

Object Oriented Programming



Dra. M^a Dolores Rodríguez Moreno

Objectives

Specific Objectives

- Overview of main programming paradigms
- Understand the Object Oriented concepts

Source

- <https://docs.python.org/3.10/tutorial/appetite.html>
- <https://python-textbok.readthedocs.io/en/1.0/index.html>
- Python Tutorial - Tapa blanda. Guido Van Rossum (2012)

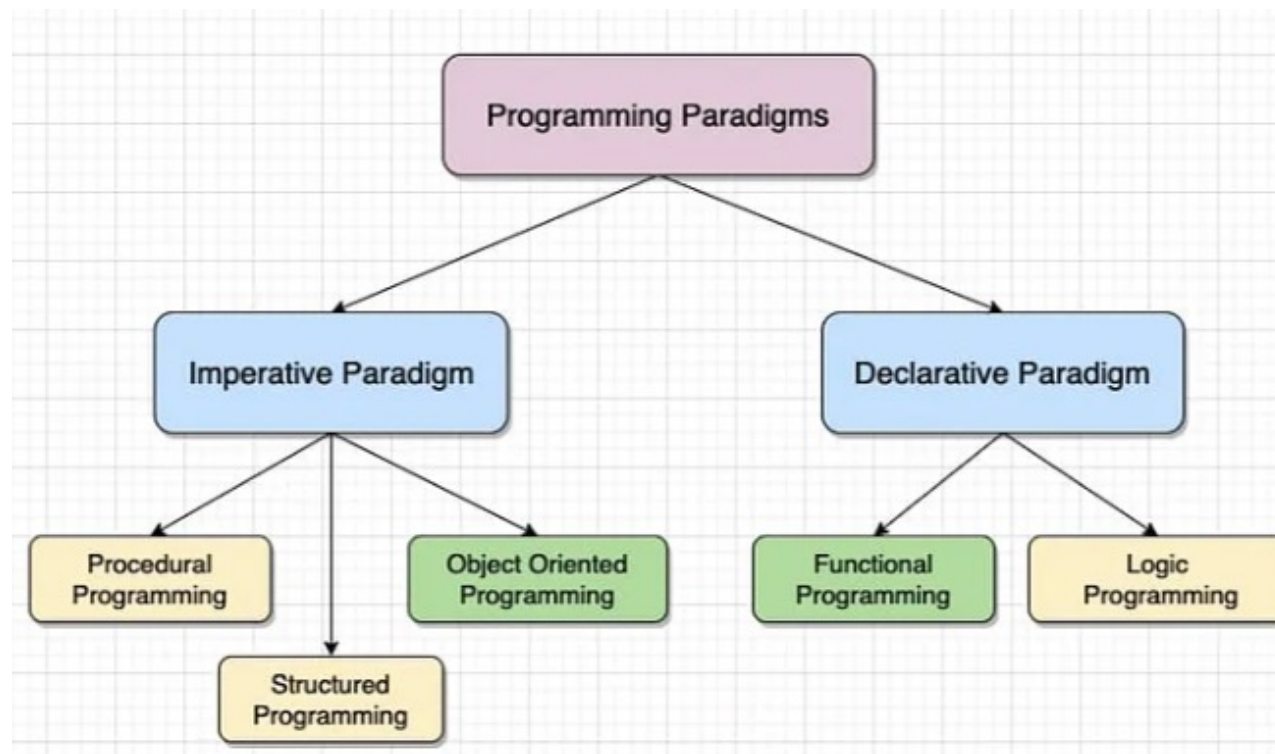
Outline

- Introduction
- Basic Principles of OOP
- Benefits of OOP

Introduction

- Programming is the process of creating a set of instructions that tell a computer how to perform a task
 - Purpose: to solve problems or perform specific tasks through automation
 - Components: code, algorithms, data structures, and logic
- A programming language is a formal language comprising a set of instructions that produce various kinds of output
 - Purpose: to communicate instructions to a computer
 - Examples: Python, Java, C++, JavaScript, Ruby
- A paradigm is a style or way of programming
 - Purpose: To provide a methodology for solving problems using programming languages
 - Types: Different paradigms offer different ways to structure and organize code

Programming paradigms



Programming paradigms

- Imperative : focuses on how to perform tasks
 - Procedural: the program is built from procedures or functions. It uses a modular approach, where the program is divided into small functions e.g. C, Pascal, Python
 - Structural: emphasizes the use of blocks, structured control flow constructs like loops and conditionals e.g. Ada, C
 - Object-Oriented: organizes code around objects and their interactions, e.g. C++, Python
 - Others: Event-Driven, Concurrent, Reactive...
- Declarative: focuses on what the program should accomplish, without explicitly specifying how to achieve the result e.g. SQL, HTML
 - Functional Paradigm: treats computation as the evaluation of functions, e.g. Lisp, Python
 - Logic Paradigm: uses logic statements to express computation, e.g. Prolog

Advantages (I)

