Tipos de métodos e interacciones entre objetos

Constructor, accesadores y mutadores

{desafío} latam_

Codificar un programa con clases, atributos y métodos utilizando colaboración y composición para resolver un problema de baja complejidad acorde al lenguaje Python.

- Unidad 1:
  Introducción a la programación orientada a objetos con Python
- Unidad 2:
  Tipos de métodos e interacciones entre objetos
- Unidad 3:
  Herencia y polimorfismo
- Unidad 4:
  Manejo de errores y archivos

{desafío} latam_

¿Qué aprenderás en esta sesión?

- *Codifica una clase con métodos constructores, accesadores y mutadores para resolver un problema.*

¿Qué necesitamos para dar valores a los atributos de una instancia?

/* Constructores */

Es importante saber qué son los constructores, accesadores y mutadores, pues estos tipos de métodos son ampliamente utilizados en programación orientada a objetos, ya que permiten incorporar lógica y reglas específicas para cada clase y el trabajo con sus instancias.

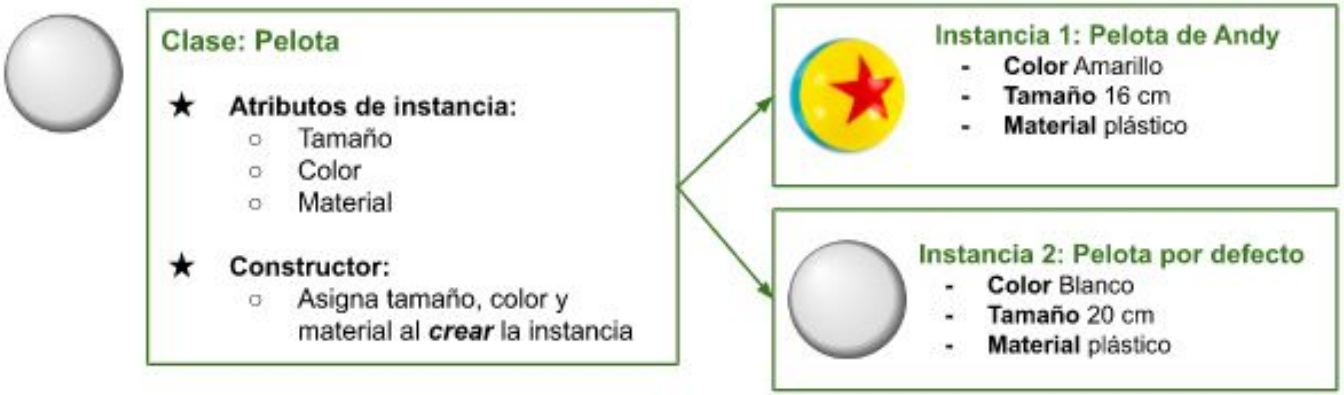
Método constructor

Qué es y para qué sirve

- Es un método que **se ejecuta automáticamente** al momento de crear una **instancia** de la clase, sin necesidad de ser llamado explícitamente.
- Su función es dar valores a los atributos de la instancia recién creada. Los valores que se dan a los atributos pueden ser simplemente un valor por defecto (definido en el constructor) o valores que se deben especificar explícitamente al momento de crear la instancia.
- Dependiendo del problema a resolver, existen dos opciones a utilizar:
  - valores por defecto
  - valores entregados explícitamente
  - además, se puede utilizar una combinación de ambas, es decir, valores por defecto que pueden ser sobreescritos por valores entregados.

Método constructor

Ejemplo



¿Cómo se define un método constructor en Python?

- Se debe definir exclusivamente con el nombre **`__init__`**.
- Debe tener como primer parámetro la instancia de la clase, **`self`**.
- No puede tener retorno (a menos que sea **None**).
- Por lo general, es el primer método definido dentro de la clase.

```
class Pelota():
    def __init__(self):
        print("¡Se ha creado una pelota!")

# Salida: "¡Se ha creado una pelota!"
p = Pelota()
```

Atributos de la clase

En el constructor normalmente se asigna valores a los atributos de la clase

a. Asignar valores en el cuerpo del método

```
class Pelota():
    def __init__(self):
        self.color = "blanco"
        self.tamanio = 20
        self.material = "plástico"

p = Pelota()

# Salida: blanco 20 plástico
print(p.color, p.tamanio, p.material)
```

