# PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

## Object Oriented Programming

**Prof. Sindhu R Pai**
PCPS Theory Anchor - 2024
Department of Computer Science and Engineering

# Object oriented Programming

Programming Paradigm: Style or way or strategy using which we write the solution

## Procedure Oriented Programming (POP)

- Focus is on procedures.
- Procedure is a set of instructions used to accomplish a specific task.
- It can be routines, sub-routines, functions etc.

Examples : C, Pascal, Fortran, Cobol, Algol etc.

## Object Oriented Programming (OOP)

Focus is on the data and the operations that manipulate the data.

Helps in organizing and designing code by representing real-world entities as objects.

OOP is mainly useful to develop big and complex projects carried out by large teams.

Ex: Java, C#, C++, Python etc.

## Features of OOP

### Data Abstraction:

- The way you view an object(Ex: employee, library, customers etc)
- Represents the essential features without background details.
- Depending on the application, abstraction has to be implemented.

### Data encapsulation:

- Binding of data and procedures as a single unit.
- Encapsulation is a way to implement data abstraction.

## Object oriented Programming

### Data Hiding:

- Its about who can access the data .
- Implemented using access specifiers.

### Inheritance:

- Capability of one class (child) to derive the capabilities of another class(parent)
- Code reusability is the major benefit of inheritance.

### Polymorphism:

- Capability of an object of a class to behave differently in response to the data.

## Key concepts in OOP:

class:

- It is a methodology to create an entity.
- Blueprint or template for creating objects.
- Defines attributes (variables) and methods (functions) that all objects of that class will have.
- The class by itself is a **type and implementation**.
- A class specifies the set of instance variables and methods that are "bundled together"

Syntax: (To create a class)

```
class ClassName:
        <statement-1>
        .
        <statement-N>
```

Example 1

```
class Ex1:
        pass
print(Ex1, type(Ex1))
```

Output:

```
<class '__main__.Ex1'> <class 'type’>
```

So, Ex1 is a type in the package __main__

## Object oriented Programming

Example 2: class with attributes (fields or variables)

```
class Car:
        car_name="Benz"
print(Car.car_name)
```

Output:

```
Benz
```

Example 3: class with methods (behaviour)

```
class Car:
        def fuel():
                print("Petrol")
Car.fuel()
```

Output:

```
Petrol
```

## Example 4: Class having data and a method.

```
class Car:
        car_name="Benz"
        def display():
                print("This is a Petrol Car")
print(Car.car_name)
Car.display()
```

Output:

```
Benz
This is a Petrol Car
```

## Objects (Instances):

- It is a a physical instance of a class.
- Represents the real-world entities
- Have their own attributes (class variables/instance variables) and methods(class functions), as defined by the class.

## Objects have:

1.Identity: Each object has a unique identity throughout its lifetime.
The id() function can be used to get the identity of an object.

2.Type: The type() function can be used to determine the type of an object.

3.Value: Objects have a value that can be modified or accessed (like integers, strings, lists)

Example 5:

```
class Car:
        pass
c1 = Car()
print(c1)
print(type(c1))
print(id(c1))
```

Output:

```
<__main__.Car object at 0x000001C9E2200DC0>
<class '__main__.Car'>
1966593805760
```

## Instantiation

- The existence of a class does not create the object.
- Object must be created explicitly
- To create an object of the class, use the function which has the same name as class.

  Ex:  c1= Car()    will create an object(c1) of class Car
- (.)Dot operator notation can be used  to access attributes of the class.

  Ex: c1.car_name

## Constructor

- It is a special function of the class which is called when an object is created
- The name of the this function (constructor) is **__init__**
- It is invoked automatically and implicitly when the object of the class is created.
- The constructor can be used to initialize the internal state of an object, and create instance variables

## Destructor

- It is a special function within the class by name __del__, that performs the clean-up actions when an object is destroyed or deleted.

- The destructor is automatically called just before an object is removed from memory by the garbage collector.

- It is often used to release resources before an object is removed from the system.

## Example 6

```
class Car:
        def __init__(self):
                print("constructor called")
                print('self : ', self)
c1= Car()
print('c1 : ', c1)
```

Output:

```
constructor called
self :  <__main__.Car object at 0x000001F214B30DC0>
c1 :  <__main__.Car object at 0x000001F214B30DC0>     #Same output
```

**Note: self is a reference to the instance of a class and is used to access the attributes and methods within that instance and should be the first parameter.**

Example 7:

```
class Car:
        def __init__():
                print("constructor called")
c1 = Car()
```

Output:

```
Traceback (most recent call last):
 File "C:/Users/ADMIN/Desktop/practice programs.py", line 38, in
<module>  car() TypeError: __init__() takes 0 positional arguments but 1
was given
```