1. **Reusability:** ability to use existing software components in new applications without significant modification
 - Impact: promotes efficient development practices by leveraging pre-existing code, reducing duplication, and ensuring consistency across projects
 - E.g: libraries and frameworks that provide reusable components for common functionalities
2. **Extensibility:** the ease with which a software system can be expanded and enhanced with new capabilities
 - Impact: ensures that the software can evolve over time, accommodating new requirements and technologies without extensive rewrites
 - E.g: plugin architectures and modular design patterns that allow for easy addition of new features

Advantages (II)

- **Maintainability:** the effort required to identify and fix defects, update functionalities, and improve performance
 - Impact: reduces long-term costs associated with software maintenance by facilitating easier updates and enhancements
 - E.g.: well-documented code and adherence to coding standards that make it easier for new developers to understand and work on the project
- **Usability:** the degree to which Sw can be used by specified consumers to achieve quantified objectives with effectiveness, efficiency, and satisfaction
 - Impact: increases user adoption and satisfaction by providing intuitive and accessible interfaces
 - E.g.: user interface design principles that focus on user-centric design and usability testing

Outline

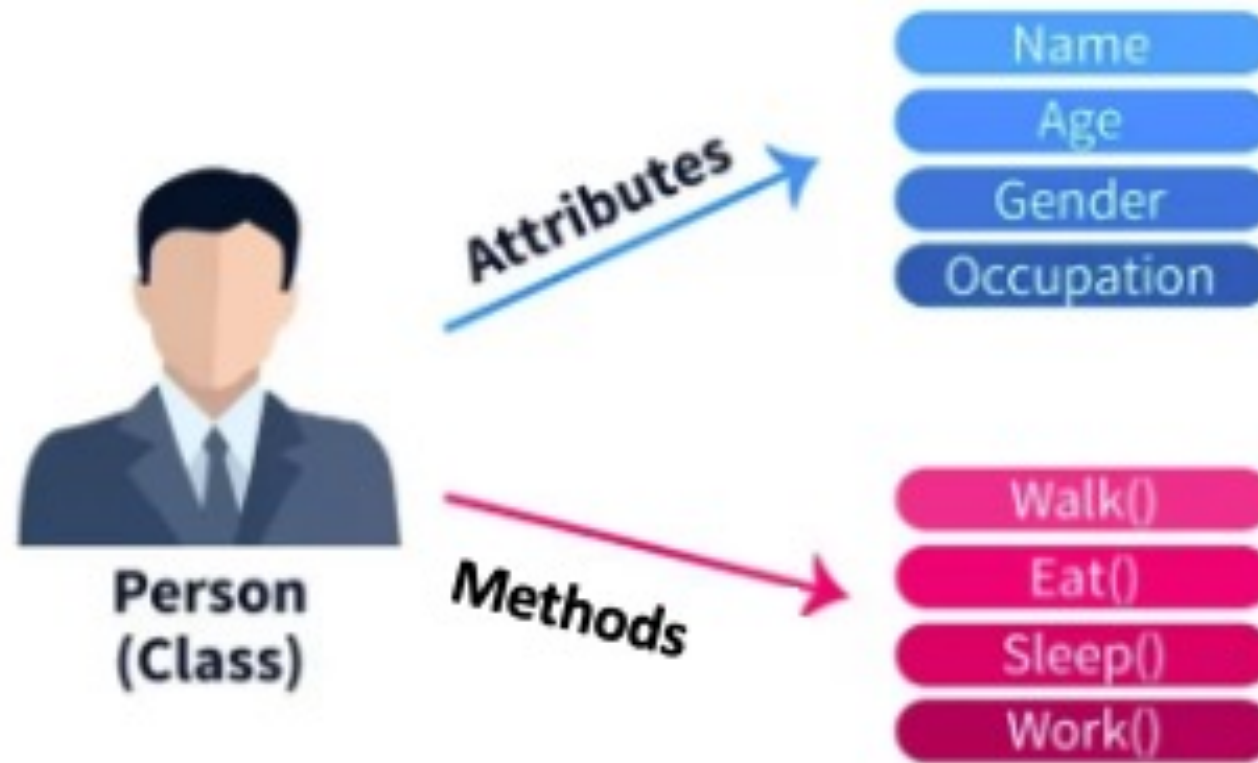
- Introduction
- **Basic Principles of OOP**
 - Classes and Objects
 - Encapsulation
 - Inheritance
 - Polymorphism
 - Abstraction
- Benefits of OOP

Class

- Is a blueprint or template that defines the structure and behavior (data and methods) that the objects of the class will have
- Is used to describe something in the world, such as occurrences, things, external entities, etc
- Is a collection of objects
- Once a class is declared, the programmer can create any number of objects of that class
- Defines properties and behavior of objects
- A user-defined datatype that behaves the same as the built-in datatypes

