Clase9_POO_cont

May 8, 2023

1 Seminario de Lenguajes - Python

1.1 Cursada 2023

1.1.1 Aspectos básicos de POO (Cont.)

2 Repasemos algunos conceptos vistos previamente

#

Un objeto es una colección de datos con un comportamiento asociado en una única entidad.

3 POO: conceptos básicos

- En POO un programa puede verse como un **conjunto de objetos** que interactúan entre ellos **enviándose mensajes**.
- Estos mensajes están asociados al **comportamiento** del objeto (conjunto de **métodos**).

4 El mundo de los objetos

- No todos los objetos son iguales, ni tienen el mismo comportamiento.
- Así **agrupamos** a los objetos de acuerdo a **características comunes**.

5 Objetos y clases

Una clase describe los atributos de objetos (**variables de instancia**) y las acciones o **métodos** que pueden hacer o ejecutar dichos objetos.

6 La clase Banda

[ ]:
```
class Banda():
    """ Representa a una banda de música"""

    generos = set()

    def __init__(self, nombre, genero="rock"):
        self.nombre = nombre
        self.genero = genero
```

```
        self._integrantes = []
        Banda.generos.add(genero)

    def agregar_integrante(self, nuevo_integrante):
        self._integrantes.append(nuevo_integrante)
```

¿self?¿Cuáles son las variables de instancias?¿Y los métodos?¿Qué es **generos**?

7 Creamos instancias de Banda

[ ]:
```
soda = Banda("Soda Stereo")
soda.agregar_integrante("Gustavo Cerati")
soda.agregar_integrante("Zeta Bosio")
soda.agregar_integrante("Charly Alberti")

nompa = Banda("Nonpalidece", genero="reggae")
nompa.agregar_integrante("Néstor Ramljak")
nompa.agregar_integrante("Agustín Azubel")
```

8 Mostramos el contenido de Banda.generos

[ ]:
```
for genero in Banda.generos:
        print(genero)
```

8.1 Objetos y clases

- La **clase** define las propiedades y los métodos de los objetos.
- Los **objetos** son instancias de una clase.
- Cuando se crea un objeto, se ejecuta automáticamente el método **___init()___** que permite inicializar el objeto.
- La definición de la clase especifica qué partes son públicas y cuáles vamos a considerar no públicas.

¿Cómo se especifica privado o público en Python?

9 Mensajes y métodos

TODO el procesamiento en este modelo es activado por mensajes entre objetos.

- El **mensaje** es el modo de comunicación entre los objetos. Cuando se invoca una función de un objeto, lo que se está haciendo es **enviando un mensaje** a dicho objeto.
- El **método** es la función que está asociada a un objeto determinado y cuya ejecución sólo puede desencadenarse a través del envío de un mensaje recibido.

- La **interfaz pública** del objeto está formada por las variables de instancias y métodos que otros objetos pueden usar para interactuar con él.

10 Hablemos de @property

Clase 8_1 sobre propiedades

11 Métodos de clase

- ¿A qué creen que hacen referencia?

  Corresponden a los mensajes que se envían a la **clase**, no a las instancias de la misma.

- Se utiliza el decorador **@classmethod**.
- Se usa **cls** en vez de **self**. ¿A qué hace referencia este argumento?

[ ]:
```
class Banda():
    generos = set()

    @classmethod
    def limpio_generos(cls, confirmo=False):
        if confirmo:
            cls.generos =set()

    def __init__(self, nombre, genero="rock"):
        self._nombre = nombre
        self._genero = genero
        self._integrantes = []
        Banda.generos.add(genero)
```

[ ]:
```
soda = Banda("Soda Stereo")
nompa = Banda("Nonpalidece", genero="reggae")
```

[ ]:
```
Banda.generos
```

[ ]:
```
Banda.limpio_generos()
```

12 Ahora, pensemos en los músicos de la banda

Podemos pensar en:

Donde: - Un guitarrista **"es un"** músico. - Un vocalista también **es un** músico.

