Key Concepts: Method Resolution Order (MRO) and Linearization

What is Method Resolution Order (MRO)?

- **Definition**:
  - MRO determines the order in which methods and attributes are searched in a class hierarchy.
  - It defines the sequence used to resolve a method or attribute when a child class inherits from one or more parent classes.
- **Purpose**:
  - Ensures a predictable and logical resolution of methods.
  - Helps manage complexity in multi-level and multiple inheritance scenarios.
  - Avoids conflicts and ambiguity when multiple classes define the same method or attribute.

Types of Inheritance in Python

1. **Simple Inheritance**:
   - A child class inherits from a single parent class.
2. **Multiple Inheritance**:
   - A child class inherits from more than one parent class.
3. **Multi-level Inheritance**:
   - Inheritance occurs across multiple levels.
   - Example: Class C inherits from Class B, which inherits from Class A.
4. **Hierarchical Inheritance**:
   - Multiple child classes inherit from a single parent class.
5. **Hybrid Inheritance**:
   - A combination of two or more types of inheritance.

How MRO Works

- **Linearization**:

- MRO creates a linear path (or sequence) through the class hierarchy.
- In Python:
  - **Single inheritance**: Bottom-to-top search.
  - **Multiple inheritance**: Bottom-to-top, **left-to-right** search in the class declaration.

- **C3 Linearization Algorithm**:
  - Used in Python 3 for new-style classes.
  - Ensures monotonicity (no skipping of direct parents).
  - Follows:
    a. Local precedence order: Processes the class itself before its parents.
    b. Left-to-right order in class declarations.
    c. Resolves conflicts by considering the order in which parents are defined.

Methods to Find MRO

1. **`<class>.mro()`** :
   - Returns a list of classes in the MRO sequence.
   - Example:

```
class A:
    pass
class B(A):
    pass
class C(B):
    pass

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

2. **`help(<class>)`** :
   - Provides detailed information, including MRO.
   - Example:

```
help(C)
```

Example: Single and Multi-level Inheritance

```
class A:
    num = 5
```

```
class B(A):
    num = 9

class C(B):
    pass

# MRO for Class C
print(C.mro())
# Output: [<class '__main__.C'>, <class '__main__.B'>, <class '__main__.A'>, <class 'object'>]

# Accessing num
c = C()
print(c.num)  # Output: 9 (inherited from Class B)
```

Example: Multiple Inheritance

```
class X:
    def greeting(self):
        return "Hello from X"

class Y:
    def greeting(self):
        return "Hello from Y"

class Z(X, Y):
    pass

z = Z()
print(z.greeting())  # Output: "Hello from X"
print(Z.mro())
# Output: [<class '__main__.Z'>, <class '__main__.X'>, <class '__main__.Y'>, <class 'object'>]
```

- **Explanation**:
  - Class `Z` inherits from `X` and `Y`.
  - MRO determines that `X` is searched before `Y`.

Conclusion

- **MRO Simplifies Complexity**:
  - Ensures consistent behavior in multi-level and multiple inheritance.
  - Avoids ambiguity when resolving methods and attributes.

- **Tools for Understanding MRO**:
  - Use `mro()` for the sequence of resolution.
  - Use `help()` for detailed class and MRO information.

By understanding MRO, you can effectively navigate and manage inheritance relationships in Python, even as they grow more complex.