Conceptos avanzados de programación orientada a objetos II

Propiedades: getters, setters, deleter

Para implementar la encapsulación y no permitir el acceso directo a los atributos, podemos poner los atributos ocultos y declarar métodos especificos para acceder y modificar los atributos (mutadores). Estos métodos se denominan getters y setters.

```
class circulo():
    def __init__(self,radio):
        self.set_radio(radio)
    def set_radio(self,radio):
        if radio>=0:
            self._radio = radio
        else:
            raise ValueError("Radio positivo")
            self._radio=0
    def get_radio(self):
        print("Estoy dando el radio")
        return self._radio

>>> c1=circulo(3)
>>> c1.get_radio()
Estoy dando el radio
3
>>> c1.set_radio(-1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/jose/github/curso_python3/curso/u51/circulo.py", line 8, in set_radio
    raise ValueError("Radio positivo")
ValueError: Radio positivo
```

En Python, las propiedades nos permiten implementar la funcionalidad exponiendo estos métodos como atributos.

```
class circulo():
    def __init__(self,radio):
        self.radio=radio

    @property
    def radio(self):
        print("Estoy dando el radio")
        return self._radio

    @radio.setter
    def radio(self,radio):
        if radio>=0:
            self._radio = radio
        else:
            raise ValueError("Radio positivo")
            self._radio=0


>>> c1=circulo(3)
>>> c1.radio
Estoy dando el radio
3
>>> c1.radio=4
>>> c1.radio=-1
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
  File "/home/jose/github/curso_python3/curso/u52/circulo2.py", line 15, in radio
    raise ValueError("Radio positivo")
ValueError: Radio positivo
```

Hay un tercera property que podemos crear: el deleter

```
...
@radio.deleter
def radio(self):
    del self._radio


>>> c1=circulo(3)
>>> c1.radio
Estoy dando el radio
3
>>> del c1.radio
>>> c1.radio
Estoy dando el radio
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/jose/github/curso_python3/curso/u52/circulo2.py", line 8, in radio
    return self._radio
AttributeError: 'circulo' object has no attribute '_radio'
>>> c1.radio=3
```

Representación de objetos __str__ y __repr__

La documentación de Python hace referencia a que el método `__str()__` ha de devolver la representación "informal" del objeto, mientras que `__repr()__` la "formal".

- La función `__str()__` debe devolver la cadena de texto que se muestra por pantalla si llamamos a la función `str()`. Esto es lo que hace Python cuando usamos `print`. Suele devolver el nombre de la clase.
- De `__repr()__`, por el otro lado, se espera que nos devuelva una cadena de texto con una representación única del objeto. Idealmente, la cadena devuelta por `__repr()__` debería ser aquella que, pasada a `eval()`, devuelve el mismo objeto.

Continuamos con la clase `circulo`:

```
...
def __str__(self):
    clase = type(self).__name__
    msg = "{0} de radio {1}"
    return msg.format(clase, self.radio)

def __repr__(self):
    clase = type(self).__name__
    msg = "{0}({1})"
    return msg.format(clase, self.radio)
```

Suponemos que estamos utilizando la clase `circulo` sin la instrucción `print` en el getter.

```
>>> c1=circulo(3)
>>> print(c1)
circulo de radio 3
>>> repr(c1)
'circulo(3)'
>>> type(eval(repr(c1)))
<class 'circulo2.circulo'>
```

Comparación de objetos __eq__

Tampoco podemos comparar dos `circulos` sin definir `__eq()__`, ya que sin este método Python comparará posiciones en memoria.

Continuamos con la clase `circulo`:

```
...
def __eq__(self,otro):
    return self.radio==otro.radio

>>> c1=circulo(5)
>>> c2=circulo(3)
>>> c1 == c2
False
```

Si queremos utilizar `<`, `<=`, `>` y `>=` tendremos que rescribir los métodos: `__lt()__`, `__le()__`, `__gt()__` y `__ge()__`

Operar con objetos __add__ y __sub__

Si queremos operar con los operadores `+` y `-`:

```
def __add__(self,otro):
    self.radio+=otro.radio

def __sub__(self,otro):
    if self.radio-otro.radio>=0:
        self.radio-=otro.radio
    else:
        raise ValueError("No se pueden restar")

>>> c1=circulo(5)
>>> c2=circulo(3)
>>> c1 + c2
>>> c1.radio
8


>>> c1=circulo(5)
>>> c2=circulo(3)
>>> c1 - c2
>>> c1.radio
2
>>> c1 - c2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/jose/github/curso_python3/curso/u52/circulo2.py", line 42, in __sub__
    raise ValueError("No se pueden restar")
ValueError: No se pueden restar
```

Más métodos especiales

Existen muchos más métodos especiales que podemos sobreescribir en nuestras clases para añadir funcionalidad a las mismas. Puedes ver la documentación oficial para aprender más sobre ellas.