INSTITUTO SUPERIOR POLITÉCNICO CÓRDOBA

Introducción a la Programación Orientada a Objetos (cont)

Contenido

Clases e instancias en Python

Clases

Una clase en Python no es más que una secuencia de símbolos (o caracteres). Esta secuencia de símbolos forma lo que se denomina el código fuente de la clase. Hay dos aspectos que determinan si una secuencia de símbolos es correcta o no: la **sintaxis** y la **semántica**.

Por un lado, la sintaxis de Python nos permite determinar de qué manera los símbolos del vocabulario pueden combinarse para escribir código fuente correcto mientras que la semántica, guarda una estrecha relación con lo que es código hace permitiendo así determinar el significado de la secuencia de símbolos para que se lleve a cabo la acción por la computadora.

Así, por ejemplo, en el lenguaje natural son las reglas de la sintaxis las que nos permiten determinar que la frase *"***programa hombre el autómata***"* no es correcta y, las reglas semánticas nos posibilitan detectar que la siguiente frase *"***el autómata programa al hombre***"* nos es correcta en términos de semántica (aunque en términos de sintaxis si lo es)

Sintaxis para la definición de clases en Python:

```
class <NombreDeLaClase>:
    <nombreDeAtributoDeClase> = <valor>
    def __init__(self,<parametro1>, <parametro2>, ...):
        self.<atributo1> = <parametro1>
        self.<atributo2> = <parametro2>
        .
        .
        .
        # Tantos atributos como se necesite.

   def <nombre_de_metodo>(self):
       # Código del método
       # Tantos métodos como se necesite
```

Como podemos observar, dentro de las clases podemos encontrar:

- ***Atributos:*** Son *variables* que se declaran dentro de la clase, y sirven para indicar la *forma o características* de cada objeto representado por esa clase. Los atributos, de alguna manera, muestran lo que cada objeto *es*, o también, lo que cada objeto *tiene*.

En el ejemplo demostrado arriba, **< nombreDeAtributoDeClase >** es el nombre del atributo de la clase, y **<valor>** es el valor inicial del atributo de la clase. Los atributos de clase se definen fuera de cualquier método de la clase, y se pueden acceder desde cualquier instancia de la clase utilizando la sintaxis **objeto.nombre_de_atributo_de_la_clase**

Por otro lado, **self.<atributo>** es la sintaxis que se utiliza para definir un atributo de instancia en Python. Los atributos de instancia se definen dentro del método **__init__** de la clase y se inicializan con los valores de los parámetros que se pasan al crear una instancia de la clase. Los atributos de instancia se pueden acceder desde cualquier método de la instancia utilizando la sintaxis **self.atributo** (https://www.freecodecamp.org/).

- **Métodos:** Son funciones, procedimientos o rutinas declaradas dentro de la clase, usados para describir el *comportamiento o las acciones* de los objetos descritos por esa clase. Los métodos, de alguna manera, muestran lo que cada objeto *hace*.

  La sintaxis es:

```
class <NombreDeLaClase>:
  def <nombreDelMétodo>(self, <parametro1>, <parametro2> ...):
      # Código del método
```

  En este ejemplo, **<nombreDelMétodo>** es el nombre del método, y **<parametro1>, <parametro2>**, etc. son los argumentos que recibe el método. El primer argumento de un método siempre es **self**, que se refiere como mencionamos previamente a la instancia de la clase que está siendo manipulada.

  Es común (pero no obligatorio) que los atributos de la clase se declaren antes que los métodos. El conjunto de atributos y métodos de una clase se conoce como el conjunto de ***miembros*** de la clase.

  Finalmente es importante mencionar que las clases no se construyen para que trabajen de manera aislada, la idea es que ellas se puedan relacionar entre sí, de manera que puedan compartir atributos y métodos sin necesidad de volver a escribirlos.

- ***Constructores:*** es un método especial que permite instanciar un objeto. Su nombre está definido por la palabra **__init__**. Puede recibir 1 a n parámetros.

Sintaxis:

```
class <NombreDeLaClase>:
    def __init__(self,<parametro1>, <parametro2>, ...):
# Código del constructor
```

El constructor suele usarse para la inicialización de los atributos del objeto a crear, sobre todo cuando el valor de éstos no es constante o incluye acciones más allá de una asignación de valor.

- ***Propiedades (getters y los setters).*** Las propiedades getter y setter en Python son métodos especiales que se utilizan para acceder y modificar los atributos privados de una clase (convencionalmente se utiliza el _ antes del nombre del atributo para indicar que es privado).