Class

Features that determine the qualities of an object

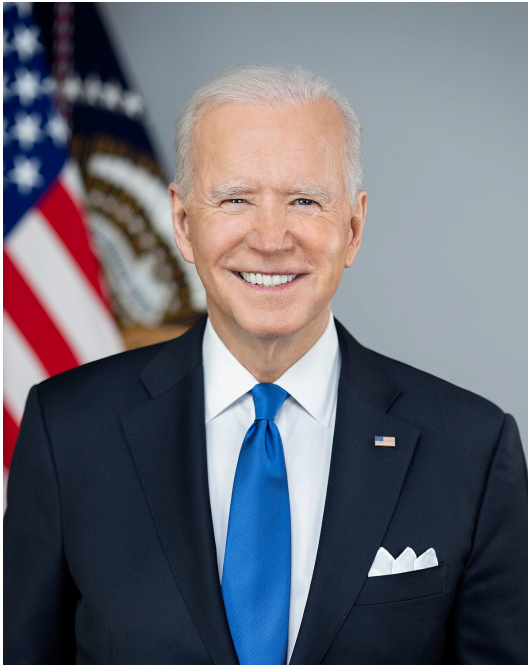
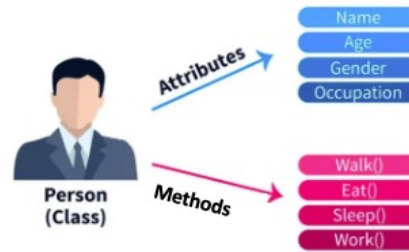


Function that perform operations

Object

- Is an instance of a class
- Every object contains some data and function(s) (called methods)
- These methods store data in variables and respond to the messages that they receive from other objects by executing their methods
- While a class is a logical structure, an object is a physical entity
- It is created from a class and can use the class's methods and properties

Objects



Summary

Class	Object
A class is a template for declaring and creating objects	An object is a class instance that allows programmers to use variables and methods from inside the class
Memory is not allocated to classes. Classes have no physical existence	When objects are created, memory is allocated to them in the heap memory
You can declare a class only once	A class can be used to create many objects
A class is a logical entity	An object is a physical entity
We cannot manipulate a class as it is not available in memory	Objects can be manipulated
Classes can have attributes and methods defined	Objects can have specific values assigned to their attributes

Example

Class

```
class Person:
    def __init__(self, name, age, gender, occupation):
        self.__name = name
        self.__age = age
        self.__gender = gender
        self.__occupation = occupation

    def get_name(self):
        return self.__name

    def walk(self):
        ...

    ...
```

Access Rules

- Method Execution and Access
 - A method is always executed for an object of its class
 - A method can access (invoke) all other members (attributes and methods) of its class
- Object Invocation
 - An object of a class can only invoke methods of its class
 - It can only respond to the messages for which it has been programmed

Outline

- Introduction
- Basic Principles of OOP
 - Classes and Objects
 - **Encapsulation**
 - Inheritance
 - Polymorphism
 - Abstraction
- Benefits of OOP

Encapsulation (I)

- Wrapping data (variables) and code (methods) together as a single unit
- Purpose is to protect the data from unauthorized access and modification

Encapsulation (II)

- It defines three access levels:
 1. Public: any data or method can be accessed by any method belonging to any class. This is the lowest level of data protection
 2. Protected: any data or method can be accessed only by that class or by any class that is inherited from it
 3. Private: any data or method can be accessed only by that class in which it is declared. This is the highest level of data protection

Outline

- Introduction
- Basic Principles of OOP
 - Classes and Objects
 - Encapsulation
 - **Inheritance**
 - Polymorphism
 - Abstraction
- Benefits of OOP

Inheritance

- A mechanism where a new class (*child class*) inherits the properties and behaviors of an existing class (*super class*)
- Purpose is to promote code reuse
- Allows the use and adaptation of the attributes and methods of the parent class to the child class
- Types of inheritance (both supported by Python)
 - Single: if the child class inherits from a single class
 - Multiple: if the child class inherits from more classes
- When we need to adapt an inheritanced method → Overriding

Example

Inheritance

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

    def speak(self):
        pass

class Dog(Animal):
    def speak(self):
        return 'Woof!'

class Cat(Animal):
    def speak(self):
        return 'Meow!'

...
```

Outline

- Introduction
- Basic Principles of OOP
 - Classes and Objects
 - Encapsulation
 - Inheritance
 - **Polymorphism**
 - Abstraction
- Benefits of OOP

Polymorphism

- The ability of different classes to respond to the same method call in different ways
- Purpose is to allow for methods to be used interchangeably among different classes

Example

Polymorphism

```
def make_animal_speak(animal):  
    print(animal.speak())  
  
dog = Dog('Rex')  
cat = Cat('Whiskers')  
  
make_animal_speak(dog)    # Output: Woof!  
make_animal_speak(cat)    # Output: Meow!
```

Outline

- Introduction
- Basic Principles of OOP
 - Classes and Objects
 - Encapsulation
 - Inheritance
 - Polymorphism
 - **Abstraction**
- Benefits of OOP

Abstraction

- The concept of hiding the complex implementation details and showing only the necessary features
- Purpose is to reduce complexity and increase efficiency

Example

Abstraction

```
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height
```

Outline

- Introduction
- Basic Principles of OOP
- **Benefits of OOP**

Benefits of OOP

- Modularity: code is organized into distinct classes
- Reusability: use existing code through inheritance
- Pluggability and Debugging Ease: replaceable components and easier debugging
- Maintainability: easier to manage and update code