Python OOP - Object Oriented Programming for Beginners

Basic Concepts and Advantages of OOP

- What is OOP?
- Name the four basic principles of Object-Oriented Programming.
- Why does OOP contribute to the modularity of a program?
- With OOP, we avoid code repetition because...

Classes

- What is a class?
- Why do we say that classes act like 'blueprints'?
- What type of word is typically used as the name of a class?
- Explain the syntax used to define a class in Python.
- Is 'class' a Python keyword?
- True or False: The name of a Python class should start with an uppercase letter by convention.
- How should you write the name of a class if that name has two words (e.g a checking account)?
- Classes define the .... and the ..... of an object.
- True or False: The body of a Python class has to be indented.

Instances and Instance Attributes

- What is an instance?
- How are classes related to instances?
- What is an instance attribute?

- Instance attributes belong to...
- True or False: If the value of an instance attribute of an instance is modified, that change will be applied to all instances of the same class.
- What is the role of __init__() in the creation and initialization of instance attributes?
- True or False: __init__() runs when an instance is created in the program and it is used to set the initial state of the instance.
- Explain the syntax used to define an instance attribute.
- What is the role of the keyword 'self' in a Python class?
- True or False: 'self' is the first parameter of the __init__() method.
- True or False: you can add more parameters after 'self' in __init__() .
- How do you create an instance in a Python program? Do you have to pass a value for the 'self' parameter? Explain this with an example.
- How do you set a default value for an argument of __init__() ?
- Should you add a space before and after the equal sign when you define a set value for a parameter?
- How can you access the value of an instance attribute outside and inside the class? Explain the syntax for both cases.
- How can you modify the value of an instance attribute? Explain the syntax.

Class Attributes

- What is a class attribute?
- Describe the differences between class attributes and instance attributes.
- True or False: Changing the value of a class attribute affects all instances of the class in the program.
- Where do we typically define class attributes in a Python class?
- How do you define a class attribute? Describe the syntax.
- How do you access a class attribute in a Python program? Describe the syntax and compare it with the code used to access an instance attribute.
- How do you update the value of a class attribute? Describe the syntax.

Encapsulation and Abstraction

- Describe the principle of Encapsulation and its importance in Object-Oriented Programming.
- Describe the principle of Abstraction and its importance in Object-Oriented Programming.
- Compare public and non-public attributes. Explain their differences.
- Are we really making an attribute "private" in Python when we add a leading underscore to its name?
- What is the main functional effect of adding a leading underscore to the name of an attribute? (Hint: It's related to import statements).
- Name the process that occurs when we add two leading underscores to the name of an attribute.
- Describe what happens during the process of Name Mangling and its main purpose.

Properties, Getters, and Setters

- What is a getter? Describe its purpose.
- Explain the naming convention established for getters.
- How do you call a getter to get the corresponding value? Explain the syntax.
- What is a setter? Describe its purpose.
- Explain the naming convention established for setters.
- How do you call a setter to set the corresponding value? Explain the syntax.
- What is a property in Python?
- Describe the advantages of defining a property in Python instead of using common getters and setters.
- What is @property?
- Describe the syntax used to define a property in Python with and without the @property decorator.
- Explain the advantages of using the @property decorator instead of the property() function.

Methods

- Methods define the ..... that objects of a class can perform.
- A method has access to the .... of the instance that calls it.
- Explain the syntax used to define a method.
- Name the first parameter of a method.
- How do you call a method? Explain the syntax and how you can pass values (arguments) to the method call.
- Is 'self' a required argument when you call a method?
- What is the role of 'self' in a method?
- How do you set a default argument?
- Can you call other methods of the same class in a method? If so, explain how.

Objects in Memory

- True or False: Everything in Python is an object.
- What happens in memory when you create an instance?
- What is the role of the id of an object?
- How can you get the id of an object?
- What does it mean when two variables have the same id when you call the id() function?
- What is the 'is' operator used for? Explain the two possible results with a examples.
- Explain the difference between the 'is' operator and ==.
- What is string interning?
- Describe the rules of string interning.
- In Python, objects are passed by...

Aliasing, Mutation, and Cloning

- What is aliasing?
- If two variables refer to the same object in memory, they are...
- Define mutation.
- What is the key characteristic of an object of a mutable data type?
- Describe the advantages and disadvantages of mutability.
- Describe the risks of creating aliases for a mutable object (e.g. a list).
- What is cloning?
- Describe the difference between a clone and an alias.
- If you modify a clone, will the original object be modified?

Inheritance (Attributes)

- Why do we need inheritance in Python?
- Describe the advantages of inheritance.
- True or False: With inheritance we can reduce code repetition.
- Define superclass and subclass. How are they related? Explain this with an example.
- Describe how you can make a class inherit attributes from another class.
- Describe the line of code that you need to add to __init__() to make a subclass inherit the attributes of its superclass (assume that the subclass has its own __init__() method defined).

Inheritance (Methods)

- Explain why it is helpful to make a class inherit methods from another class.
- Describe the syntax used to make a class inherit methods from another class.
- What is method overriding? When does it occur?
- How can you call the method from the superclass when that method has been overridden in the subclass?
- What is the role of super() in method overriding?

Working with Multiple Files

- What is a module?
- What is an import statement?
- Where do we usually write import statements in the program?
- Describe the naming conventions for modules.
- Explain the syntax used to import all the elements of a module.
- Explain the syntax used to import only one or a few elements of a module.
- Explain the syntax used to import a module and access it with a different name.

Docstrings

- What is a Docstring in Python? Why are they important?
- Describe the differences between a docstring and a comment.
- Docstrings are linked to the elements they describe through a specific attribute. Name this attribute.
- What function can you use to read the docstring of an element of a program in the Python shell?
- True or False: Docstrings can be converted into documentation.
- Explain the guidelines presented to document classes, attributes, properties, and methods.

Special Methods

- What is a special method?
- Why are these methods "different" from other common methods?
- Describe the characteristic syntax of their names.
- Explain the purpose of these special methods:
    * __str__
    * __len__
    * __getitem__
    * __add__
    * __bool__
    * __lt__
    * __le__
    * __eq__
    * __ne__
    * __gt__
    * __ge__