Python

Introduction to Programming

Comp07027

Lecture 11

Object Oriented Programming (OOP)

In this series, you will learn OOP (Object Oriented Programming) in Python. OOP concepts include object, classes, constructor and encapsulation, polymorphism, and inheritance.

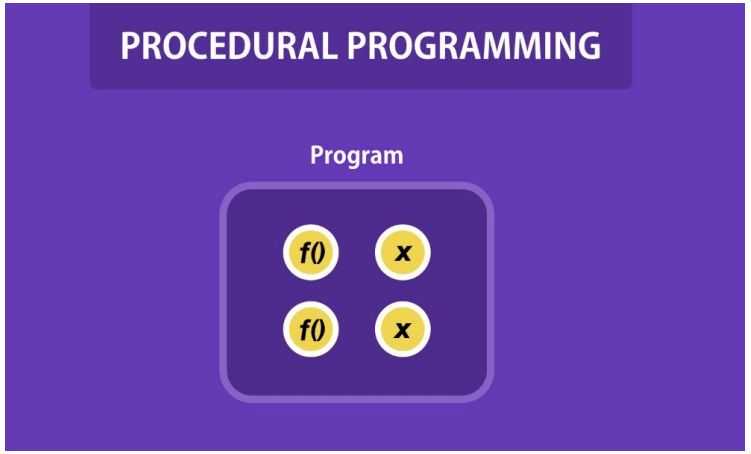
OOP

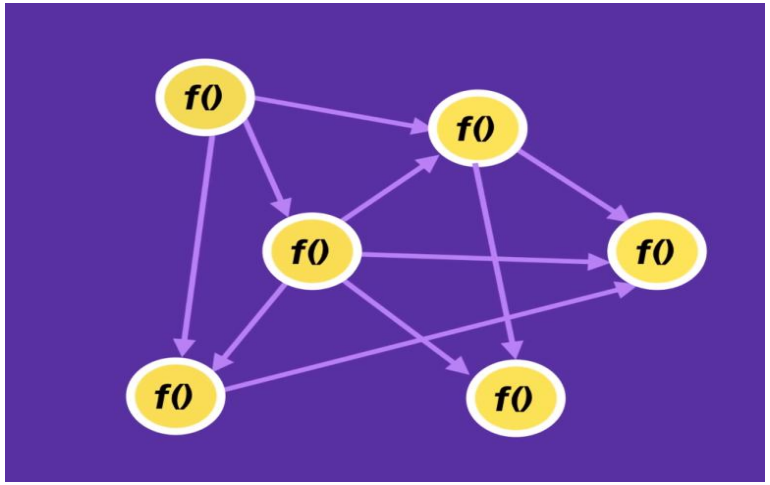
So far we have used python for structured (procedural) programming.

We have introduced how to define and use functions

However, Python can do much more!

- What is class and objects in Python
- Class attributes and methods
- Creating and accessing object properties
- Modify and delete an object





Objects – in the Real World

In the real world we are surrounded by objects (car, house, person)

Objects are ‘things’, they are tangible and are usually nouns

Objects in the real world have state (e.g. the car is white)

Objects in the real world also exhibit behaviours (e.g. the man is running)

Objects – in the Real World

Look at this Dog :

Object	State	Behaviour
Dog	Name Colour Breed Happy	Bark Wag Tail Eat Fetch

Objects – in software

In software we can make a model of the real world.

Software objects model real world objects.

They also have state and behaviour.

Software objects have variables (attributes) which maintain the state of the object.

Software objects have methods which implement the behaviours of the object.

Objects – in software

Look at this Dog object:

Object	Attributes	Methods
Dog	Name Colour Breed Happy	Bark Wag Tail Eat Fetch

Objects – in software

Software objects are not tangible but are models of tangible objects.

A software object is a model consisting of-

- A set of data (attributes to maintain state)
- A set of possible methods (to implement behaviours)

Classes

In the real world there are many millions of similar objects.

