OBJECT ORIENTED PROGRAMMING IN PYTHON & INTRODUCTION

Object Oriented Programming

12/06/2023

Why Class & Object???

Class & Object in Python

Introduction

- A Class is a Blueprint or a template for creating objects, while an object is an instance of a class.
- A Class defines a set of attributes (variables) and methods (functions) that the objects of that class will have.
- A Class acts as a container for data and behaviour.
- A Class defines the structure and behaviour that the objects should possess.

init() method in Python

- The ‘_init_’ method in Python is a special method, also known as the constructor.
- It is automatically called when an object is created from a class.
- The The purpose of the ‘_init_’ method is to initialise the attributes (variables) of the object.

```
class Person:

    def __init__(self, name, age):

        self.name = name

        self.age = age

person1 = Person("Alice", 25)

print(person1.name)  # Output: Alice

print(person1.age)   # Output: 25
```

Constructor and Self in Python

- A constructor is a special method that is used to initialise an object of a class. It is typically named '_init_' and is called automatically when you create a new instance of a class.
- The 'Self' parameter in the constructor (and other methods in the class) refers to the instance of the class itself.

```
class Person:

    def __init__(self, name, age):

        self.name = name

        self.age = age

person1 = Person("Alice", 25)

print(person1.name)  # Output: Alice

print(person1.age)   # Output: 25
```

Assignment - 1

Q1) Create a class 'Product' has a parameterised constructor '__init__()' that takes 'name','price' and 'quantity' as parameters. inside the constructor these parameters are assigning to the corresponding instance variables 'self.name','self.price' and 'self.quantity'.
The class also includes a 'display_info()' that prints information about the product, including its name, price and quantity.

Assignment - 2

Q1) In a class representing a Book, how can a non-parameterized constructor be used to set default values for attributes such as title, author, and publication year?(initialise variables within a non parameterised constructor)

Q2)How a non-parameterized constructor can be used in a class representing a Game to set default values for attributes such as score, level, and player name.(initialise variables as a class variables)

Types of Variables in Python

Instance and Class(static) Variable

- Instance Variables:
  - Instance variables are unique to each instance of a class.
  - They are defined inside the methods of a class, usually within the constructor (_init_ () method), and are prefixed with the self parameter.
  - Instance variables are accessed using the instance name (‘self.instance_variable’) within the class methods or using the instance name outside the class.
  - Instance variables are used to store data specific to each object/instance of the class.

- Class(static) Variables:
  - Class variables are shared among all instances of a class.
  - They are defined outside of any method in the class, typically at the top of the class definition.
  - Class variables are accessed using the class name (ClassName.Variable_Name) within the class methods or using the class name outside the class.
  - Class variables are used to store data that is common to all objects/instances of the class.

Types of Variables in Python

Instance and Class(static) Variable

```
class Person:

    def __init__(self, name, age):

        self.name = name

        self.age = age

person1 = Person("Alice", 25)

person2 = Person("Bob", 30)

print(person1.name)  # Output: Alice

print(person2.age)   # Output: 30
```

```
class Circle:
    pi = 3.14159  # Class variable

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

    def calculate_area(self):
        return Circle.pi * self.radius * self.radius

circle1 = Circle(5)
circle2 = Circle(3)

print(circle1.calculate_area())  # Output: 78.53975
print(circle2.calculate_area())  # Output: 28.27431
print(Circle.pi)                 # Output: 3.14159
```

Types of Methods in Python

Instance, Class and Static Methods

- Instance Methods:
  - They operate on instances of the class and can access and modify the instance variables.
  - Instance methods are the most common type of methods in Python classes.
  - They operate on instances of the class and can access and modify the instance variables.
  - Instance methods are defined with the self parameter as the first parameter, which refers to the instance of the class on which the method is called.
  - They can be used to perform actions or calculations specific to each instance.

- Class Methods:
  - Class methods are methods that operate on the class itself rather than instances of the class.
  - They are defined using the @classmethod decorator and have the class as the first parameter, conventionally named ‘cls’.
  - Class methods are commonly used for operations that involve the class itself, such as creating alternative constructors or accessing class variables.

- Static Methods:
  - Static methods are methods that do not operate on instances or the class itself.
  - They are defined using the @staticmethod decorator and do not have the self or cls parameter.
  - Static methods are independent of the class and its instances. They don't have access to instance variables or class variables.
  - They are commonly used for utility functions that are related to the class but don't depend on its state.

Types of Methods in Python

Instance, Class and Static Methods

Instance Methods

```
class Circle:
    def __init__(self, radius):
        self.radius = radius

    def calculate_area(self):
        return 3.14159 * self.radius *
self.radius

circle = Circle(5)
print(circle.calculate_area())  #
Output: 78.53975
```

```
class Rectangle:
    width = 0
    height = 0

    def __init__(self, width, height):
        self.width = width
        self.height = height

    @classmethod
    def create_square(cls, side):
        return cls(side, side)

rectangle = Rectangle.create_square(4)
print(rectangle.width)   # Output: 4
print(rectangle.height)  # Output: 4
```

```
class MathUtils:
    @staticmethod
    def add(a, b):
        return a + b

result = MathUtils.add(5, 3)
print(result)  # Output: 8
```

Inner Class in Python

- In Python, an inner class is a class that is defined within another class. It is also known as a nested class.
- The inner class is typically intended to be used only within the outer class and is often used to represent a specialised or related concept.

```
    class OuterClass:
    def __init__(self):
        self.outer_var = "Outer variable"

    class InnerClass:
        def __init__(self):
            self.inner_var = "Inner variable"

        def inner_method(self):
            print("This is the inner method")

outer_obj = OuterClass()
inner_obj = outer_obj.InnerClass()
inner_obj.inner_method()
print(inner_obj.inner_var)
print(outer_obj.outer_var)
```