8. Programación orientada a objetos

La programación orientada a objetos es un paradigma basado en el concepto de que todo es un "objeto" que tiene datos y código. Los datos se representan como atributos o propiedades y el código como procedimientos o métodos.

Los metodos están asociados a objetos, generalmente mediante el uso de las palabras reservadas **this** y **self**.

El origen de esta metodología, en su representación actual, nace en en los 50 en el MIT dentro del laboratorio de inteligencia artificial.

Simula es considerado el primer lenguaje de programación orientado a objetos que introduce los términos **clase** y **objeto** así como **herencia** y **dynamic binding**

8.1 Conceptos de programación orientada a obetos

Clases

Definición de formato de datos, y acciones disponibles para un tipo específico de objetos.

En Python una mezcla de los mecanismos de clases encontrados en C++ y Modula-3. Las clases de Python proveen todas las características normales de la Programación Orientada a Objetos: el mecanismo de la herencia de clases permite múltiples clases base, una clase derivada puede sobre escribir cualquier método de su(s) clase(s) base, y un método puede llamar al método de la clase base con el mismo nombre. Los objetos pueden tener una cantidad arbitraria de datos de cualquier tipo. Igual que con los módulos, las clases participan de la naturaleza dinámica de Python: se crean en tiempo de ejecución, y pueden modificarse luego de la creación.

```
class Complex:
    def __init__(self, realpart, imagpart):
        self.r = realpart
        self.i = imagpart
    def print():
        return('{} + {}i'.format(self.r, self.i))
```

8.5 Crear y administrar el comportamiento de las instancias de clase

Objetos

Instancisa de clases en memoria.

```
# Ejemplo de objeto instancia
numero = Complex(3.0, -4.5)
```

```
print(x.r) #3.0
print(x.y) # -4.5
```

En python también existen los objetos método.

```
# Ejemplo de objeto instancia
numero = Complex(3.0, -4.5)
numeroTexto = numero.print
print(numeroTexto())
```

Namesapces

```
def scope_test():
    def do_local():
        spam = "local spam"

    def do_nonlocal():
        nonlocal spam
        spam = "nonlocal spam"

    def do_global():
        global spam
        spam = "global spam"

    spam = "test spam"
    do_local()
    print("After local assignment:", spam)
    do_nonlocal()
    print("After nonlocal assignment:", spam)
    do_global()
    print("After global assignment:", spam)

scope_test()
print("In global scope:", spam)
```

```
After local assignment: test spam
After nonlocal assignment: nonlocal spam
After global assignment: nonlocal spam
In global scope: global spam
```

8.2 Tipos de datos definidos por el usuario

getters() y setters()

```
class Label:
    def __init__(self, text, font):
        self.set_text(text)
        self.font = font
```

```
    def getText(self):
        return self._text

    def setText(self, value):
        self._text = value.upper()  # Attached behavior
```

8.3 Herencia y encapsulación

Encapsulación

La encapsulación es la abstracción que permite hacer referencia a métodos de un objeto sin tener que referenciar su comportamiento interno. Esto permite, en python, la introducción de rutinas externas de cómputo en otros lenguajes de programación.

Composición y herencia

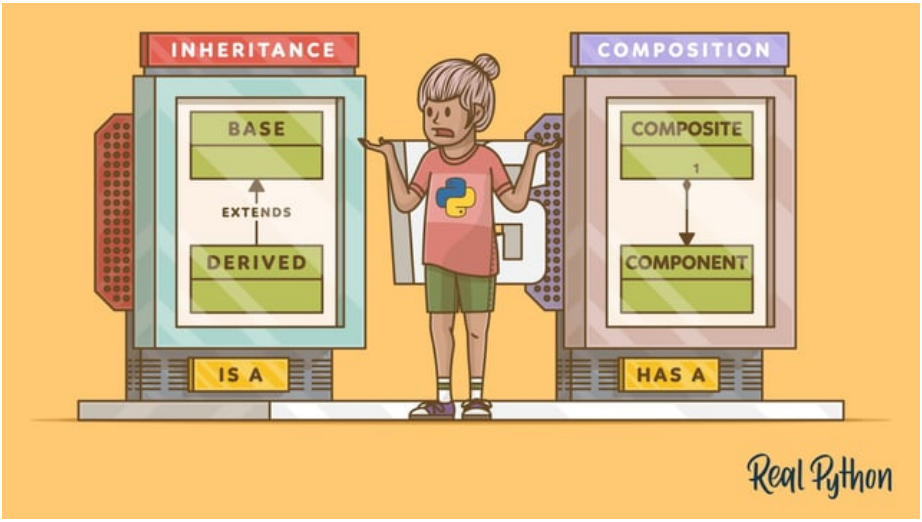

Figure 1: Herencia vs Composición

8.4 Polimorfismo

Es la capacidad de proveer una sola interfaz para entidades de distintos tipos o el uso de un solo simbolo para represetnar diferentes tipos de entidades.

```
class India():
    def capital(self):
        print("New Delhi is the capital of India.")
```

```
    def language(self):
        print("Hindi is the most widely spoken language of India.")

    def type(self):
        print("India is a developing country.")

class USA():
    def capital(self):
        print("Washington, D.C. is the capital of USA.")

    def language(self):
        print("English is the primary language of USA.")

    def type(self):
        print("USA is a developed country.")

obj_ind = India()
obj_usa = USA()
for country in (obj_ind, obj_usa):
    country.capital()
    country.language()
    country.type()
```

8.6 Metaclases

Para mayor referencia vease el siguiente thread en stackoverflow.

Ejecicio en clase

Realice un programa en python que defina números complejos en sus representación binómica y sus operaciones básicas de:

- Imprimir
- Multiplicación por escalar
- Módulo
- Fase
- Conjugación
- Conversión a polar

Fuentes

- Python Docs
- GeeksForGeeks - Polymorphism