Take three dogs-

They each are dogs, but each is a distinct dog in its own right

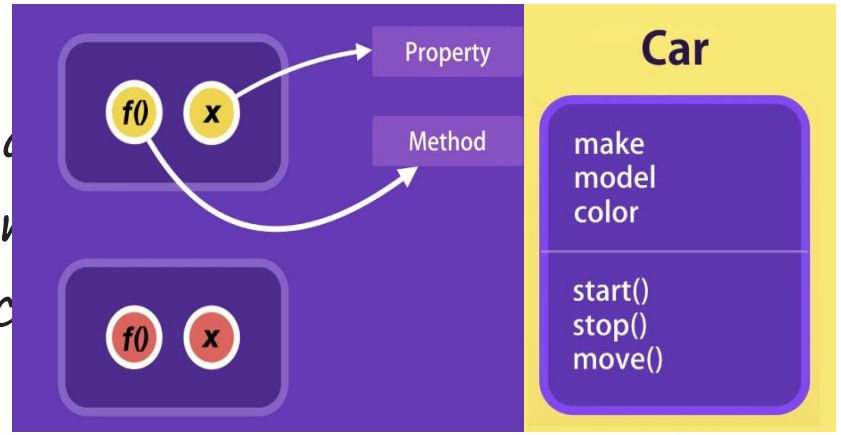
All three dogs have the same set of characteristics (state and behaviour)

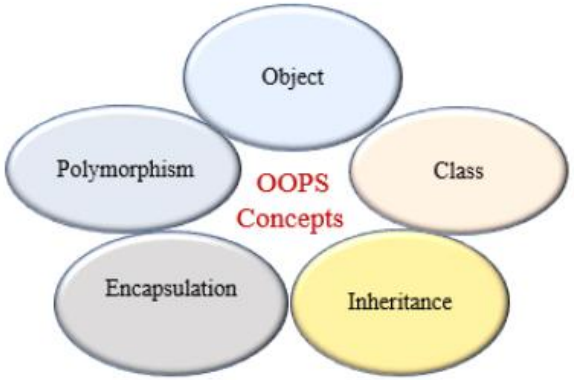
In OOP a class is a ‘blueprint’ for objects which share the same set of states and behaviours (attributes and methods)

Each object is an ***instance*** of the class – it may have a different state from other instances of the class (one dog might be happy and another not!)

OOP Basic

- ***Object-oriented programming*** (OOP) is a programming pa
on the concept of "**objects**". The object contains both data an
in the form of properties (often known as attributes), and c
in the form of methods (actions object can perform).
- An object-oriented paradigm is to design the program
using classes and objects.





Python OOP concepts

What is Encapsulation in Python?

Encapsulation in Python describes the concept of **bundling data and methods within a single unit**. So, for example, when you create a class, it means you are implementing encapsulation. A class is an example of encapsulation as it binds all the data members (instance variables) and methods into a single unit.

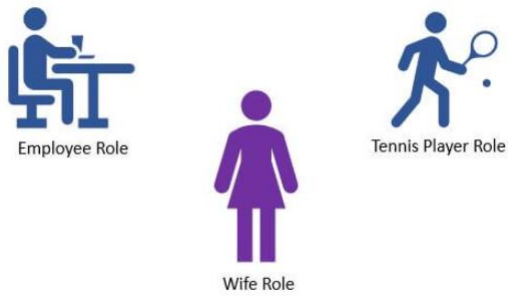
```
class Employee:
    def __init__(self, name, project):
        self.name = name
        self.project = project

    def work(self):
        print(self.name, 'is working on', self.project)
```

Data Members

Method

What is Polymorphism in Python?

- Polymorphism in Python is the ability of an object to take many forms. In simple words, polymorphism allows us to perform the same action in many different ways.
- For example, Jessa acts as an employee when she is at the office. However, when she is at home, she acts like a wife. Also, she represents herself differently in different places. Therefo takes different forms as per the situation.