Atributos de la clase

En el constructor normalmente se asigna valores a los atributos de la clase

b. Asignar valores desde parámetros del método

```
class Pelota():
    def __init__(self, color: str, tamanio: int, material: str):
        self.color = color
        self.tamanio = tamanio
        self.material = material

p = Pelota("Amarillo", 16, "plástico")

# Salida: Amarillo 16 plástico
print(p.color, p.tamanio, p.material)

# Salida:
# TypeError: __init__() missing 3 required positional arguments:
# 'color', 'tamanio', and 'material'
p2 = Pelota()
```

Atributos de la clase

En el constructor normalmente se asigna valores a los atributos de la clase

c. Asignar valores desde parámetros del método, con valores por defecto

```
class Pelota():
    def __init__(self, color: str, tamanio = 20, material =
"plástico"):
        self.color = color
        self.tamanio = tamanio
        self.material = material

p = Pelota("Amarillo", 16)

# Salida: Amarillo 16 plástico
print(p.color, p.tamanio, p.material)
```

Dependiendo de cada caso, en un constructor se pueden asignar valores a atributos directamente en el cuerpo, y/o mediante parámetros (con o sin valores por defecto). Esto está generalmente dado por las reglas del negocio y/o requerimientos.

/* Accesadores y mutadores */

Accesadores y mutadores

Accesadores (getters)

- Son métodos que permiten acceder al valor de un atributo en una instancia.
- Permiten acceder a información de los atributos expresada de forma diferente.

Mutadores (setters)

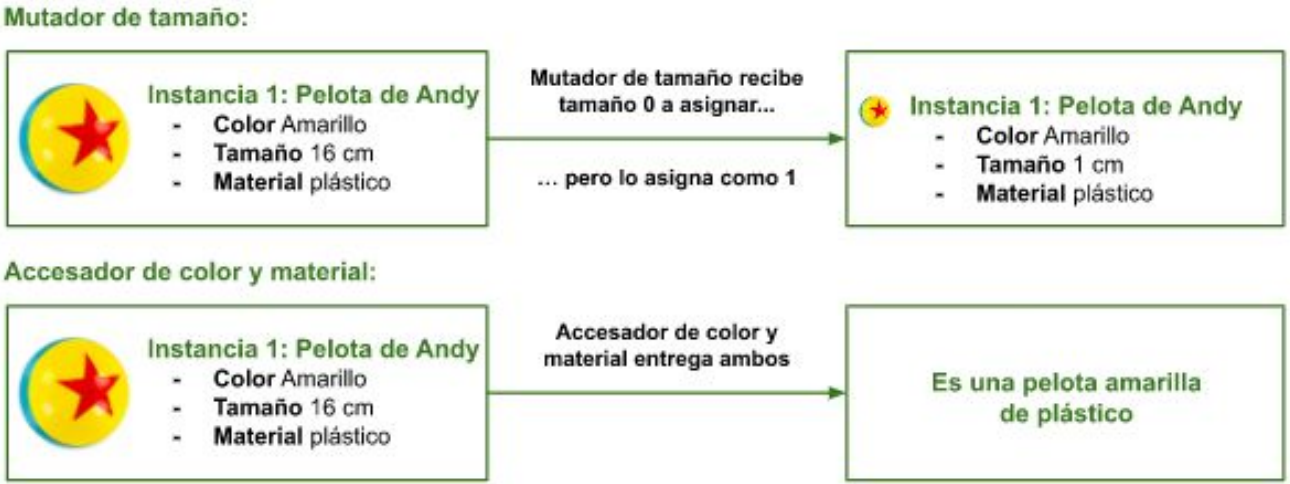
- Son métodos que permiten modificar el valor de un atributo en una instancia.
- Permiten aplicar reglas al momento de asignar un valor a un atributo.

Ambos se definen explícitamente dentro de la clase, y la forma de hacerlo dependerá del lenguaje de programación utilizado.

La principal razón para forzar el acceso o modificación de un atributo mediante un método, es impedir que el valor sea visto o modificado directamente desde la instancia de una clase (mediante sintaxis de punto `.`)

Accesadores y mutadores

Ejemplo



El decorador `@property`

- Permite definir un método accesador.
- Al utilizarlo se está “definiendo una propiedad de la clase”, donde una vez definido el accesador mediante el decorador @property, también es posible definir el método mutador asociado, usando un nuevo decorador a partir del nombre del accesador.

En la instancia creada, para hacer uso de un accesador basta con usar la sintaxis de punto `.`, y el nombre de la propiedad (sin paréntesis al final, tal como si fuera un atributo). En el caso del mutador, este es invocado al hacer uso del signo =.

