

Day 24: Polymorphism & Scope

The word "polymorphism" means "many forms", and in programming it refers to methods/functions/operators with the same name that can be executed on many objects or classes.

Function Polymorphism

An example of a Python function that can be used on different objects is the `len()` function.

```
# String
# For strings len() returns the number of characters
x = "Hello World!"

print(len(x))

# Tuple
# For tuples len() returns the number of items in the tuple
mytuple = ("apple", "banana", "cherry")

print(len(mytuple))

# Dictionary
# For dictionaries len() returns the number of key/value pairs in the dictionary
thisdict = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
}

print(len(thisdict))
```

Class Polymorphism

Polymorphism is often used in Class methods, where we can have multiple classes with the same method name.

For example, say we have three classes: `Car`, `Boat`, and `Plane`, and they all have a method called `move()`.

```
class Car:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

    def move(self):
        print("Drive!")

class Boat:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

    def move(self):
        print("Sail!")

class Plane:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

    def move(self):
        print("Fly!")

car1 = Car("Ford", "Mustang")    #Create a Car object
boat1 = Boat("Ibiza", "Touring 20") #Create a Boat object
plane1 = Plane("Boeing", "747")  #Create a Plane object

for x in (car1, boat1, plane1):
    x.move()
```

Inheritance Class Polymorphism

```
class Vehicle:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

    def move(self):
        print("Move!")

class Car(Vehicle):
    pass

class Boat(Vehicle):
    def move(self):
        print("Sail!")

class Plane(Vehicle):
    def move(self):
        print("Fly!")

car1 = Car("Ford", "Mustang")    #Create a Car object
boat1 = Boat("Ibiza", "Touring 20") #Create a Boat object
plane1 = Plane("Boeing", "747")  #Create a Plane object

for x in (car1, boat1, plane1):
    print(x.brand)
    print(x.model)
    x.move()
```

Scope

A variable is only available from inside the region it is created. This is called **scope**.

Local Scope

A variable created inside a function belongs to the *local scope* of that function, and can only be used inside that function.

```
def myfunc():  
    x = 300  
    print(x)  
  
myfunc()
```

Function Inside Function

As explained in the example above, the variable `x` is not available outside the function, but it is available for any function inside the function.

```
def myfunc():  
    x = 300  
    def myinnerfunc():  
        print(x)  
        myinnerfunc()  
  
myfunc()
```

Global Scope

A variable created in the main body of the Python code is a global variable and belongs to the global scope.

Global variables are available from within any scope, global and local.

```
x = 300  
  
def myfunc():  
    print(x)
```

```
myfunc()
```

```
print(x)
```

Naming Variables

If you operate with the same variable name inside and outside of a function, Python will treat them as two separate variables, one available in the global scope (outside the function) and one available in the local scope (inside the function).

```
x = 300
```

```
def myfunc():
```

```
    x = 200
```

```
    print(x)
```

```
myfunc()
```

```
print(x)
```

Global Keyword

If you need to create a global variable, but are stuck in the local scope, you can use the `global` keyword.

The `global` keyword makes the variable global.

```
def myfunc():
```

```
    global x
```

```
    x = 300
```

```
myfunc()
```

```
print(x)
```

Also, use the `global` keyword if you want to make a change to a global variable inside a function.

```
x = 300

def myfunc():
    global x
    x = 200

myfunc()

print(x)
```

Nonlocal Keyword

The `nonlocal` keyword is used to work with variables inside nested functions.

The `nonlocal` keyword makes the variable belong to the outer function.

```
def myfunc1():
    x = "Jane"
    def myfunc2():
        nonlocal x
        x = "hello"
    myfunc2()
    return x

print(myfunc1())
```