Jessa takes different forms as per the situation

- In polymorphism, a method can **process objects differently depending on the class type or data type**.

- **What is Inheritance in Python?**

- The **process of inheriting the properties of the parent class into a child class is called inheritance**. The existing class is called a base class or parent class and the new class is called a subclass or child class or derived class.

- In Object-oriented programming, inheritance is an important aspect. The main purpose of inheritance is the **reusability** of code because we can use the existing class to create a new class instead of creating it from scratch.

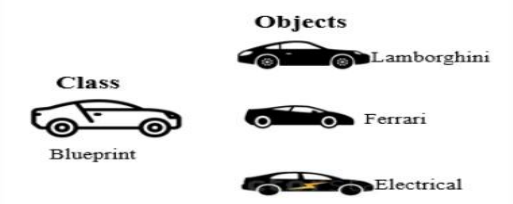
Benefits of the OOP

BENEFITS OF OOP

Encapsulation	Reduce complexity + increase reusability
Abstraction	Reduce complexity + isolate impact of changes
Inheritance	Eliminate redundant code
Polymorphism	Refactor ugly switch/case statements

Relationship between class and object

- 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.
- A class contains the properties (attribute) and action (behavior) of the object. Properties represent variables, and the methods represent actions. Hence class includes both variables and methods.
- **Object is an instance of a class**. The physica ss is nothing but an object. In other words, the that has a state and behavior. It may be any real-w e mouse, keyboard, laptop, etc.



Python Class and Objects

Class attributes and method

- 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.

- Inside a Class, we can define the following two major types of methods:
- **Instance method**: Used to access or modify the object attributes. 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.

Creating Class and Objects

- In Python, Use the keyword `class` to define a Class. In the class definition, the first string is docstring which, is a brief description of the class.

```
class classname:
    '''documentation string'''
    class_suite
```

- **Documentation string**: represent a description of the `class`. It is optional.
- **class_suite**: `class` suite contains class attributes and methods

- We can create any number of objects of a class. use the following syntax to create an object of a class.
- `reference_variable = classname()`

- ***OOP Example: Creating Class and Object in Python***

```python
class Employee:
    # class variables
    company_name = 'ABC Company'

    # constructor to initialize the object
    def __init__(self, name, salary):
        # instance variables
        self.name = name
        self.salary = salary

    # instance method
    def show(self):
        print('Employee:', self.name, self.salary, self.company_name)

# create first object
emp1 = Employee("Harry", 12000)
emp1.show()

# create second object
emp2 = Employee("Emma", 10000)
emp2.show()
```

•Constructors are used for initializing the objects . If you don't mention the constructor, it will use default constructor.
•The self is used to represent the instance of the class. With this keyword, you can access the attributes and methods of the class in python. It binds the attributes with the given arguments.

- **Explanation of OOP Example: Creating Class and Object in Python**

```python
class Employee:
    # class variables
    company_name = 'ABC Company'

    # constructor to initialize the object
    def __init__(self, name, salary):
        # instance variables
        self.name = name
        self.salary = salary

    # instance method
    def show(self):
        print('Employee:', self.name, self.salary, self.company_name)

# create first object
emp1 = Employee("Harry", 12000)
emp1.show()

# create second object
emp2 = Employee("Emma", 10000)
emp2.show()
```

•In the above example, we created a Class with the name Employee.
•Next, we defined two attributes name and salary.
•Next, in the `__init__()` method, we initialized the value of attributes. This method is called as soon as the object is created. The init method initializes the object.
•Finally, from the Employee class, we created two objects, Emma and Harry.
•Using the object, we can access and modify its attributes.

constructor

Parameters to constructor

```
class Student:
    def __init__(self, name, percentage):
        self.name = name  #
        self.percentage = percentage

    def show(self):
        print("Name is:", self.name, "and percentage is:", self.percentage)
```

