**Inheritance:**

* **Class Inheritance:**
  + Python supports class inheritance, allowing a new class (derived class) to inherit attributes and methods from an existing class (base class).
  + Syntax: **class DerivedClassName(BaseClassName):**
* **Base Class Scope:**
  + The base class name must be defined in a scope containing the derived class definition.
  + Other expressions can be used in place of a base class name.
* **Execution of Derived Classes:**
  + Execution of a derived class definition is similar to a base class.
  + The base class is remembered when the class object is constructed.
* **Method Resolution:**
  + Method references in derived classes are resolved by searching in the class attribute, descending the chain of base classes if necessary.
  + Derived classes can override methods of their base classes.
* **Calling Base Class Method:**
  + To call a base class method directly from a derived class, the syntax is **BaseClassName.methodname(self, arguments)**.
* **Built-in Functions for Inheritance:**
  + **isinstance(obj, int)** checks if an instance's type is **int**.
  + **issubclass(bool, int)** checks if a class is a subclass of another.
* **Multiple Inheritance:**
  + Python supports multiple inheritance with the syntax: **class DerivedClassName(Base1, Base2, Base3):**
* **Attribute Search Order in Multiple Inheritance:**
  + The search for attributes inherited from parent classes is depth-first, left-to-right, without searching twice in the same class where there's an overlap in the hierarchy.
* **Dynamic Method Resolution Order:**
  + The method resolution order changes dynamically to support cooperative calls to **super()**.
  + Necessary due to diamond relationships in multiple inheritance.

**Private Variables:**

* **Private Variables in Python:**
  + True private variables don't exist, but a convention is followed by using a single leading underscore (**\_**) or name mangling with double leading underscores (**\_\_**).
  + Name mangling is done without regard to the syntactic position of the identifier.
* **Name Mangling:**
  + Any identifier of the form **\_\_spam** is textually replaced with **\_classname\_\_spam**.
  + Helps avoid name clashes with subclasses.
* **Getter and Setter Methods:**
  + Getter method (**get\_private\_variable**) retrieves the value of the private variable.
  + Setter method (**set\_private\_variable**) allows modification of the private variable.
* **Instance Creation and Accessing Variables:**
  + An instance of the **MyClass** class is created.
  + Public, protected, and private variables are accessed and modified using various methods.

**Multiple Inheritance Example:**

* **Platypus Class:**
  + Demonstrates a class (**Platypus**) with multiple inheritance from **Animal**, **Mammal**, and **Bird**.
  + Initializes base classes' attributes in its own constructor and provides an implementation for the **speak** method.
* **Instance Creation and Method Access:**
  + Creates an instance of **Platypus** (**perry**) and accesses methods from all three base classes.
  + Note: Careful design is required to avoid potential issues such as the diamond problem in multiple inheritance.