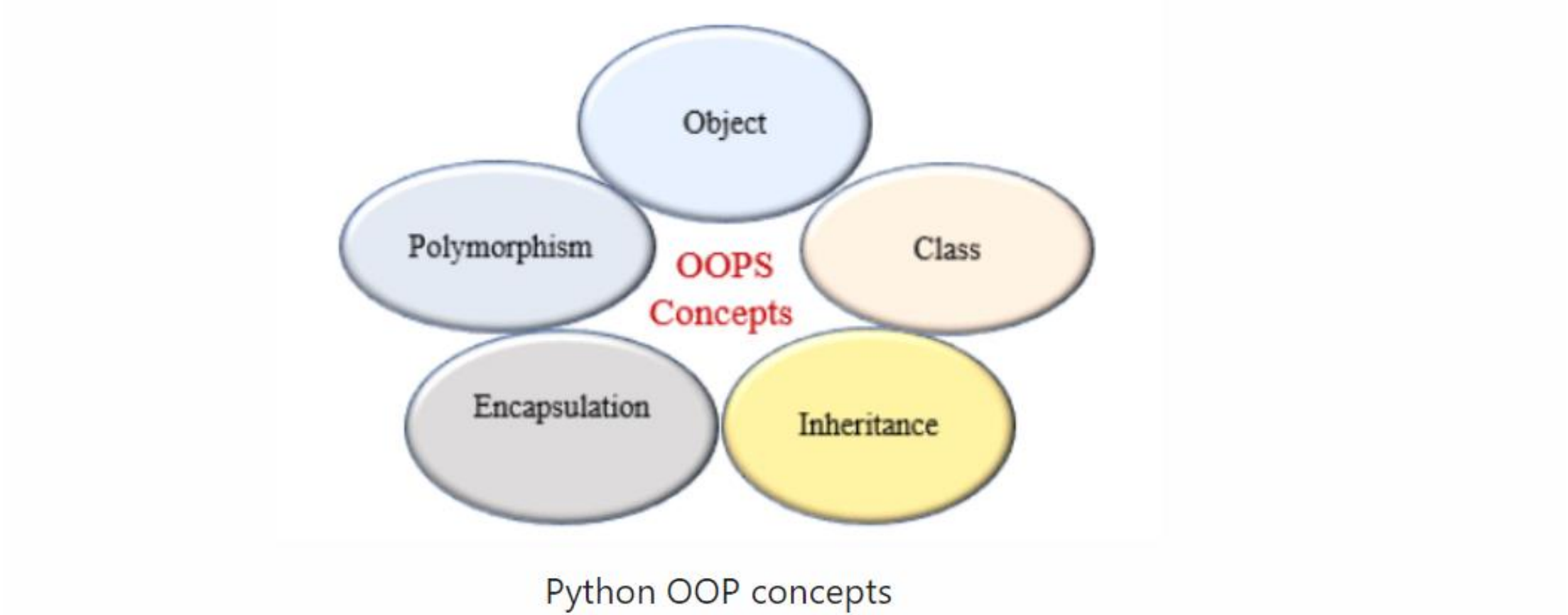
OOPs Python

What is Object Oriented Programming in Python

Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects". The object contains both data and code: Data in the form of properties (often known as attributes), and code, in the form of methods (actions object can perform).



Python OOP concepts

An object has the following two characteristics:

- Attribute
- Behavior

For example, A Car is an object, as it has the following properties:

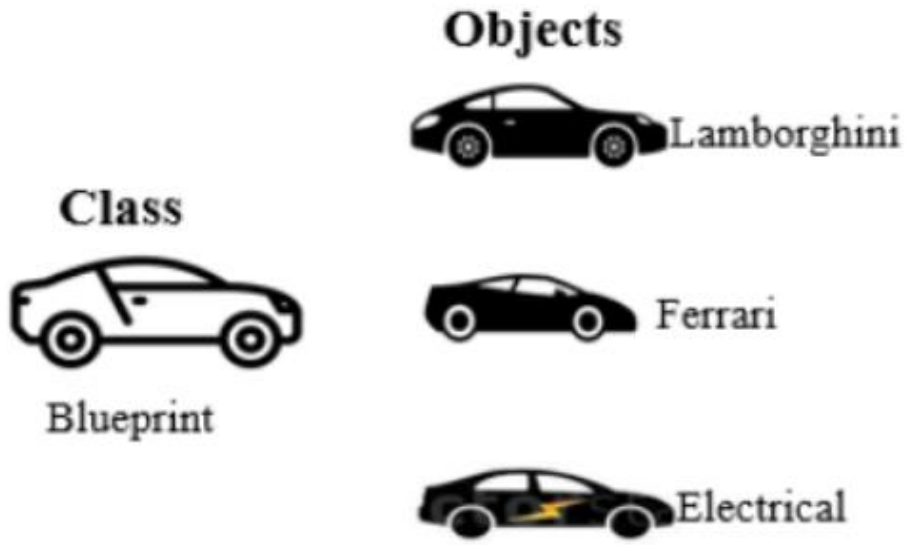
- name, price, color as **attributes**
- breaking, acceleration as **behavior**

1. Classes and objects
2. Instance Variables and methods (Object variables and methods)
3. Class Variables and methods

Class and Objects

In Python, everything is an object. A class is a blueprint for the object. To create an object we require a model or plan or blueprint which is nothing but class.

For example, you are creating a vehicle according to the Vehicle blueprint (template). The plan contains all dimensions and structure. Based on these descriptions, we can construct a car, truck, bus, or any vehicle. Here, a car, truck, bus are objects of Vehicle class



Python Class and Objects

Object is an instance of a class. The physical existence of a class is nothing but an object. In other words, the object is an entity that has a state and behavior. It may be any real-world object like the mouse, keyboard, laptop, etc.

Class Attributes and Methods
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.

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 **Syntax**

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

A real-life example of class and objects.

Class: Person

- **State**: Name, Sex, Profession
- **Behavior**: Working, Study

Using the above class, we can create multiple objects that depict different states and behavior.

Object 1: Jessa

- **State**:
  - Name: Jessa
  - Sex: Female
  - Profession: Software Engineer
- **Behavior**:
  - Working: She is working as a software developer at ABC Company
  - Study: She studies 2 hours a day

Object 2: Jon

- **State**:
  - Name: Jon
  - Sex: Male
  - Profession: Doctor
- **Behavior**:
  - Working: He is working as a doctor
  - Study: He studies 5 hours a day

Example: Define a class in Python

In this example, we are creating a Person Class with name, sex, and profession instance variables.

```
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.

Syntax

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

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

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

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

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

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.

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.

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.

Instance methods work on the instance level (object level).

A class method is bound to the class and not the object of the class.

```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()
```

We should follow specific rules while we are deciding a name for the class in Python.

Rule-1: Class names should follow the UpperCaseCamelCase convention
Rule-2: If a class is callable (Calling the class from somewhere), in that case, we can give a class name like a function.
Rule-3: Python’s built-in classes are typically lowercase words