Instance variable

Instance variable

Instance method

Object of class

```
stud = Student("Jessa", 80)
stud.show()
# Output: Name is: Jessa and percentage is: 80
```

instance variables and methods

Constructors in Python

- In Python, a **constructor** is a special type of method used to initialize the object of a Class. The constructor will be executed automatically when the object is created. If we create three objects, the constructor is called three times and initialize each object.
- The main purpose of the constructor is to declare and initialize instance variables. It can take at least one argument that is `self`.
  The `__init()__` method is called the constructor in Python. In other words, the name of the constructor should be `__init__(self)`.
- A constructor is optional, and if we do not provide any constructor, then Python provides the default constructor. Every class in Python has a constructor, but it's not required to define it.

constructor

Parameters to constructor

```
class Student:
    def __init__(self, name, percentage):
        self.name = name  #
        self.percentage = percentage

    def show(self):
        print("Name is:", self.name, "and percentage is:", self.percentage)
```

Instance variable

Instance variable

Instance method

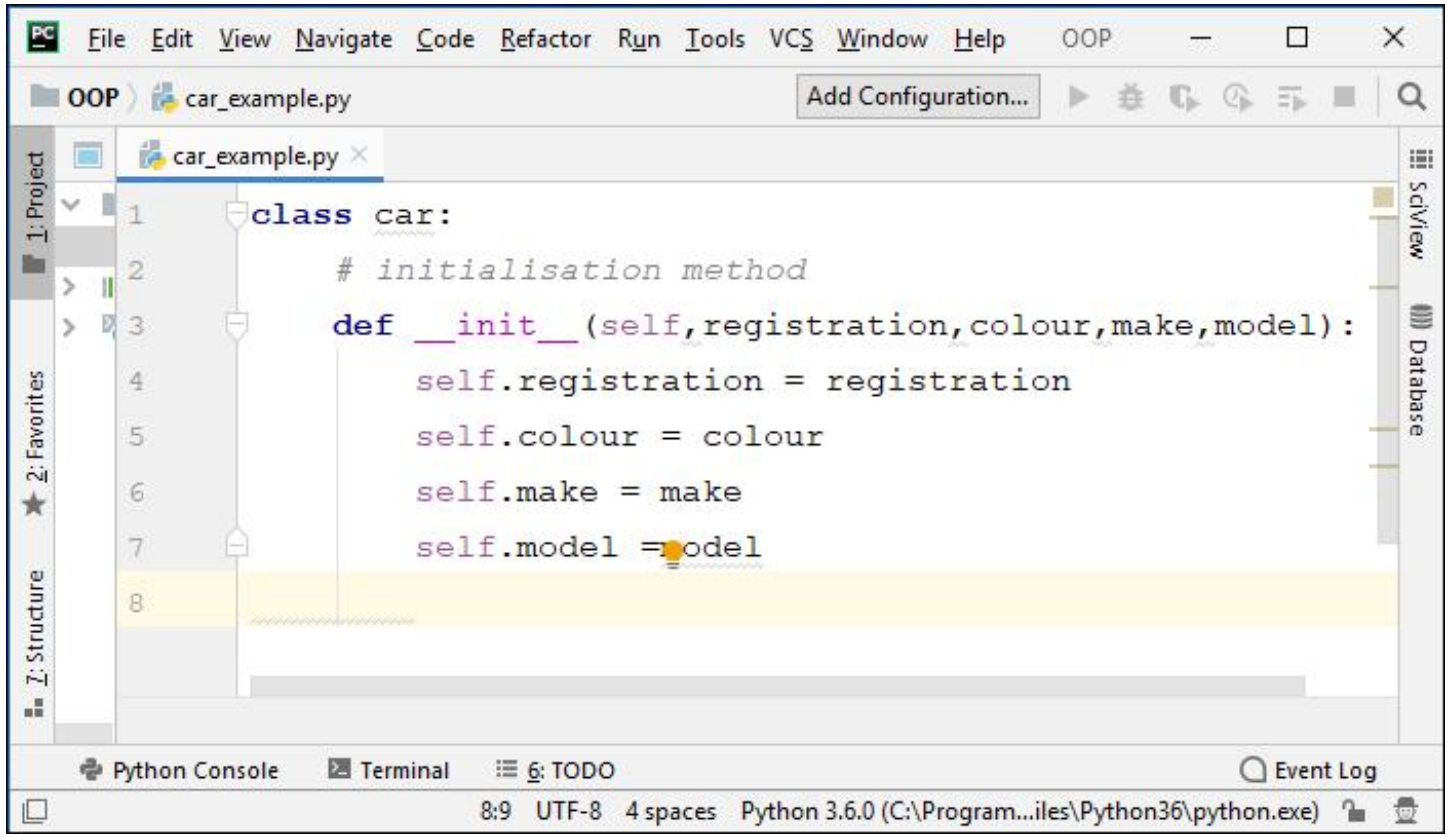
Object of class

```
stud = Student("Jessa", 80)
stud.show()
# Output: Name is: Jessa and percentage is: 80
```

