Módulo 3: Programación Orientada a Objetos - propiedades

Variables de instancia

En general, una clase puede equiparse con dos tipos diferentes de datos para formar las propiedades de una clase. Ya viste uno de ellos cuando estábamos estudiando pilas.

Este tipo de propiedad existe solo cuando se crea explícitamente y se agrega a un objeto. Como ya sabes, esto se puede hacer durante la inicialización del objeto, realizada por el constructor.

Además, se puede hacer en cualquier momento de la vida del objeto. Es importante mencionar también que cualquier propiedad existente se puede eliminar en cualquier momento.

Tal enfoque tiene algunas consecuencias importantes:

- Diferentes objetos de la misma clase **pueden poseer diferentes conjuntos de propiedades**.
- Debe haber una manera de **verificar con seguridad si un objeto específico posee la propiedad** que deseas utilizar (a menos que quieras generar una excepción, siempre vale la pena considerarlo).
- Cada objeto **lleva su propio conjunto de propiedades**, no interfieren entre sí de ninguna manera.

Tales variables (propiedades) se llaman **variables de instancia**.

La palabra instancia sugiere que están estrechamente conectadas a los objetos (que son instancias de clase), no a las clases mismas. Echemos un vistazo más de cerca.

Aquí hay un ejemplo:

```
class ExampleClass:
    def __init__(self, val = 1):
        self.first = val

    def set_second(self, val):
        self.second = val


example_object_1 = ExampleClass()
example_object_2 = ExampleClass(2)

example_object_2.set_second(3)

example_object_3 = ExampleClass(4)
example_object_3.third = 5

print(example_object_1.__dict__)
print(example_object_2.__dict__)
print(example_object_3.__dict__)
```

Se necesita una explicación adicional antes de entrar en más detalles. Echa un vistazo a las últimas tres líneas del código.

Los objetos de Python, cuando se crean, **están dotados de un pequeño conjunto de propiedades y métodos predefinidos**. Cada objeto los tiene, los quieras o no. Uno de ellos es una variable llamada __dict__

(es un diccionario).

La variable contiene los nombres y valores de todas las propiedades (variables) que el objeto contiene actualmente. Vamos a usarla para presentar de forma segura el contenido de un objeto.

Vamos a sumergirnos en el código ahora:

- La clase llamada ExampleClass tiene un constructor, el cual **crea incondicionalmente una variable de instancia** llamada first, y le asigna el valor pasado a través del primer argumento (desde la perspectiva del usuario de la clase) o el segundo argumento (desde la perspectiva del constructor); ten en cuenta el valor predeterminado del parámetro: cualquier cosa que puedas hacer con un parámetro de función regular también se puede aplicar a los métodos.
- La clase también tiene un **método que crea otra variable de instancia**, llamada second.
- Hemos creado tres objetos de la clase ExampleClass, pero todas estas instancias difieren:
  - example_object_1 solo tiene una propiedad llamada first.
  - example_object_2 tiene dos propiedades: first y second.
  - example_object_3 ha sido enriquecido sobre la marcha con una propiedad llamada third uera del código de la clase: esto es posible y totalmente permisible.

La salida del programa muestra claramente que nuestras suposiciones son correctas: aquí están:

```
{'first': 1}
{'second': 3, 'first': 2}
{'third': 5, 'first': 4}
```

Hay una conclusión adicional que debería mencionarse aquí: **el modificar una variable de instancia de cualquier objeto no tiene impacto en todos los objetos restantes**. Las variables de instancia están perfectamente aisladas unas de otras.

```
class ExampleClass:
    def __init__(self, val = 1):
        self.__first = val

    def set_second(self, val = 2):
        self.__second = val


example_object_1 = ExampleClass()
example_object_2 = ExampleClass(2)

example_object_2.set_second(3)

example_object_3 = ExampleClass(4)
example_object_3.__third = 5


print(example_object_1.__dict__)
print(example_object_2.__dict__)
print(example_object_3.__dict__)
```

Es casi lo mismo que el anterior. La única diferencia está en los nombres de las propiedades. Hemos **antepuesto dos guiones bajos** (__).

Como sabes, tal adición hace que la variable de instancia sea privada, se vuelve inaccesible desde el mundo exterior.

El comportamiento real de estos nombres es un poco más complicado, así que ejecutemos el programa. Esta es la salida:

```
{'_ExampleClass__first': 1}
{'_ExampleClass__first': 2, '_ExampleClass__second': 3}
{'_ExampleClass__first': 4, '__third': 5}
```

¿Puedes ver estos nombres extraños llenos de guiones bajos? ¿De dónde provienen?

Cuando Python ve que deseas agregar una variable de instancia a un objeto y lo vas a hacer dentro de cualquiera de los métodos del objeto, **maneja la operación** de la siguiente manera:

- Coloca un nombre de clase antes de tu nombre.
- Coloca un guión bajo adicional al principio.

Es por ello que __first se convierte en _ExampleClass__first.

