Object Oriented Programming Concepts

Dates: 07/04/2024 - 09/06/2024

Dhanashree N P

Session Plan

Session 1 - Friday

- Introduction to OOP
- Objects and Classes
- Inheritance

Session 2 - Saturday

- Polymorphism
- Encapsulation and Abstraction

Session 3 - Sunday

- Hands-on / Practice Exercises
- Q&A Session

Introduction

Object Oriented Programming (OOP) is a type of **programming paradigm**.

- What are programming paradigms ?
  - different ways or styles in which a given program or programming language can be organized.
  - contains structures, features, and opinions about how common programming problems should be tackled
  - that is, guidelines

Different Programming Paradigms

1. Imperative programming - detailed instructions
2. Procedural programming - functions and subroutines
3. Functional programming - ‘pure’ functions
4. Declarative programming - hides complexity
5. Object Oriented Programming - entities and classes

Examples - sorting an array of numbers, cooking a dish

Why are there different paradigms ?

There are so many programming languages too! Why ?

- Why do we need to know this ?

    - General Knowledge!
    - Some problems are better solved by some paradigms than the others
    - Gives a better approach to solve problems/code and helps to think out of the box!

Object Oriented Programming

- What is OOP ?

  OOP is a programming paradigm that focuses on organizing and structuring code using **objects** and **classes**. It represents a shift away from procedural programming, which focuses on linear, step-by-step processes.

  Data (attributes) and related code (methods) are grouped together.

  Key terms/concepts - **classes**, **object**, **attributes** and **methods**.

Classes and Objects

‘**Class**’ is a blueprint of the characteristics and behavior of a real world entity. It represents broad categories, like Car, Person, Dog, Bank Account etc.

‘**Object**’ is a specific instance of this class with real data and related behavior.

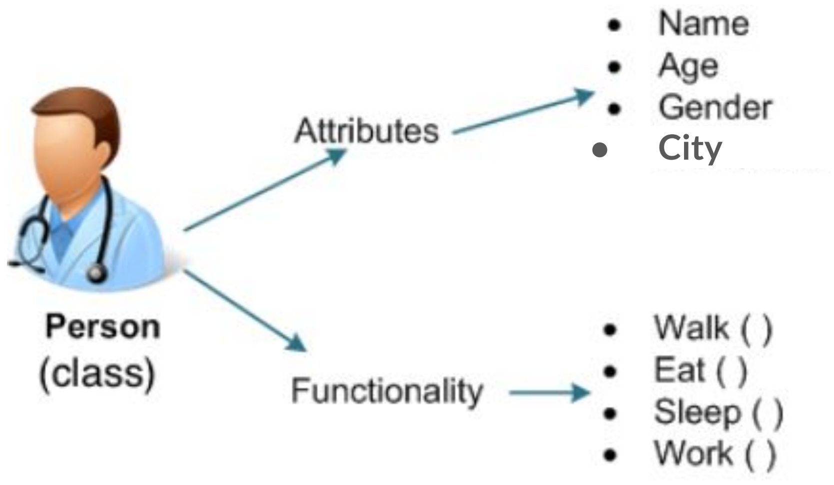
Let’s look at some real world examples.

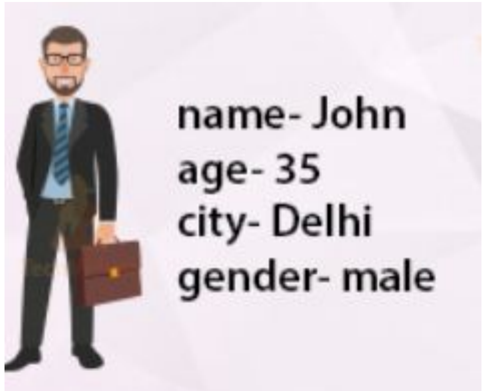
Example - 1

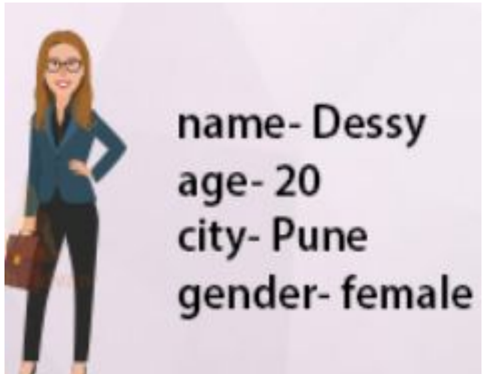
‘Person’ - is a **class** - with ‘age’ , ‘name’, ‘gender’ etc as **attributes**/ characteristics and ‘walking’, sleeping, ‘eating’ etc. as behavior/functionality/**methods.**

Dessy, a female aged 20 is an **instance / object** created from this class.