instance variables and methods

Defining a Class

We can define a Class (blueprint) for a car, with the characteristics registration number, colour, make and model.



File Edit View Navigate Code Refactor Run Tools VCS Window Help OOP

OOP car_example.py car_example

car_example.py

```python
class car:
    # initialisation method
    def __init__(self,registration,colour,make,model):
        self.registration = registration
        self.colour = colour
        self.make = make
        self.model =model


car1 = car("AB01CDE","White","Ford","Focus")
car2 = car("FG02HIJ","Blue","Vauxhall","Corsa")
car3 = car("KL03MNO","Green","Volkswagen","Polo")
car4 = car("PQ04RST","Red","Toyota","Yaris")
```

Run: car_example

Python Console Terminal 4: Run 6: TODO Event Log

9:1 CRLF UTF-8 4 spaces Python 3.6.0 (C:\Program...iles\Python36\python.exe)

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.

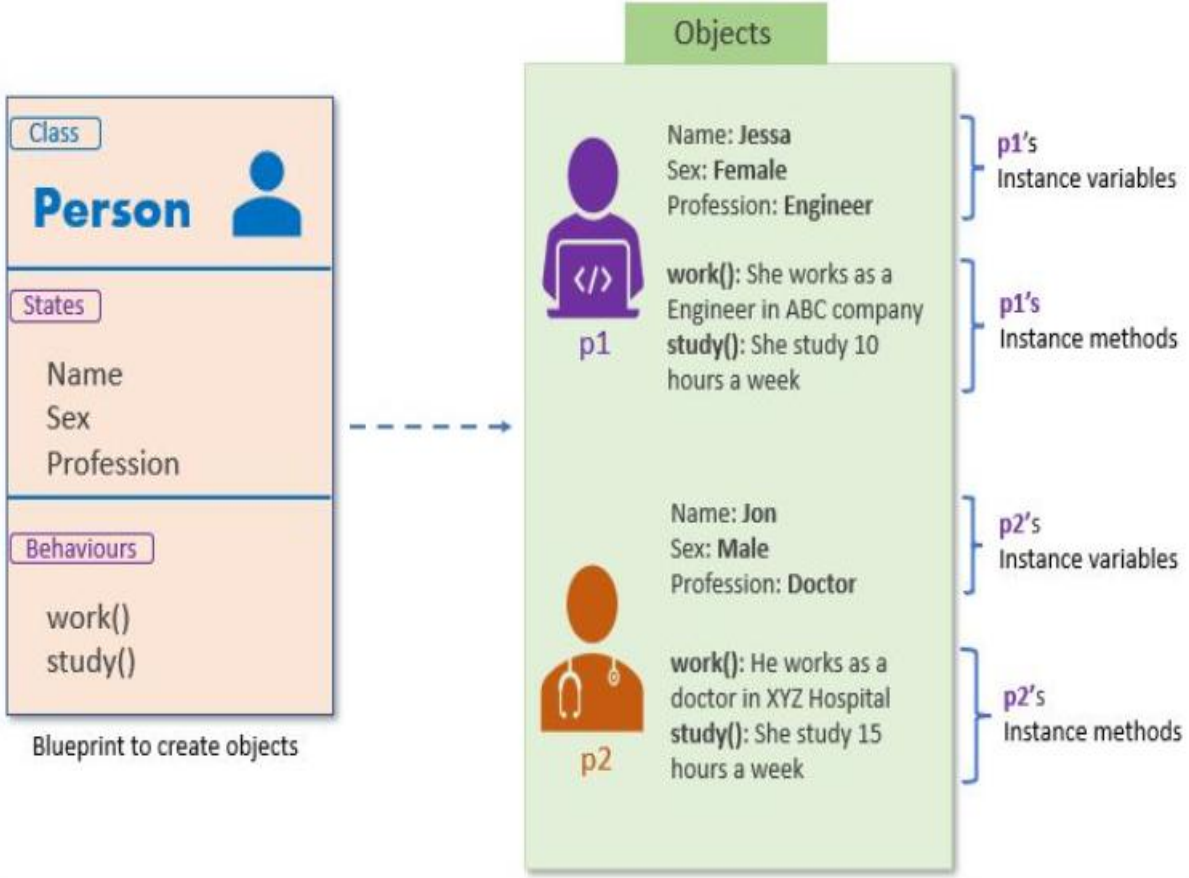
Object 1: Jessa

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

Object 2: Jon

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

As you can see, Jessa is female, and she works as a Software engineer. On the other hand, Jon is a male, and he is a lawyer. Here, both **objects are created from the same class, but they have different states and behaviors**.



Define a class in Python

```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.

Syntax

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

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

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

The complete example:

```
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()
```

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

Example

```
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)
```

Define and call an instance method and class method

```python
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()
```

Object Properties

Every object has properties with it. In other words, we can say that object property is an association between **name** and **value**.