13 Hablemos de herencia

- Es uno de los conceptos más importantes de la POO.

- La herencia permite que una clase pueda *heredar* los atributos y métodos de otra clase, que se **agregan** a los propios.
- Este concepto permite sumar, es decir **extender** una clase.
- La clase que hereda se denomina **clase derivada** y la clase de la cual se deriva se denomina **clase base**.
- Así, **Musico es la clase base** y **Guitarrista y Vocalista son clases derivadas** de Musico.

14 Ahora en Python

[ ]:
```python
class Musico:
    def __init__(self, nombre, puesto=None, banda=None):
        self.nombre = nombre
        self.tiene_banda = banda!=None
        self._banda = banda
        self.puesto = puesto

    def info(self):
        if self.tiene_banda:
            print (f"{self.nombre}  integra la banda {self.banda}")
        else:
            print(f"{self.nombre} es solista ")

    @property
    def banda(self):
        if self.tiene_banda:
            return self._banda
        else:
            return "No tiene banda"

    @banda.setter
    def banda(self, banda):
        self._banda = banda
        self.tiene_banda = banda!=None
```

[ ]:
```python
class Guitarrista(Musico):

    def __init__(self, nombre, banda=None):
        Musico.__init__(self, nombre, "guitarrista", banda)
        self.instrumento = "guitarra acústica"

    def info(self):
        print (f"{self.nombre} toca {self.instrumento}")
```

- ¿Cuál es la clase base? ¿Y la clase derivada? ¿Cuáles son las variables de instancia de un objeto Guitarrista?

- ¿Por qué invoco a **Músico.__init__()**? ¿Qué pasa si no hago esto?

[ ]:
```
class Vocalista(Musico):

    def __init__(self, nombre, banda=None):
        Musico.__init__(self, nombre, "vocalista")
        self.tipo_voz = "Barítono"
```

[ ]:
```
bruce = Vocalista('Bruce Springsteen')
brian = Guitarrista("Brian May", "Queen")
```

[ ]:
```
bruce.info()
brian.info()
```

[ ]:
```
bruce.tiene_banda
```

[ ]:
```
bruce.banda = "E Street Band"
bruce.info()
```

15 También podemos chequear ...

[ ]:
```
f"{bruce.nombre} es vocalista" if isinstance(bruce, Vocalista) else f"{bruce.
 ↪nombre} NO es vocalista"
```

[ ]:
```
"Guitarrista ES subclase de Musico" if issubclass(Guitarrista, Musico) else␣
 ↪"Guitarrista NO es subclase de Musico"
```

16 Bruce Springsteen es un vocalista, pero también es un guitarrista...

Podríamos pensar en algo así:

Aunque en Python podemos hacer algo mejor ...

17 Python tiene herencia múltiple

- Un guitarrista y vocalista **"es un"** guitarrista, pero también **es un** vocalista..

18 En Python ...

[ ]:
```
class Guitarrista(Musico):
    def __init__(self, nombre, banda=None):
        Musico.__init__(self, nombre, "guitarrista", banda)
        self.instrumento = "guitarra acústica"
```

```
    def info(self):
        print (f"{self.nombre} toca {self.instrumento}")
```

[ ]:
```
class Vocalista(Musico):
    def __init__(self, nombre, banda=None):
        Musico.__init__(self, nombre, "vocalista")
        self.tipo_voz = "Barítono"

    def info(self):
        if self.tiene_banda:
            print (f"{self.nombre} CANTA en la banda {self.banda}")
        else:
            print(f"{self.nombre} es solista ")
```

[ ]:
```
class VocalistaYGuitarrista(Guitarrista, Vocalista):

    def __init__(self, nombre, banda=None):
        Vocalista.__init__(self, nombre, banda)
        Guitarrista.__init__(self, nombre, banda)
```

[ ]:
```
bruce = VocalistaYGuitarrista('Bruce Springsteen')
bruce.info()
```

[ ]:
```
mollo = VocalistaYGuitarrista("Ricardo Mollo", "Divididos")
mollo.info()
```