El nombre ahora es completamente accesible desde fuera de la clase. Puedes ejecutar un código como este:

print(example_object_1._ExampleClass__first)

Obtendrás un resultado válido sin errores ni excepciones.

Como puedes ver, hacer que una propiedad sea privada es limitado.

No funcionará si agregas una variable de instancia fuera del código de la clase. En este caso, se comportará como cualquier otra propiedad ordinaria.

Variables de clase

Una variable de clase es **una propiedad que existe en una sola copia y se almacena fuera de cualquier objeto**.

Nota: no existe una variable de instancia si no hay ningún objeto de la clase; solo existe una variable de clase en una copia, incluso si no hay objetos en la clase.

Las variables de clase se crean de manera diferente. El ejemplo te dirá más:

```
class ExampleClass:
    counter = 0
    def __init__(self, val = 1):
        self.__first = val
        ExampleClass.counter += 1


example_object_1 = ExampleClass()
example_object_2 = ExampleClass(2)
example_object_3 = ExampleClass(4)

print(example_object_1.__dict__, example_object_1.counter)
print(example_object_2.__dict__, example_object_2.counter)
print(example_object_3.__dict__, example_object_3.counter)
```

Observa:

- Hay una asignación en la primera linea de la definición de clase: establece la variable denominada counter a 0; inicializando la variable dentro de la clase pero fuera de cualquiera de sus métodos hace que la variable sea una variable de clase.
- El acceder a dicha variable tiene el mismo aspecto que acceder a cualquier atributo de instancia; está en el cuerpo del constructor; como puedes ver, el constructor incrementa la variable en uno; en efecto, la variable cuenta todos los objetos creados.

Ejecutar el código provocará el siguiente resultado:

```
{'_ExampleClass__first': 1} 3
{'_ExampleClass__first': 2} 3
{'_ExampleClass__first': 4} 3
```

Dos conclusiones importantes se pueden sacar del ejemplo:

- Las variables de clase **no se muestran en el diccionario de un objeto** __dict__ (esto es natural ya que las variables de clase no son partes de un objeto), pero siempre puedes intentar buscar en la variable del mismo nombre, pero a nivel de clase, te mostraremos esto muy pronto.
- Una variable de clase **siempre presenta el mismo valor** en todas las instancias de clase (objetos).

El cambiar el nombre de una variable de clase tiene los mismos efectos que aquellos con los que ya está familiarizado.

Mira el ejemplo en el editor. ¿Puedes adivinar su salida?

```
class ExampleClass:
    __counter = 0
    def __init__(self, val = 1):
        self.__first = val
        ExampleClass.__counter += 1


example_object_1 = ExampleClass()
example_object_2 = ExampleClass(2)
example_object_3 = ExampleClass(4)

print(example_object_1.__dict__, example_object_1._ExampleClass__counter)
print(example_object_2.__dict__, example_object_2._ExampleClass__counter)
print(example_object_3.__dict__, example_object_3._ExampleClass__counter)
```

Hemos dicho antes que las variables de clase existen incluso cuando no se creó ninguna instancia de clase (objeto).

Ahora aprovecharemos la oportunidad para mostrarte **la diferencia entre estas dos variables** __dict__, la de la clase y la del objeto.

```
class ExampleClass:
    varia = 1
    def __init__(self, val):
        ExampleClass.varia = val


print(ExampleClass.__dict__)
example_object = ExampleClass(2)
```

```
print(ExampleClass.__dict__)
print(example_object.__dict__)
```

Echemos un vistazo más de cerca:

1. Definimos una clase llamada ExampleClass.
2. La clase define una variable de clase llamada varia.
3. El constructor de la clase establece la variable con el valor del parámetro.
4. Nombrar la variable es el aspecto más importante del ejemplo porque:
    - El cambiar la asignación a self.varia = val crearía una variable de instancia con el mismo nombre que la de la clase.
    - El cambiar la asignación a varia = val operaría en la variable local de un método; (te recomendamos probar los dos casos anteriores; esto te facilitará recordar la diferencia).
5. La primera línea del código fuera de la clase imprime el valor del atributo ExampleClass.varia . Nota: utilizamos el valor antes de instanciar el primer objeto de la clase.

Ejecuta el código en el editor y verifica su salida.

```
{'__module__': '__main__', 'varia': 1, '__init__': <function ExampleClass.__init__
at 0x7fc83922b0e0>, '__dict__': <attribute '__dict__' of 'ExampleClass' objects>,
'__weakref__': <attribute '__weakref__' of 'ExampleClass' objects>, '__doc__':
None}
{'__module__': '__main__', 'varia': 2, '__init__': <function ExampleClass.__init__
at 0x7fc83922b0e0>, '__dict__': <attribute '__dict__' of 'ExampleClass' objects>,
'__weakref__': <attribute '__weakref__' of 'ExampleClass' objects>, '__doc__':
None}
```

Como puedes ver __dict__ contiene muchos más datos que la contraparte de su objeto. La mayoría de ellos son inútiles ahora, el que queremos que verifiques cuidadosamente muestra el valor actual de varia.