As we see in the output, the constructor needs  a parameter called **self**

## Example 8: Use of constructor and destructor

```
class MyClass:
   def __init__(self, name):
      self.name = name
      print(f"Constructor called. {self.name} created.")

   def some_method(self):
      print(f"Hello from {self.name}!")

   def __del__(self):
      print(f"Destructor called.{self.name} deleted.")

obj1 = MyClass("Object 1")          # Creating objects
obj1.some_method()
del obj1                            # Explicitly deleting an object
```

## Output

```
Constructor called. Object 1 created.
Hello from Object 1!
Destructor called. Object 1 deleted.
```

Note: The destructor is called when the object obj1 is explicitly deleted using “del obj1”

**The module garbage collector automatically manages memory and calls destructors when objects are not needed.**

## Types of Constructors :

1. Parameterized  constructor
2. Non Parameterized constructor

## Example 9: Non-parameterized constructors

```
#when no parameters are passed for invoking constructor
class Person:
        def __init__(self):
                print("Constructor without parameters")
p = Person()
```

Output:

```
Constructor without parameters
```

## Example 10: Parameterized constructors

```
#Parameters are passed for invoking the constructor
class Person:
        def __init__(self,name,age):
                self.name = name #instance variables
                self.age = age
        def display(self):
                print(self.name,self.age)        #all names must be fully qualified
p = Person("joe",30)                        # value of self will be implicitly assigned
p.display()
p.gender=‘M’                # add instance variables to the object outside the class
print(p.gender)
```

# Object oriented Programming(Practice Program)

Example 11:Demonstration class & object with constructor and destructor.

```
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def display(self):
        print(f"Rectangle dimensions: {self.length} x {self.width}")

    def __del__(self):
        print("Rectangle object destroyed.")

# Creating a Rectangle object
rect = Rectangle(5, 10)
 rect.display()
```

## Getter and Setter Method:

### Getter:

- Used to retrieve the value of attribute of a class without directly exposing it.

### Setter:

- Used to modify the value of attribute of a class.

- Allows controlled modification of the attribute's value by performing checks or validations before assigning the new value.

## Ex: Getter and setter using user defined functions

```
class getset:
    def __init__(self, name):
        self.name = name
    def get_name(self):
        return self.name
    def set_name(self,new_name):
        self.name=new_name

 obj = getset("Arun")
 print("Name:: before calling setter",obj.get_name())           # Output: Arun
obj.set_name("Ram")
print("Name:: after calling setter",obj.get_name())              # Output: Ram
```

## Use of predefined functions

1 setattr() function sets the value of the specified attribute of the specified object.
**Syntax** setattr(object, attribute, value)

2 getattr() function returns the value of the specified attribute from the specified object.
**Syntax:** getattr(object, attribute, default)

3 hasattr() function returns True if the specified object has the specified attribute, otherwise False.
**Syntax** hasattr(object, attribute)

4 delattr() function will delete the specified attribute from the specified object.
**Syntax** delattr(object, attribute)

## Ex: Getter setter using predefined functions

```
class Person:
        name = "John"
        age = 36
        country = "Norway"
person=Person()
x = getattr(person, 'age')
print(x)
setattr(person,'age',"40")
x = getattr(person, 'age')
print(x)
x =hasattr(person,"age")
print(x)
```

## Practice programs

1.Define a class called "candidate" which has the Registration number and Score as the data members.
Write a function Input() which allows the user to enter values for the above data members.
Write a function called Selection() which assigns remarks as per the score obtained by the candidate as follows.
If score>=60 then assign remarks="Selected" else remarks= "Not selected"
Write a display() function to view the data members.

Test this created class for all its functionality by creating objects

## Object oriented Programming

2. Define a class called Travel with the following descriptions:
Class members: Travel _Code, Place, Number _of _travelers, Number _of _buses

Define a constructor which initializes the above class members with the values  105, “Bombay”,15, and 5 respectively.
Take the input() from the user for all the data members.
Also assign the number of buses equal to 1 if number of travelers is  greater or equal to 10 otherwise no bus  is assigned.

Test this created class for all its functionality by creating objects

## Object oriented Programming

3. Define a class called “ Stock” with following  data members
item _code, item _name, price, quantity , Discount.

Define a member function CalcDisc() to calculate the discount as per the following:
If quantity<=100  then discount=0
If quantity <=150 then discount is 5%
If quantity >150  then discount is 10%
Write a function Enter_Details() which allows the user to enter values to the data members and call CalcDisc() to calculate the discount.
Write a function  Display() to view the contents of all the data members.

Test this created class for all its functionality by creating objects

4.Define a class to represent the bank account of a customer with the information like Name  of the depositor, Account number and type of the account(Savings, Current) and Balance amount.

Define separate methods for the following:

1. To initialize the data member
2. To Deposit the amount
3. To withdraw the amount after checking the balance amount
4. To display the data members

PES
UNIVERSITY

# THANK YOU

Department of Computer Science and Engineering

Dr. Shylaja S S, Director, CDSAML & CCBD, PESU
Prof. Sindhu R Pai – sindhurpai@pes.edu
Prof. C N Rajeswari