{desafío} latam_

Si quieres profundizar sobre la implementación de este decorador, puedes consultar la documentación oficial en este enlace.

Definir accesador en Python

Se debe definir un método de instancia a continuación del decorador @property.

Ejemplo: Se crea un accesador para obtener el color y material de una pelota.

```
class Pelota():
   def __init__(self, color: str, material: str):
       self.color = color
       self.material = material

   @property
   def color_y_material(self):
       return f"Pelota {self.color} de {self.material}"

p = Pelota("Amarilla", "plástico")

# Salida: Pelota Amarilla de plástico
print(p.color_y_material)
```

{desafío} latam_

Definir mutador en Python

Se debe haber definido primero el accesador, ya que si no se desea aplicar ninguna lógica en el accesador, simplemente se retorna el valor del atributo. Luego, se debe usar como decorador el nombre del método definido para el accesador y añadir "**`.setter`**".

Ejemplo: Se define el método, el cual debe ser de instancia y debe tener el mismo nombre del accesador.

```python
class Pelota():
    def __init__(self):
        self.tamanio_pelota = 1

    @property
    def tamanio(self):
        return self.tamanio_pelota

    @tamanio.setter
    def tamanio(self, tamanio: int):
        self.tamanio_pelota = tamanio if tamanio > 0 else 1

p = Pelota()

# Salida: 1
print(p.tamanio)

p.tamanio = 0

# Salida: 1
print(p.tamanio)
```

Nombre de las propiedades

En el ejemplo anterior, se ve cómo se le asignó el nombre:

- Propiedad: “tamanio”, en sus métodos getter y setter.
- Atributo: “tamanio_pelota”.

🛑 Si se hubiese nombrado al atributo con el mismo nombre de la propiedad, es decir, “tamanio”, se habría producido un “error de recursividad” (RecursionError) al intentar modificar el valor de “tamanio”.
🛑 Por otro lado, si hubiéramos incluido el atributo “tamanio” dentro del constructor para recibir un valor al momento de asignar la instancia, se habría producido un error de tipo AttributeError, indicando que el atributo no existe.

Los problemas descritos anteriormente son posibles de solucionar, creando atributos de uso interno o “privados”. Esto, además permite que un atributo solo pueda ser accedido o modificado mediante su getter y setter respectivamente.

Ejercicio guiado

"Ingreso de medicamentos"

{desafío}
latam_

Ingreso de medicamentos

Desde la cadena farmacéutica, te han solicitado generar un prototipo de aplicación (mediante un script Python) que permita generar medicamentos a partir de datos ingresados por el usuario. Para ello, el script debe solicitar (mediante línea de comandos) que el usuario ingrese nombre del medicamento, precio y stock.

Con los datos de nombre y stock, debes crear una instancia de la clase Medicamento. Esta clase debe ser refactorizada, a partir de la generada en la unidad anterior en el Ejercicio guiado: ***Crear método que asigna descuento a un medicamento***, para cumplir con los siguientes requerimientos:

Ingreso de medicamentos

Requerimientos

1. Por defecto, cada medicamento debe tener un precio bruto de 0, precio neto de 0 y descuento de 0. Estos tres datos no se pueden modificar al momento de crear la instancia, por lo que toda instancia creada debe tener estos valores inicialmente.

2. Cada instancia creada debe también tener un nombre, el cual necesariamente debe ser entregado al momento de crear la instancia, y un stock. Si el stock no es entregado al momento de crear la instancia, se debe asignar un valor de 0.

Ingreso de medicamentos

Requerimientos

3. El precio bruto se debe asignar en una instancia ya creada. Al ser asignado, en caso de que sea mayor a 0, se debe asignar a la vez el precio bruto del medicamento, el descuento y el precio final. El descuento se debe aplicar sobre el precio bruto más el IVA, y solo se aplicará en caso de que el valor del medicamento esté entre $10.000 y $19.999 (10% de descuento), o sea mayor o igual a $20.000 (20% de descuento).

Finalmente, el script debe mostrar en pantalla el nombre del medicamento ingresado y su precio bruto. En caso de tener descuento, también lo debe mostrar en pantalla. Por último, debe mostrar también por pantalla el precio final (considerando IVA y descuento).