For example, a car is an object, and its properties are car color, sunroof, price, manufacture, model, engine, and so on. Here, color is the name and red is the **value**. Object properties are nothing but instance variables.

Object Properties

Modify Object Properties

Every object has properties associated with them. We can set or modify the object's properties after object initialization by calling the property directly using the dot operator.

```python
#Modify Object Properties

class Fruit:
    def __init__(self, name, color):
        self.name = name
        self.color = color

    def show(self):
        print("Fruit is", self.name, "and Color is", self.color)

# creating object of the class
obj = Fruit("Apple", "red")

# Modifying Object Properties
obj.name = "strawberry"

# calling the instance method using the object obj
obj.show()
# Output Fruit is strawberry and Color is red
```

Delete object properties

We can delete the object property by using the `del` keyword. After deleting it, if we try to access it, we will get an error.

```python
#Delete object properties

class Fruit:
    def __init__(self, name, color):
        self.name = name
        self.color = color

    def show(self):
        print("Fruit is", self.name, "and Color is", self.color)

# creating object of the class
obj = Fruit("Apple", "red")

# Deleting Object Properties
del obj.name

# Accessing object properties after deleting
print(obj.name)
# Output: AttributeError: 'Fruit' object has no attribute 'name'
```

Delete Objects

In Python, we can also delete the object by using a `del` keyword. An object can be anything like, class object, `list`, `tuple`, `set`, etc.

```
class Employee:
    depatment = "IT"

    def show(self):
        print("Department is ", self.depatment)

emp = Employee()
emp.show()

# delete object
del emp

# Accessing after delete object
emp.show()
# Output : NameError: name 'emp' is not defined
```

Questions??