Educa2 – Clases y cursos de matemáticas y Programación

Curso Básico de Programación en Python

Preparador: Alexanyer Naranjo

Guía Teórica N°7

Programación Orientada a Objetos (POO)

Python es un lenguaje de programación orientado a objetos. Hasta ahora, hemos utilizado una serie de clases incorporadas para mostrar ejemplos de datos y estructuras de control. Una de las características más poderosas en un lenguaje de programación orientado a objetos es la capacidad de permitir a un programador (solucionador de problemas) crear nuevas clases que modelen los datos necesarios para resolver el problema.

Algunas particularidades de POO en Python son las siguientes:

- Todo es un objeto, incluyendo los tipos y clases.
- Permite herencia múltiple.
- No existen métodos ni atributos privados.
- Los atributos pueden ser modificados directamente.
- Permite la sobrecarga de operadores.
- Permite la creación de nuevos tipos de datos.

A continuación, se procede a definir algunos conceptos necesarios para entender la POO:

Objeto: Los ***objetos*** son abstracción de Python para data. Toda la data en un programa Python es representado por objectos o por relaciones entre objectos. (En cierto sentido, y en el código modelo de Von Neumann de una “computadora almacenada del programa” también es un código representado por los objetos.)

Cada objeto tiene una identidad, un tipo y un valor. Una identidad de objecto nunca cambia una vez es creada; usted puede pensar eso como la dirección de objeto en memoria. El operador *in* compara la identidad de dos objetos; la función **id()** devuelve un número entero representando la identidad (actualmente implementado como su dirección).

El ***tipo*** de un objeto también es inmutable. El tipo de un objeto determina las operaciones que admite el objeto (por ejemplo, "¿tiene una longitud?") Y también define los valores posibles para los objetos de ese tipo. La función ***type()*** devuelve el tipo de un objeto (que es un objeto en sí mismo). El ***valor de algunos objetos puede cambiar***. Se dice que los objetos cuyo valor puede cambiar son **mutables**; los objetos cuyo valor no se puede cambiar una vez que se crean se llaman **immutable**.

Los objetos son la clave para entender la *POO*. Si mira a nuestro alrededor encontrará un sin fin de objetos de la vida real: perro, escritorio, televisor, bicicleta, entre otros.

En Python puede definir una clase con la palabra reservada **class**, de la siguiente forma:

```
class Persona:
    pass #No realiza ninguna acción
```

En el ejemplo anterior, el nombre de la clase es Persona y dentro del bloque de código usa la sentencia **pass**. Aunque no es requerido por el intérprete, los nombres de las clases se escriben por convención capitalizadas. Las clases pueden (y siempre deberían) tener comentarios.

Atributos: Los atributos o propiedades de los objetos son las características que puede tener un objeto, como el color. Si el objeto es Persona, los atributos podrían ser: cedula, nombre, apellido, sexo, entre otros.

Los atributos describen el estado de un objeto. Pueden ser de cualquier tipo de dato.

Métodos: Los métodos describen el comportamiento de los objetos de una clase. Estos representan las operaciones que se pueden realizar con los objetos de la clase.

Se definen de la misma forma que las funciones normales pero deben declararse dentro de la clase y su primer argumento siempre referencia a la instancia que la llama, de esta forma se afirma que los métodos son *funciones*, adjuntadas a *objectos*.

- La única diferencia sintáctica entre la definición de un método y la definición de una función es que el primer parámetro del método por convención debe ser el nombre **self**.

Clases: Las clases definen las características del objeto.

Con todos los conceptos anteriores explicados, se puede decir que una clase es una plantilla genérica de un objeto. La clase proporciona variables iniciales de estado (donde se guardan los atributos) e implementaciones de comportamiento (métodos) necesarias para crear nuevos objetos, son los modelos sobre los cuáles serán construidos.

Instancias: Ya sabe que una clase es una estructura general del objeto. Por ejemplo, puede decir que la clase Persona necesita tener una cedula, un nombre, un apellido y un sexo, pero no va a decir cuál es cedula, nombre, apellido y sexo, es aquí donde entran las instancias.

Una instancia es una copia específica de la clase con todo su contenido.

Métodos __init__(): Es un método especial, el cual se ejecuta al momento de instanciar un objeto. El comportamiento de __init__() es muy similar a los **"constructores"** en otros lenguajes. Los argumentos que se utilizan en la definición de __init__() corresponden a los parámetros que se deben ingresar al instanciar un objeto.