Nota que el __dict__ del objeto está vacío, el objeto no tiene variables de instancia.

Comprobando la existencia de un atributo

La actitud de Python hacia la instanciación de objetos plantea una cuestión importante: en contraste con otros lenguajes de programación, **es posible que no esperes que todos los objetos de la misma clase tengan los mismos conjuntos de propiedades**.

```
class ExampleClass:
    def __init__(self, val):
        if val % 2 != 0:
            self.a = 1
        else:
            self.b = 1


example_object = ExampleClass(1)

print(example_object.a)
print(example_object.b)
```

El objeto creado por el constructor solo puede tener uno de los dos atributos posibles: a o b.

La ejecución del código producirá el siguiente resultado:

```
1
Traceback (most recent call last):
  File ".main.py", line 11, in
    print(example_object.b)
AttributeError: 'ExampleClass' object has no attribute 'b'
```

Como puedes ver, acceder a un atributo de objeto (clase) no existente genera una excepción AttributeError.

La instrucción **try-except** te brinda la oportunidad de evitar problemas con propiedades inexistentes.

```
class ExampleClass:
    def __init__(self, val):
        if val % 2 != 0:
            self.a = 1
        else:
            self.b = 1


example_object = ExampleClass(1)
print(example_object.a)

try:
    print(example_object.b)
except AttributeError:
    pass
```

Como puedes ver, esta acción no es muy sofisticada. Esencialmente, acabamos de barrer el tema debajo de la alfombra.

Afortunadamente, hay una forma más de hacer frente al problema.

Python proporciona una **función que puede verificar con seguridad si algún objeto / clase contiene una propiedad específica**. La función se llama *hasattr*, y espera que le pasen dos argumentos:

- La clase o el objeto que se verifica.
- El nombre de la propiedad cuya existencia se debe informar (Nota: debe ser una cadena que contenga el nombre del atributo).

La función retorna True o False.

Así es como puedes utilizarla:

```
class ExampleClass:
    def __init__(self, val):
        if val % 2 != 0:
            self.a = 1
        else:
            self.b = 1


example_object = ExampleClass(1)
print(example_object.a)

if hasattr(example_object, 'b'):
```

```
    print(example_object.b)
```

No olvides que la función `hasattr()` también puede operar en clases. Puedes usarla **para averiguar si una variable de clase está disponible**, como en el ejemplo en el editor.

La función devuelve True si la clase especificada contiene un atributo dado, y False de lo contrario.

```
class ExampleClass:
    attr = 1


print(hasattr(ExampleClass, 'attr'))
print(hasattr(ExampleClass, 'prop'))
```

Un ejemplo más: analiza el código a continuación e intenta predecir su salida:

```
class ExampleClass:
    a = 1
    def __init__(self):
        self.b = 2


example_object = ExampleClass()

print(hasattr(example_object, 'b'))
print(hasattr(example_object, 'a'))
print(hasattr(ExampleClass, 'b'))
print(hasattr(ExampleClass, 'a'))
```

Bien, hemos llegado al final de esta sección. En la siguiente sección vamos a hablar sobre los métodos, ya que los métodos dirigen los objetos y los activan.

Puntos Clave

1. Una variable de instancia es una propiedad cuya existencia depende de la creación de un objeto. Cada objeto puede tener un conjunto diferente de variables de instancia.

Además, se pueden agregar y quitar libremente de los objetos durante su vida útil. Todas las variables de instancia de objeto se almacenan dentro de un diccionario dedicado llamado __dict__, contenido en cada objeto por separado.

2. Una variable de instancia puede ser privada cuando su nombre comienza con __, pero no olvides que dicha propiedad aún es accesible desde fuera de la clase usando un **nombre modificado** construido como < codel>_ClassName__PrivatePropertyName.

3. Una **variable de clase** es una propiedad que existe exactamente en una copia y no necesita ningún objeto creado para ser accesible. Estas variables no se muestran como contenido de __dict__.

Todas las variables de clase de una clase se almacenan dentro de un diccionario dedicado llamado __dict__, contenido en cada clase por separado.

4. Una función llamada `hasattr()` se puede utilizar para determinar si algún objeto o clase contiene cierta propiedad especificada.

Por ejemplo:

```
class Sample:
    gamma = 0 # Class variable.
    def __init__(self):
        self.alpha = 1 # Variable de instancia.
        self.__delta = 3 # Variable de instancia privada.


obj = Sample()
obj.beta = 2  # Otra variable de instancia (que existe solo dentro de la instancia "obj").
print(obj.__dict__)
```

El código da como salida:

```
{'alpha': 1, '_Sample__delta': 3, 'beta': 2}
```

https://edube.org/learn/python-essentials-2-esp/poo-m-eacute-todos-10