John, a male aged 35 is another **instance/ object** created from this class.



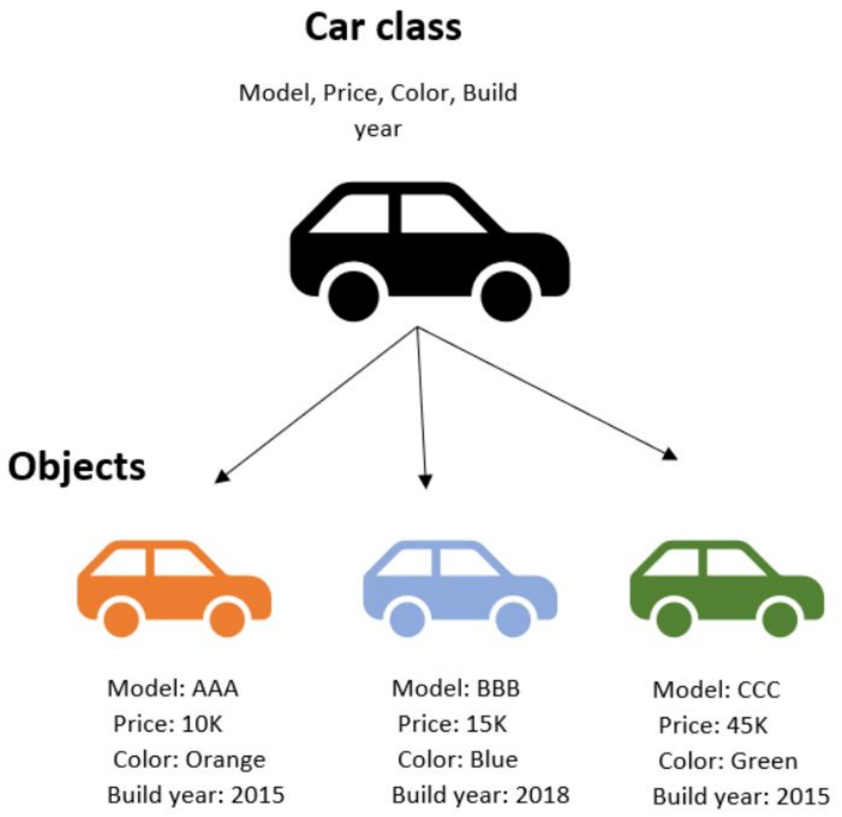




Example - 2

Think of some methods?

- Calculate resale_value()
- time_to_service()



Example - 3

Consider - Bank Account as class ?

What are the **attributes**? - We know ‘attributes’ is the ‘data’ associated with the objects

What can be the **methods** ? - We know ‘methods’ are the functions that define the object’s behavior.

Classes and Objects using Python

1. Defining classes
2. Attributes
3. Methods
4. Creating objects from classes

Let’s look at Python codes to understand these

OOP Concepts , Benefits of OOP

- Modularization - breaking down a large program to smaller independent components
- Reusability - use a module/code without having to re-write / duplicate code.
- Maintainability - ease with which a program/code can be improved/modified without unintended consequences
- **Encapsulation** - hiding implementation details, only expose interfaces to interact
- **Inheritance** - an object/class can inherit the properties and behavior of another object/class.
- **Abstraction** - expose essential features without details of implementation
- **Polymorphism** - object / function takes multiple forms / behavior based on the scenario

Inheritance

Inheritance is when one class can derive/inherit the methods and properties of other class(es).

It helps you to reuse functionalities of another class without rewriting everything.