19 A tener en cuenta ...

- MRO “Method Resolution Order”
- Por lo tanto, es MUY importante el orden en que se especifican las clases bases.
- Más información en documentación oficial

[ ]:
```
VocalistaYGuitarrista.__mro__
```

[ ]:
```
class VocalistaYGuitarrista(Vocalista, Guitarrista):

    def __init__(self, nombre, banda=None):
        Vocalista.__init__(self, nombre, banda)
        Guitarrista.__init__(self, nombre, banda)
```

[ ]:
```
bruce = VocalistaYGuitarrista('Bruce Springsteen')
bruce.info()
```

[ ]:
```
mollo = VocalistaYGuitarrista("Ricardo Mollo", "Divididos")
mollo.info()
```

[ ]:
```
VocalistaYGuitarrista.__mro__
```

20 ¿Qué términos asociamos con la programación orientada a objetos?

21 Destacados ...

- Encapsulamiento
  - **class**, métodos privados y públicos, propiedades.
- Herencia
  - Clases bases y derivadas.
  - Herencia múltiple.
- **¿Alguno más?**

22 Polimorfismo

- Capacidad de los objetos de distintas clases de responder a mensajes con el mismo nombre.
- Ejemplo: + entre enteros y cadenas.

[ ]:
```
print("hola " + "que tal.")
print(3 + 4)
```

23 ¿Podemos sumar dos músicos?

[ ]:
```
adele = Musico("Adele")
sting = Musico("Sting", "The Police")

print(adele + sting)
```

24 Pero pdemos definir el método ___add___

[ ]:
```
class Musico:
    def __init__(self, nombre, puesto=None, banda=None):
        self.nombre = nombre
        self.tiene_banda = banda!=None
        self._banda = banda
        self.puesto = puesto

    def info(self):
        if self.tiene_banda:
            print (f"{self.nombre}  integra la banda {self.banda}")
        else:
            print(f"{self.nombre} es solista ")
```

```
    def __add__(self, otro):
        return (f"Ambos músicos son {self.nombre} y {otro.nombre}")

    @property
    def banda(self):
        if self.tiene_banda:
            return self._banda
        else:
            return "No tiene banda"
    @banda.setter
    def banda(self, banda):
        self._banda = banda
        self.tiene_banda = True
```

[ ]:
```
adele = Musico("Adele")
sting = Musico("Sting", "The Police")

print(adele + sting)
```

24.1 ¿Polimorfismo en nuestros músicos?

[ ]:
```
bruce.info()
brian.info()
```

25 Probamos en casa

¿Qué podemos decir de las variables de instancias cuyo nombre comienza con ___?

[ ]:
```
class A:
    def __init__(self, x, y, z):
        self.varX = x
        self._varY = y
        self.__varZ = z

    def demo(self):
        return f"ESTOY en A: x: {self.varX} -- y:{self._varY} --- z:{self.
 ↪__varZ}"

class B(A):
    def __init__(self):
        A.__init__(self, "x", "y", "z")

    def demo(self):
        return f"ESTOY en B: x: {self.varX} -- y:{self._varY} --- z:{self.
 ↪__varZ}"
```

[ ]:
```
objB = B()
print(objB.demo())
```

26 Herencia y propiedades

- Observemos este código: ¿qué imprime?, ¿qué significa?

[ ]:
```
class A:
    def __init__(self):
        self._x = 0

    @property
    def x(self):
        return self._x
    @x.setter
    def x(self, value):
        self._x = value

class B(A):
    def __init__(self):
        A.__init__(self)
```

[ ]:
```
obj = B()
#obj.x = 10
print(obj.x)
```

27 Para los que quieran seguir un poco más ...

- https://realpython.com/python-classes/
- https://realpython.com/inheritance-composition-python/
- https://realpython.com/instance-class-and-static-methods-demystified/

28 Seguimos la próxima ...