Summarize Multi Inheritance in Python Method Resolution Order (MRO):
Multiple inheritance allows a class to inherit from more than one parent class.
Syntax:
class ChildClass(ParentClass1, ParentClass2):
    Pass
Method Resolution Order (MRO)

- **MRO** determines the order in which Python looks for methods or attributes in classes when inheritance is involved.
- Python uses the **C3 Linearization (or CPL)** algorithm to compute MRO. This ensures:
    1. **Consistency**: The order is deterministic and avoids ambiguity.
    2. **Depth-First Search**: A child class is searched before its parent.
    3. **Left-to-Right Priority**: For multiple parents, the left-most parent in the class declaration is searched first.

Example:

```
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" (based on MRO)
```

Dictionary Comprehension Example:

General Form:

{key_expression: value_expression for item in iterable if condition}

Examples:

Square numbers

```
squares = {x: x**2 for x in range(1, 6)}

print(squares)
```

Conditional Dictionary Comprehension:

```
cubes = {x: x**3 for x in range(1, 11) if x % 2 == 0}

print(cubes)
```

Transforming an Existing Dictionary:

```
# Convert all keys to uppercase

original_dict = {'one': 1, 'two': 2, 'three': 3}

uppercase_dict = {k.upper(): v for k, v in
original_dict.items()}

print(uppercase_dict)
```

**********Data class:

A **data class** in Python is a special type of class that is designed to store data more efficiently with **less**

boilerplate code. It was introduced in **Python 3.7** through the `@dataclass` decorator from the `dataclasses` module.

Without a data class, defining a class to store data requires writing:

- `__init__()` to initialize attributes
- `__repr__()` for string representation
- `__eq__()` for equality checks
- Optional methods like `__hash__()` and `__post_init__()`

A `@dataclass` automates all of this for you!

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self):
        return f"Point({self.x}, {self.y})"

    def __eq__(self, other):
        if isinstance(other, Point):
            return self.x == other.x and self.y == other.y
        return False

p1 = Point(2, 3)
```

```
p2 = Point(2, 3)

print(p1)         # Output: Point(2, 3)

print(p1 == p2)   # Output: True
```

Using a Data Class:

```
@dataclass

class Point:

    x: int

    y: int


p1 = Point(2, 3)

p2 = Point(2, 3)

print(p1)         # Output: Point(x=2, y=3)

print(p1 == p2)   # Output: True
```

1. `__init__()` - Automatically generates an initializer.
2. `__repr__()` - Provides a readable string representation.
3. `__eq__()` - Supports equality comparison.
4. `__hash__()` - Adds hashing support if `frozen=True`.
5. `__post_init__()` - Hook to customize initiali