Global Variable:

```
x = 10

def change():
   global x
   x = 20
change()
print(x)  # Output: 20
```

Class and Object:

Program1--

```
class MyClass:
 x = 5
print(MyClass)
```

Program2--

```
class MyClass:
 x = 5
p1 = MyClass()
print(p1.x)
```

Constructor Types in Python:

1. **Default Constructor** – No parameters except `self`.

```
class Hello:
    def __init__(self):
        print("Hello, world!")

obj = Hello()  # Output: Hello, world!
```

2. **Parameterized Constructor** – Takes arguments to initialize the object.

```
class Student:
    def __init__(self, name):
        self.name = name
```

```
s1 = Student("John")
print(s1.name)  # Output: John
```

Class and instance variable:

```
class Dog:

    species = "Canine"  # class variable


    def __init__(self, name):

        self.name = name  # instance variable


dog1 = Dog("Tommy")

dog2 = Dog("Buddy")

print(dog1.name)

print(dog2.name)

print(dog1.species)  # Canine

print(dog2.species)  # Canine

Dog.species = "Animal"

print(dog1.species)  # Animal

print(dog2.species)  # Animal
```

Creating Function for Multiple Fetch Values:

```
class Student:

    def __init__(self, name, grade):

        self.name = name

        self.grade = grade


    def get_details(self):

        return self.name, self.grade
```

```
s1 = Student("John", "A")
name, grade = s1.get_details()
print (name, grade )
```

Object as a list:

```
class Student:
    def __init__(self, name):
        self.name = name


s1 = Student("Alice")
s2 = Student("Bob")
students = [s1, s2]
for s in students:
    print(s.name)
```

Inheritance:

```
class Animal:
    def speak(self):
        print("Animal Speaking")
#child class Dog inherits the base class Animal
class Dog(Animal):
    def bark(self):
        print("dog barking")
d = Dog()
d.bark()
d.speak()
```

a= Animal()

a.speak()

a.bark()

Inheritance Type	Description
Single	One child, one parent
Multiple	One child, multiple parents
Multilevel	Chain of inheritance
Hierarchical	One parent, multiple children
Hybrid	Combination of multiple types (like multiple + multilevel).

Polymorphism:

1. Polymorphism with Functions and Objects

You can use the same function name for different object types.

```
class Dog:
    def speak(self):
        return "Woof!"

class Cat:
    def speak(self):
        return "Meow!"

def animal_sound(animal):
    print(animal.speak())

dog = Dog()
cat = Cat()

animal_sound(dog)  # Woof!
animal_sound(cat)  # Meow!
```

2. Polymorphism with Inheritance (Method Overriding)

This is a classic OOP approach where subclasses override methods from the base class.

```
class Animal:
    def speak(self):
        return "Some sound"

class Dog(Animal):
    def speak(self):
```

```
        return "Bark"

class Cat(Animal):
    def speak(self):
        return "Meow"

animals = [Dog(), Cat(), Animal()]

for animal in animals:
    print(animal.speak())
```

Python Access Modifier Types:

Modifier	Syntax	Access Level	Example
Public	`variable`	Accessible **everywhere**	`self.name = "Alice"`
Protected	`_variable`	Accessible **within class and subclasses** (convention only)	`self._age = 25`
Private	`__variable`	Accessible **only within class** (name mangling)	`self.__salary = 5000`

Public Members

```
class Student:
   def __init__(self):
      self.name = "Alice"  # Public

s = Student()
print(s.name)  # Accessible
```

Protected Members (`_single_underscore`)

- Intended for internal use.
- Can still be accessed from outside, but should **not** be.

```
class Student:
   def __init__(self):
      self._age = 20  # Protected

s = Student()
print(s._age)  # Technically allowed, but not recommended
```

Private Members (`__double_underscore`)

- **Name mangling** is used (`_ClassName__var`), making them harder to access directly.
- Use **getters and setters** to safely access.

```
class Student:
    def __init__(self):
        self.__marks = 90  # Private

    def get_marks(self):
        return self.__marks

s = Student()
# print(s.__marks)      # Error
print(s.get_marks())    # 90

# Still accessible like this (not recommended):
print(s._Student__marks)  # 90 (name mangling)
```