Ingreso de medicamentos

Solución

Paso 1

A partir del archivo **medicamento.py** modificar la clase **Medicamento** eliminando el atributo descuento y el método asigna precio.

```python
# archivo medicamento.py
class Medicamento():
    IVA = 0.18

    @staticmethod
    def validar_mayor_a_cero(numero: int):
        return numero > 0
```

Ingreso de medicamentos

Solución

Paso 2

Agregar constructor con parámetro nombre sin valor por defecto, y stock con valor por defecto de 0.
En el cuerpo del constructor, se debe asignar los valores ingresados por parámetro a los atributos de instancia nombre y stock, y asignar con valor 0 los atributos de instancia de los precios y el descuento.

```python
# archivo medicamento.py
class Medicamento():
    IVA = 0.18

    def __init__(self, nombre: str, stock: int = 0):
        self.nombre = nombre
        self.stock = stock
        self.precio_bruto = 0
        self.precio_final = 0.0
        self.descuento = 0.0

    @staticmethod
    def validar_mayor_a_cero(numero: int):
        return numero > 0
```

Ingreso de medicamentos

Solución

Paso 3

A continuación del código anterior, agregar property precio, para así definir el getter del atributo **precio_final**.

```
@property
    def precio(self):
        return self.precio_final
```

Paso 4

A continuación del código anterior, agregar setter de la propiedad precio. El método asociado debe recibir como parámetro un precio bruto, dentro de este se debe agregar una validación de que el precio ingresado por parámetro sea mayor a 0.

```
@precio.setter
    def precio(self, precio_bruto: int):
        if self.validar_mayor_a_cero(precio_bruto):
```

Ingreso de medicamentos

Solución

Paso 5

Dentro del bloque “if” del paso anterior, asignar el valor del atributo **precio_bruto** desde el valor ingresado por parámetro, y el del atributo **precio_final** considerando el IVA.

```
self.precio_bruto = precio_bruto
            self.precio_final = precio_bruto + precio_bruto *
self.IVA
```

Paso 6

A continuación del código anterior, dentro del bloque if, asignar valor de descuento, si corresponde, de acuerdo al precio final obtenido.

```
if self.precio_final >= 10000 and self.precio_final < 20000:
                self.descuento = 0.1

            elif self.precio_final >= 20000:
                self.descuento = 0.2
```

{desafío} latam_

Ingreso de medicamentos

Solución

Paso 7

A continuación del código anterior, aplicar descuento.

```
if self.descuento:
                self.precio_final *= 1 - self.descuento
```

Paso 8

En un archivo llamado **programa.py**, importar la clase **Medicamento**.

```
# archivo programa.py
from medicamento import Medicamento
```

Ingreso de medicamentos

Solución

Paso 9

Solicitar al usuario ingresar el nombre del medicamento, el stock y el precio bruto, luego, almacenar cada ingreso en una variable, y transformar el stock y el precio bruto a número entero.

```
nombre = input("Ingrese nombre del medicamento:\n")
stock = int(input("Ingrese stock del medicamento:\n"))
precio_bruto = int(input("Ingrese precio bruto del medicamento:\n"))
```

Ingreso de medicamentos

Solución

Paso 10

A continuación del código anterior, crear una instancia de medicamento y asignar el precio a partir del precio bruto.

```
m1 = Medicamento(nombre, stock)
m1.precio = precio_bruto
```

Paso 11

A continuación del código anterior, mostrar en pantalla el nombre del medicamento y el precio bruto.

```
print(f"El precio bruto del medicamento {m1.nombre} es {m1.precio_bruto}")
```

Ingreso de medicamentos

Solución

Paso 12

A continuación del código anterior, si tiene descuento, mostrar en pantalla.

```
if m1.descuento:
   print(f"Tiene un descuento de {m1.descuento * 100}%")
```

Paso 13

A continuación del código anterior, mostrar en pantalla el precio final.

```
print(f"El precio final del medicamento es {m1.precio_final}")
```

Ingreso de medicamentos

Solución

Ejemplo de salida

```
Ingrese nombre del medicamento:
anticonceptivos
Ingrese stock del medicamento:
100
Ingrese precio bruto del medicamento:
15000
El precio bruto del medicamento anticonceptivos es 15000
Tiene un descuento de 10.0%
El precio final del medicamento es 15930.0
```

¿De qué forma se puede asignar valores a los atributos al momento de crear una instancia?

¿Qué ventajas tiene acceder o modificar valores de atributos mediante accesadores y mutadores?

Próxima sesión...

- *Codifica una clase utilizando sobrecarga de métodos para resolver un problema.*

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales