
class Student:
    # constructor
    def __init__(self, name, age):
        self.name = name
        self.age = age

    # self points to the current object
    def show(self):
        # access instance variable using self
        print(self.name, self.age)

# creating first object
emma = Student('Emma', 12)
emma.show()

# creating Second object
kelly = Student('Kelly', 13)
kelly.show()
```



Circle.py

```
import math

class Circle:
      # Construct a circle object
  def _ _init_ _(self, radius = 1):
        self.radius = radius

  def getPerimeter(self):
        return 2 * self.radius * math.pi

      def getArea(self):
        return self.radius * self.radius * 
math.pi

      def setRadius(self, radius):
        self.radius = radius
```

TestCircle.py

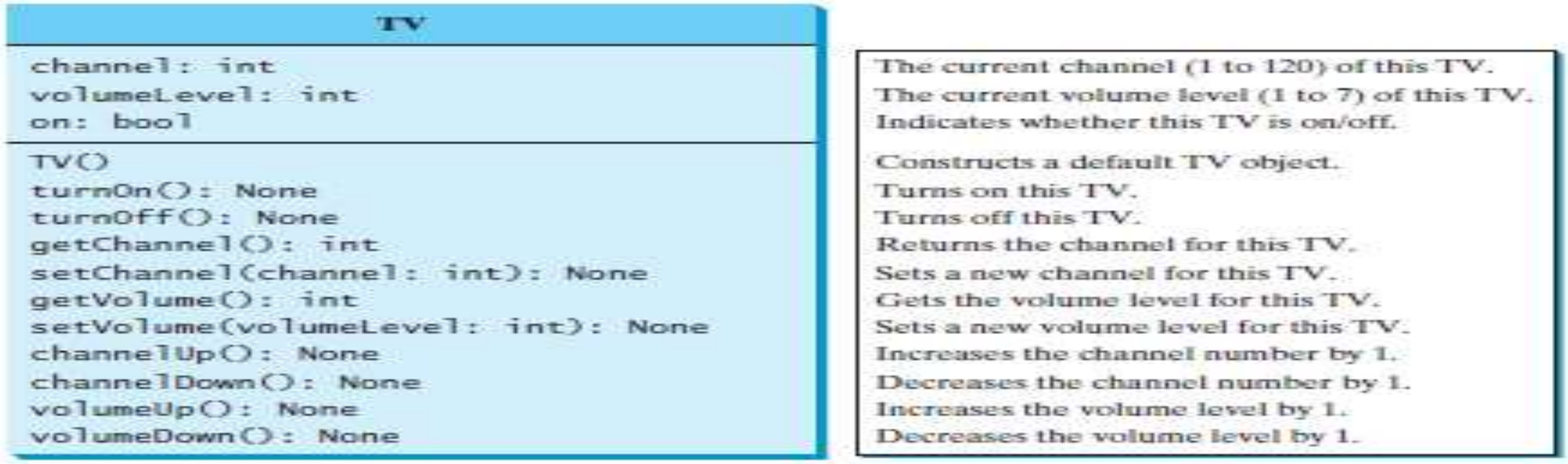
```
from Circle import Circle

def main():
         # Create a circle with radius 1
         circle1 = Circle()
         print("The area of the circle of radius", circle1.radius,"is",circle1.getArea())


         # Create a circle with radius 25
         circle2 = Circle(25)
         print("The area of the circle of radius", circle2.radius, "is",circle2.getArea())


        # Create a circle with radius 125
        circle3 = Circle(125)
        print("The area of the circle of radius", circle3.radius,  "is",circle3.getArea())


        #Modify circle radius
        circle2.radius =100  # or circle2.setRadius(100)
        print("The area of the circle radius", circle2.radius. "is", circle2.getArea())


 main()  # Call the main function
```

Exercise:

Given: A UML Class Diagram below:

Required: Create a Python Code for creating the Class named TV and a Test Driver program named TestTV that will create two objects from Class TV and will produce the following output:

```
tv1's channel is 30 and volume level is 3
tv2's channel is 3 and volume level 2
```



TV	
channel: int	The current channel (1 to 120) of this TV.
volumeLevel: int	The current volume level (1 to 7) of this TV.
on: bool	Indicates whether this TV is on/off.
TV()	Constructs a default TV object.
turnOn(): None	Turns on this TV.
turnOff(): None	Turns off this TV.
getChannel(): int	Returns the channel for this TV.
setChannel(channel: int): None	Sets a new channel for this TV.
getVolume(): int	Gets the volume level for this TV.
setVolume(volumeLevel: int): None	Sets a new volume level for this TV.
channelUp(): None	Increases the channel number by 1.
channelDown(): None	Decreases the channel number by 1.
volumeUp(): None	Increases the volume level by 1.
volumeDown(): None	Decreases the volume level by 1.

FIGURE 7.7 The TV class defines TV sets.