Seif Hendawy

Python Day 4 Report

Q2 . Summarize DataClass in Python

dataclasses are a decorator and functions in the dataclasses module that automatically add special methods to user-defined classes.

They reduce the code by automatically generating methods like __init__, __repr__, and __eq__.

- Basic Usage:

    To create a dataclass, we use the @dataclass decorator.
    Example

    ```
    from dataclasses import dataclass
    @dataclass
    class Point:
        x: int
        y: int
    ```

  This automatically creates an __init__ method that initializes x and y.

- Default Values:

    We can provide default values for attributes.
    Example:

    ```
    @dataclass
    class Point:
        x: int = 0
        y: int = 0
    ```

- Immutability:

    The frozen=True parameter to make instances immutable.
    Example:

    ```
    @dataclass(frozen=True)
    class Point:
        x: int
        y: int
    ```

- Comparison Methods:

    dataclasses automatically generate comparison methods (__eq__, __lt__, etc.) if order=True is set.
    Example:

    ```
    @dataclass(order=True)
    class Point:
    ```

```
x: int
y: int
```

- Post-Initialization:
  Use the __post_init__ method for additional initialization after the default __init__.
  Example:
  ```
  @dataclass
  class Point:
      x: int
      y: int

      def __post_init__(self):
          self.distance = (self.x**2 + self.y**2) ** 0.5
  ```
- Inheritance:
  dataclasses support inheritance, allowing child classes to extend parent dataclasses.
  Example:
  ```
  @dataclass
  class Point3D(Point):
      z: int
  ```

Data classes also have advanced features like field customization using the field function, which allows specifying default values, default factories, and metadata.

dataclasses are useful for creating data structures, configuration objects, and other scenarios where you need to store data with minimal boilerplate code.

Q3. Summarize Multi Inheritance in Python
Method Resolution Order (MRO)

Multiple inheritance in Python allows a class to inherit attributes and methods from more than one parent class. This can be powerful but also complex, especially when dealing with method conflicts. Python uses the Method Resolution Order (MRO) to resolve these conflicts and determine the order in which methods are inherited.

Multiple Inheritance: A class can inherit from multiple parent classes.

```
class A:
    def greet(self):
        print("Hello from A")

class B:
    def greet(self):
        print("Hello from B")

class C(A, B):
    pass

obj = C()
obj.greet()  # Output: "Hello from A" (due to MRO)
```

Method Resolution Order (MRO):

- MRO is the order in which Python searches for methods in a class hierarchy.
- It ensures that a method is only called once, even if it exists in multiple parent classes.
- MRO follows the C3 Linearization algorithm, which ensures a consistent and predictable order.
- We can see the MRO of a class using the .__mro__ attribute or the mro() method.

  ```
  print(C.__mro__)
  # Output: (<class '__main__.C'>, <class '__main__.A'>, <class '__main__.B'>, <class 'object'>)
  ```

  How MRO Works:

  - Python constructs a linear order of classes based on inheritance.
  - The order follows these rules:
  - Subclasses come before base classes.
  - The order of parent classes in the inheritance list is preserved.
  - No class is visited more than once.

A common issue in multiple inheritance where a class inherits from two classes that both inherit from a common base class(Diamond Problem).
MRO ensures that the class is only visited once, avoiding ambiguity.

```
class A:
    def greet(self):
        print("Hello from A")

class B(A):
    pass

class C(A):
    pass

class D(B, C):
    pass

obj = D()
obj.greet()  # Output: "Hello from A" (resolved by MRO)
```