Ejemplo:

```
  class Persona:
    def __init__(self, nombre, edad):
        self.__nombre = nombre
        self.__edad = edad

    def get_nombre(self):
        return self.__nombre

    def set_nombre(self, nombre):
        self.__nombre = nombre

    def get_edad(self):
        return self.__edad

    def set_edad(self, edad):
        self.__edad = edad

    def imprimir_datos(self):
        print(f"Nombre: {self.get_nombre()}")
        print(f"Edad: {self.get_edad()}")

p = Persona("Juan", 30)
p.imprimir_datos() # Imprime "Nombre: Juan" y "Edad: 30"
p.set_nombre("Pedro")
p.set_edad(35)
p.imprimir_datos() # Imprime "Nombre: Pedro" y "Edad: 35"
```

Como puedes observar en este ejemplo, la clase Persona tiene dos atributos privados, __nombre y __edad. Para acceder y modificar estos atributos, se definen los métodos get_nombre, set_nombre, get_edad y set_edad. Los métodos get_nombre y get_edad son los getters, que se utilizan para acceder a

los atributos privados, mientras que los métodos set_nombre y set_edad son los setters, que se utilizan para modificarlos.

Por otro lado, es posible definir las propiedades getter y setter de una forma más sencilla utilizando los decoradores @property y @x.setter como se muestra en el ejemplo:

```
class MyClass:
    def __init__(self):
        self._x = None
    @property
    def x(self):
        return self._x
    @x.setter
    def x(self, value):
        self._x = value
```

En este ejemplo, **x** es una propiedad que encapsula el atributo **_x** (privado por convención). La propiedad x se define utilizando el decorador **@property**, que indica que el método x es un getter. El setter de la propiedad se define utilizando el decorador **@x.setter**, que indica que el método x es un setter.

- ***Modificadores de acceso:*** En Python no existen modificadores de acceso como en otros lenguajes de programación orientados a objetos. Todos los miembros de una clase en Python son públicos por defecto. Sin embargo, se utiliza una convención de nomenclatura para indicar que un atributo o método de una clase no debería ser accedido desde fuera de la clase (ej. __atributoPrivado).

Instancias

En Python, una instancia de clase es un objeto que se crea a partir de una clase. Para crear el objeto, utilizamos la sintaxis **nombre_de_clase().**

Sintaxis para instanciar objetos:

```
<nombre_objeto>= <Nombre_de_Clase>(<parámetros>)
```

Ejemplo:

```
objeto = MiClase(1, 2)
```

Sintaxis para inicializar un objeto:

Hay 3 maneras de inicializar un objeto. Es decir, proporcionar datos a un objeto.

1. Por referencia a variables

   Ejemplo:

```
persona = new Persona();
persona.apellido="Rosas";
persona.nombre ="Maria";
```

2. Por medio del constructor de la clase:

   Ejemplo:

```
persona= Persona("Maria","Rosas");
```

3. Por medio de la propiedad setter:

   Ejemplo:

```
persona= Persona();
persona.Apellido="Rosas";
persona.Nombre ="Maria";
```

Recomendaciones

Aunque cada programador puede definir su propio estilo de programación, una buena práctica es seguir el estilo utilizado por los diseñadores del lenguaje pues, de seguir esta práctica será mucho más fácil analizar el código fuente de terceros y, a su vez, que otros programadores analicen y comprendan nuestro código fuente.

- Evitar en lo posible líneas de longitud superior a 80 caracteres.
- Identar los bloques de código.
- Utilizar identificadores nemotécnicos, es decir, utilizar nombres simbólicos adecuados para los identificadores lo suficientemente autoexplicativos por sí mismos para dar una orientación de su uso o funcionalidad de manera tal que podamos hacer más claros y legibles nuestros códigos.
- Los identificadores de clases, módulos, interfaces y enumeraciones deberán usar **PascalCase**.
- Los identificadores de objetos, métodos, instancias, constantes, propiedades y métodos de los objetos deberán usar **camelCase**.
- Utilizar comentarios con moderación (recuerda que si sientes la necesidad de comentar el código, posiblemente tu código no sea limpio), éstos deberán seguirán un formato general de fácil portabilidad y que no incluya líneas completas de caracteres repetidos. Los que se coloquen dentro de bloques de

código deben aparecer en una línea independiente identada de igual forma que el bloque de código que describen.

Referencias

Apuntes Programa Clip - Felipe Steffolani

https://www.freecodecamp.org/

Laura Álvarez, Helmer Avendaño, Yeison García, Sebastián Morales,
Edwin Bohórquez, Santiago Hernandez, Sebastián Moreno, Cristian Orjuela.
Programación Orientada a Objetos.
https://ferestrepoca.github.io/paradigmas-de-programacion/poo/poo_teoria/concepts.html

https://www.typescriptlang.org/

https://barcelonageeks.com/composicion-de-funciones-en-python/

111Mil. Módulo Programación Orientada a Objetos.
https://github.com/111milprogramadores/apuntes/blob/master/Programacion%20Orientada%20a%20Objetos/Apunte%20Teorico%20de%20Programacion%20OO.pdf