Veamos un ejemplo sencillo de la definición de una clase en Python.

```
class Persona:
    #Método constructor
    def __init__(self,nombre,edad,genero):
        #Atributos de la clase Persona
        self.nombre = nombre
        self.edad = edad
        self.genero = genero
    #Método saludo() de la clase Persona
    def saludo(self):
        print("Hola a todos, mi nombre es "+self.nombre)

#Instancia de la clase Persona
Yo = Persona('Alexanyer',21,'Masculino')
#Invocación del método saludo() de la clase Persona
Yo.saludo()
```

Herencia: La herencia es una de las premisas y técnicas de la POO la cual permite a los programadores crear una clase general primero y luego más tarde crear clases más especializadas que re-utilicen código de la clase general. La herencia también le permite escribir un código más limpio y legible.

Herencia Simple: La herencia simple se apoya en el uso de clase padre para compartir sus atributos y comportamientos con otras clases derivadas.

Veamos un ejemplo del uso de la Herencia simple.

```
class Persona:
    #Método constructor
    def __init__(self,nombre,edad,genero):
        #Atributos de la clase Persona
        self.nombre = nombre
        self.edad = edad
        self.genero = genero
    #Método saludo() de la clase Persona
    def saludo(self):
        print("Hola a todos, mi nombre es "+self.nombre)

class Empleado(Persona):
    #Método constructor de la clase Empleado
    def __init__(self,nombre,edad,genero,puesto,sueldo):
        #Invocamos el método constructor de la clase padre (Persona)
        Persona.__init__(self,nombre,edad,genero)
        #Inicializamos los valores de los atributos de la clase Empleado
        self.puesto = puesto
        self.sueldo = sueldo
    #Método saludoEmpleado() de la clase Empleado
    #Observe que hacemos uso del atributo nombre de la clase Persona
    def saludoEmpleado(self):
        print("Hola a todos, soy un empleado y mi nombre es: "+self.nombre)

#Instancia de la clase Empleado
Yo = Empleado('Alexanyer',21,'Masculino','Gerente',5000.00)
#Invocación del método saludo() de la clase Persona
Yo.saludo()
#Invocación del método saludoEmpleado() de la clase Empleado
Yo.saludoEmpleado()
```

Herencia Múltiple: La herencia múltiple es la capacidad de una subclase de heredar de múltiples súper clases.

Esto conlleva un problema, y es que, si varias súper clases tienen los mismos atributos o métodos, la subclase sólo podrá heredar de una de ellas.

En estos casos Python dará prioridad a las clases más a la izquierda en el momento de la declaración de la subclase.

Veamos un ejemplo del uso de la Herencia Múltiple.

```python
class Persona:
    #Método constructor
    def __init__(self,nombre,edad,genero):
        #Atributos de la clase Persona
        self.nombre = nombre
        self.edad = edad
        self.genero = genero
    #Método saludo() de la clase Persona
    def saludo(self):
        print("Hola a todos, mi nombre es "+self.nombre)

class Empleado():
    #Método constructor de la clase Empleado
    def __init__(self,puesto,sueldo):
        #Inicializamos los valores de los atributos de la clase Empleado
        self.puesto = puesto
        self.sueldo = sueldo
    #Método saludoEmpleado() de la clase Empleado
    #Observe que hacemos uso del atributo nombre de la clase Persona
    def saludoEmpleado(self):
        print("Hola a todos, mi puesto es: "+self.puesto)

class Cajero(Persona,Empleado):
    def __init__(self,nombre,edad,genero,puesto,sueldo,nro_caja):
        #Invocación del método constructor de la clase Persona
        Persona.__init__(self,nombre,edad,genero)
        #Invocación del método constructor de la clase Empleado
        Empleado.__init__(self,puesto,sueldo)
        #Inicializamos los atributos de la clase Cajero
        self.nro_caja = nro_caja
    #Método saludoCajero() de la clase Cajero
    #Observe que hacemos uso del atributo nombre de la clase Persona
    def saludoCajero(self):
        print("Hola a todos, soy un cajero y mi nombre es: "+self.nombre)

#Instancia de la clase Cajero
Yo = Cajero('Alexanyer',21,'Masculino','Cajero',2500.00,5)
#Invocación del método saludo() de la clase Persona
Yo.saludo()
#Invocación del método saludoEmpleado() de la clase Empleado
Yo.saludoEmpleado()
#Invocación del método saludoCajero() de la clase Cajero
Yo.saludoCajero()
```