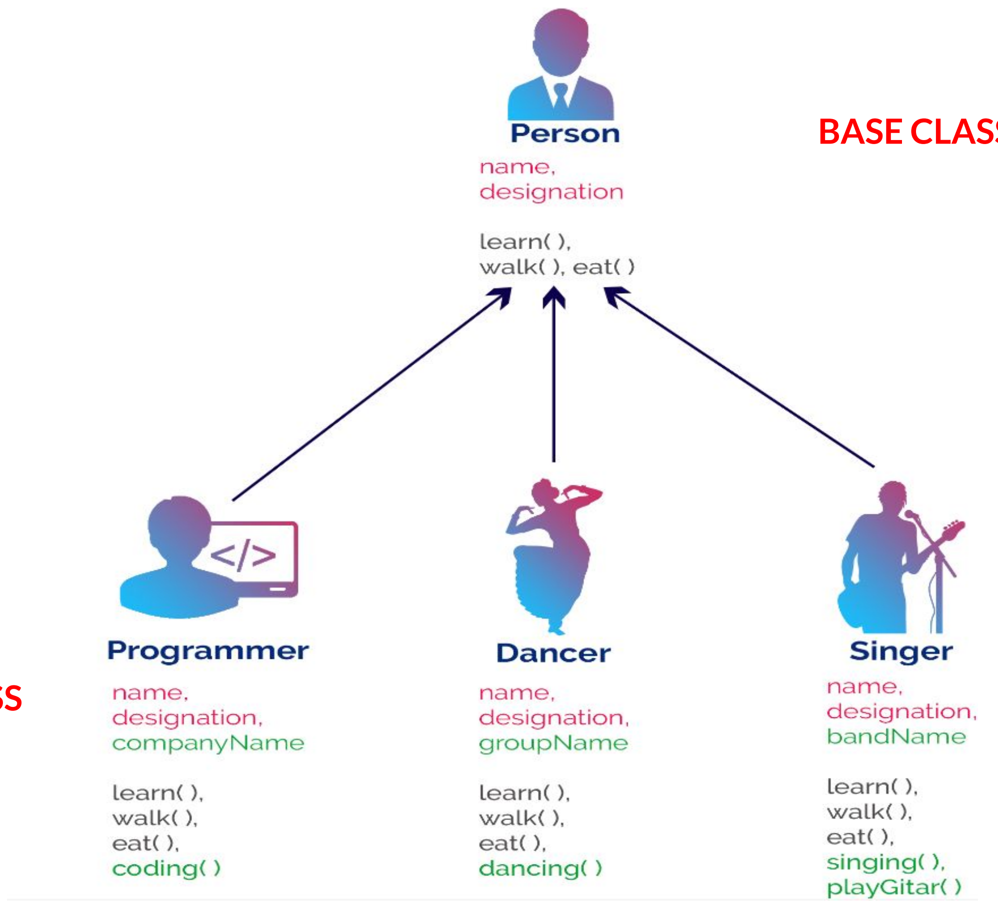
The inherited class is **parent** class and derived class is **child class**.

Child class can also **override/add** properties and methods of parent class

Example

BASE CLASS / PARENT CLASS

DERIVED CLASS / CHILD CLASS



Inheritance

Inheritance benefits - Code Reuse
Inheritance benefits - Specialization

Specialization is a technique in which a subclass inherits the attributes and methods of a superclass, and then specializes or extends the behavior of the superclass by adding new attributes or methods, or by overriding existing ones.

In the previous slide, ‘Programmer’ class has added a new attribute *companyName* and a new method/function *coding()*.

Method Overriding - Redefining inherited methods

Let’s look at Python codes to understand these

Quick Recap

1. We saw programming paradigms - as way/style of programming
2. OOP or Object Oriented Programming is one such
3. Emphasis on Classes and Objects - that have attributes and methods
4. Benefits - reusability, maintainability, modularization
5. How to define classes and create objects in Python

5. Pillars of OOPs - Inheritance, Polymorphism, Abstraction and Encapsulation

6. Inheritance - classes/object can inherit everything from other class(es)/object(s)

7. We saw parent/base class and child class example

8. How to create derived classes and create objects, redefine methods in Python

Polymorphism

‘Poly’ means *many* and ‘morph’ means *form*. So polymorphism means - having multiple forms. In OOP, we generally use it to refer the fact that objects may behave differently depending on the context.

Inheritance is a way of achieving Polymorphism - all the derived classes can either behave as the parent class or each one can have specific behaviors. Eg. Parent class = Person, Derived classes = Baby, Teenager

Talking behavior of the two classes modelled as talk() method. Baby - babbles, coos. Teenager - speaks clearly, uses slang

Polymorphism

How can a class method behave differently to achieve polymorphism?

Answer: Two ways - Function Overloading, Function Overriding

‘**Function Overriding**’ - Have same method as parent class but with different implementation inside child class. [Refer: Inheritance examples of method overriding]

‘**Function Overloading**’ - Have two/more methods with same name, but with different types/number of parameters

