OOPs | 01-OOPS concept

class, object, properties and behaviours

Posted on October 16, 2021
Last updated on February 5, 2023

1. OOPs Concept

- `Object Oriented Programming`
- OOP is a specific way of designing a program by `using classes and objects`
- It allows us to relate program with real-world objects, like `properties` and `behaviors`
- `Properties`
  - It define the `state of the object.`
  - Like: name, age, address etc that store data of the object
- `Behaviors`
  - Are the `actions` our object can take.
  - Like: talking, breathing, running etc
  - Oftentimes, this involves using or modifying the properties of our object.

```
# Example
DOOR CLASS

Properties:
  height
  color
  is_locked

Behaviors:
  open()
  closed()
  toggle_lock()
```

1.1 Class

- Classes are used to create `user-defined data structures`.
- It's like a `blueprint for an object` where we define all its properties and behaviours

```
class className:
    ''' documenttation string '''
    variables: instance variables,static and local variables
    methods: instance methods,static methods,class methods

# Help
print(className.__doc__)
help(className)
```

1.2 Object

- An Object is an `instance of a Class` it's like a copy of the class.
- Pysical existence of a class is nothing but object. We can create any number of objects for a class.

`Everything in Python is an object`, and almost everything has attributes and methods

- Properties -> `Variables/Attribute`
- Behaviour -> `Methods`

```
class Employee:
  # Variable/Attribute
  raise_amount=1.04

  def __init__(self, name):
    self.name=name

  # Methods
  def user(self):
    print(self.name)

emp1=Employee('Amrit')
# Printing object's memory hex
print(emp1)

emp1.user()
# Amrit
```

1.3 Four pillars of OOPs

[1.3.1 Encapsulation]

- It is used to `restrict access to methods and variables.`
- Code and data are wrapped together within a single unit.
- Access modifier are useful to attain encapsulation
- It is achived by access modifiers `public, private, protected`

1.3.2 Data Abstraction

- Abstraction means `hiding internal details and showing functionality`
- It can be achive by using `abstract classes and interfaces`
- Interface
  - All the methods in an interface are declared with an empty body
- Abstract class
  - That contains one or more abstract methods
  - Abstract methods are the methods that generally don't have any implementation
- NOTE:
  - we cannot create objects for the abstract class
  - can contain the both method normal and abstract method.

1.3.3 Inheritance

- It specifies that the child object acquires all the properties and behaviors of the parent object.
- The new class is known as a `derived class or child class`, and the one whose properties are acquired is known as a `base class or parent class`.
- It provides `re-usability of the code`.
- Python Inheritance Terminologies
  - `Superclass`: The class from which attributes and methods will be inherited.
  - `Subclass`: The class which inherits the members from superclass.
  - `Method Overriding`: Redefining the definitions of methods in subclass which was already defined in superclass.
- **Important Topics**
  - Multiple Inheritance vs Multi-level Inheritance
  - Resolving the Conflicts with python multiple inheritance
  - Method Resolution Order (MRO)

1.3.4 Polymorphism

- Poly means many. Morphs means forms.
- Polymorphism means `Many Forms`

Yourself is best example of polymorphism. In front of Your parents You will have one type of behaviour and with friends another type of behaviour. `Same person but different behaviours` at different places, which is nothing but polymorphism.

- Polymorphism can be achived by
  - `Method overriding`
  - `Method Overloading`

Method overriding

- Override parent methods in child class.
- Change the implementation of a method in child class that is defined in parent class.
- Following conditions must be met for overriding a function:
  - `Inheritance` should be there.
  - Function overriding cannot be done within a class.
  - The function that is redefined in the child class should have the `same signature` as in the parent class i.e. same number of parameters.

Method Overloading

- We define a number of methods with the `same name but with a different number of parameters` as well as parameters can be of different types.
- `By default function overloading is not available in python`
    - In symbol table, key is function name here.
    - Thus when another function with same name is encountered, in symbol table current key value pair is overridden.
- But by using decorator design pattern in python, function overloading can be implemented.
    - @overload(int, int) def area(length, breadth):
    - @overload(int) def area(size):

Garbage Collection

The main objective of Garbage Collector is to `destroy useless objects`. If an object does not have any reference variable then that object eligible for Garbage Collection. Garbage Collector `always running in the background` to destroy useless objects

By default Gargbage collector is enabled, but we can disable based on our requirement

```
# Returns True if GC enabled
gc.isenabled()

# To disable GC explicitly
gc.disable()

# To enable GC explicitly
gc.enable()
```

.

.

.