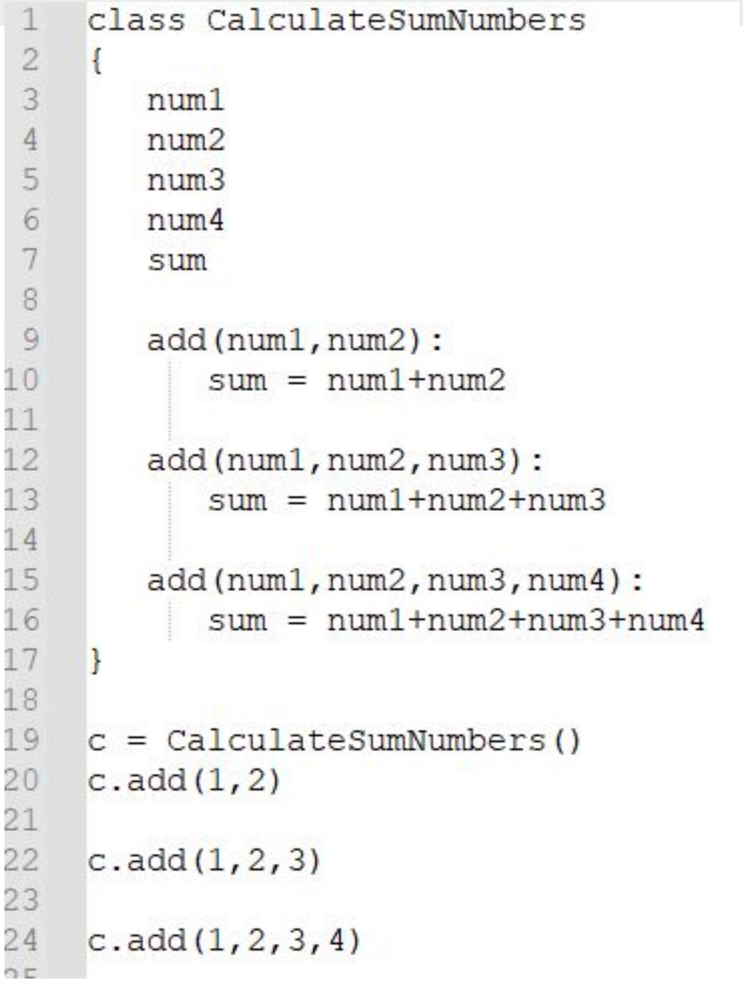
Function Overloading

'**Function Overloading**' - Have two/more methods with same name, but with different types/number of parameters.

Note: **NOT** SUPPORTED IN PYTHON, AS IT ALWAYS TAKES THE LAST IMPLEMENTATION OF THE FUNCTION.

THIS CAN BE DONE IN JAVA, C++ etc.

Pseudo-Code



```
class CalculateSumNumbers
{
    num1
    num2
    num3
    num4
    sum

    add(num1,num2):
        sum = num1+num2

    add(num1,num2,num3):
        sum = num1+num2+num3

    add(num1,num2,num3,num4):
        sum = num1+num2+num3+num4
}

c = CalculateSumNumbers()
c.add(1,2)

c.add(1,2,3)

c.add(1,2,3,4)
```

Encapsulation

- Encapsulation refers to **bundling data and methods that work on the data** in to a **single unit**.
- In this concept, we also make sure that we hide and protect attributes and methods which should not be accessed directly from outside, that is, '**Information Hiding**'
- We have something called **public**, **private** and **protected** '***access modifiers***' to achieve this.

  What does this mean?
- By **default** all class members are **public** and can be accessible outside and inside of the class.

Encapsulation

- **Protected** access - the attribute/method is accessible by that class and any of its subclasses.

- **Private** access - the attribute/method is only accessible by that class. Subclasses/derived classes and outside code **CANNOT** directly access this.

- In Python, we don't have any access modifier keywords. However we can use _ underscores in the beginning of attribute/method name.

Let's look at some Python codes to understand.

Encapsulation - Benefits

Adds security

Prevents unwanted modifications

Hides implementation details from outside

Abstraction

Abstraction means that the implementation is hidden from end user. Emphasis is on ease of use for end user.

This is an extension of the encapsulation concept.

Example: **Cars**, a driver operates a vehicle using only the car's dashboard and ABCGs

Benefits:

- Reduces complexity and improves code readability
- Facilitates code reuse and organization
- Data hiding improves data security by hiding sensitive details from users
- Enhances productivity by abstracting away low-level details

Importance of Abstraction

We can make applications more extensible and easier to share in complex systems and improve efficiencies.

We can define both classes and methods as abstract.

An abstract method doesn't have implementation and it is going to be defined in the subclasses.

Python, by default, does not provide abstract classes. We can use the abc module to define abstract classes.

Let’s look at a simple example

Questions ?

Session - 3

1. MCQ on OOP concepts - 10 questions
2. Review/Share Solution
3. Python Coding Assignment - Hands On
4. Review / Share solution

Reference Links

[Programming Paradigms – Paradigm Examples for Beginners (freecodecamp.org)]

[What is object-oriented programming? OOP explained in depth (educative.io)]

[Object oriented programming (OOP) in Python – Data stories (nonlineardata.com)]

[Top 50 Python OOPs Interview Questions and Answers in 2023 (testbook.com)]

[Object-Oriented Programming (OOP) in Python 3 – Real Python]

[What are public, protected, & private access modifiers in Python?]

[OOP Concept for Beginners: What is Encapsulation - Stackify]

[abc — Abstract Base Classes — Python 3.12.4 documentation]

[Best Ways in Python to Import Classes From Another File - Python Pool]

Thank You