Tipos de métodos e interacciones entre objetos

Sobrecarga de métodos

{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

Te encuentras aquí

{desafío} latam_

¿Qué aprenderás en esta sesión?

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

{desafío} latam_

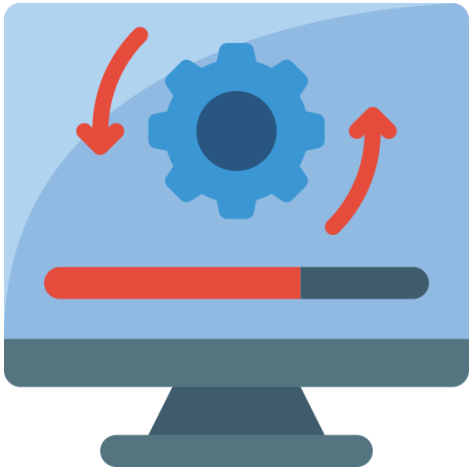
¿Qué entendemos por sobrecarga de métodos?

{desafío} latam_

/* Sobrecarga de métodos */

Sobrecarga

- Re-definir los métodos especiales de Python dentro de una clase.
- Al aplicar el método especial en una instancia de una clase que lo ha sobrecargado, se hará un llamado a la definición del método dentro de la clase a la cual pertenece la instancia.
- Mediante una sintaxis sencilla, se puede dar un comportamiento propio a ciertas operaciones.



Métodos especiales en Python

Categoría de métodos especiales	Ejemplos
Creación, eliminación y representación de objetos	`__init__`, `__del__`, `__str__`
Acceso a atributos	`__getattr__`, `__setattr__`
Atributos del descriptor	`__get__`, `__set__`, `__delete__`
Secuenciar y mapear objetos	`__len__`, `__getitem__`, `__contains__`
Iteración	`__iter__`
Operaciones matemáticas	`__add__`, `__pow__`, `__int__`
Comparaciones	`__lt__`, `__eq__`, `__ne__`
Llamado de objetos	`__call__`

Sobrecarga de métodos especiales

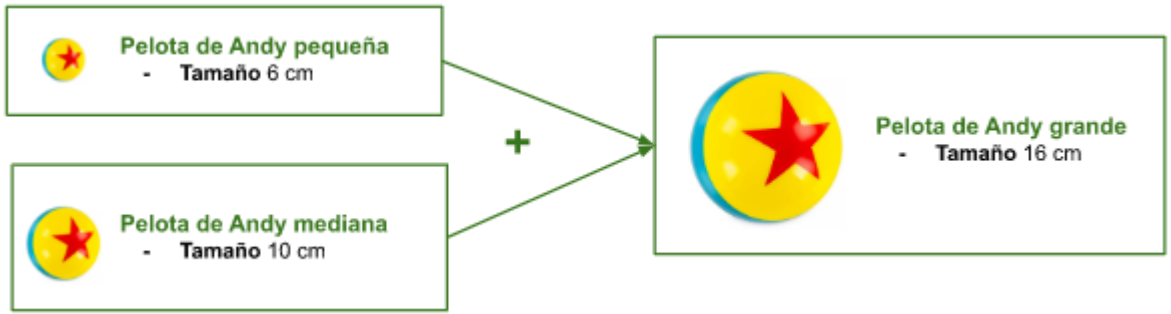
Se debe utilizar el mismo nombre (incluyendo los underscore).
Dependiendo del método, también se debe respetar la cantidad y tipos de datos de los parámetros, así como la cantidad y tipos de datos de los retornos, sin embargo, la forma en que se realiza el llamado del método se mantiene y no se puede sobreescribir.

```
class MiClase():
   def __str__(self):
       return "¡Oh no, el método str se ha sobrecargado!"

a = MiClase()

# Salida "¡Oh no, el método str se ha sobrecargado!"
print(a)
```

Para el caso de la sobrecarga del método `__str__`, el retorno necesariamente debe ser un string.

¿Cómo se definen los operadores en Python?

Cada vez que se hace uso de algún operador en Python, se está haciendo el llamado a un método especial. En el caso de la suma +, se hace el llamado al método __add__.
Por lo tanto, los operadores en Python se definen como métodos especiales, los cuales son posibles de sobrecargar.



¿Cómo sobrecargar operadores en Python?

Método other

```
class Pelota():

    def __init__(self, tamanio:int = 0):
        self.tamanio = tamanio

    def __add__(self, other):
        return self.tamanio + other.tamanio

p1 = Pelota(16)
p2 = Pelota(32)

# Salida: 48
print(p1 + p2)
```

```
class Coordenada():
    def __init__(self, x = 0, y = 0):
        self.x = x
        self.y = y

    def __add__(self, other):
        nuevo_x = self.x + other.x
        nuevo_y = self.y + other.y
        return Coordenada(nuevo_x, nuevo_y)

c1 = Coordenada(5, 10)
c2 = Coordenada(-5, 10)
suma_coordenadas = c1 + c2

# Salida: <class '__main__.Coordenada'>
print(type(suma_coordenadas))

# Salida: (0, 20)
print(f"({suma_coordenadas.x},{suma_coordenadas.y})")
```

¿Cómo sobrecargar operadores en Python?

Método __eq__

```
class Pelota():
    def __init__(self, tamanio = 0):
        self.tamanio = tamanio

p1 = Pelota(16)
p2 = Pelota(16)
p3 = p2

# Salida: False
print(p1 == p2)
# Salida: True
print(p2 == p3)
```

```
class Pelota():
   def __init__(self, tamanio = 0):
       self.tamanio = tamanio

   def __eq__(self, other):
       return self.tamanio == other.tamanio

p1 = Pelota(16)
p2 = Pelota(16)

# Salida: True
print(p1 == p2)
```

¿Cómo sobrecargar operadores en Python?

```
class Coordenada():
   def __init__(self, x = 0, y = 0):
       self.x = x
       self.y = y

   def __eq__(self, other):
       compara_x = self.x == other.x
       compara_y = self.y == other.y
       return compara_x and compara_y

c1 = Coordenada(-5, 10)
c2 = Coordenada(-5, 10)

# Salida: True
print(c1 == c2)
```

Para el caso de la clase **Coordenada**, que tiene dos atributos, se puede usar el operador and dentro de la implementación del método, de forma que dos instancias de coordenada sean equivalentes si ambas tienen los mismos valores de x e y respectivamente.

Ejercicio guiado

"Stock de medicamentos"

{desafío} latam_

Stock de medicamentos

Desde la cadena farmacéutica para la cual has venido desarrollando un programa para manejar los medicamentos en venta, te solicitan esta vez crear un prototipo que permita manejar el stock de medicamentos a medida que son ingresados. En este contexto, el cliente especifica que el programa debe considerar lo siguiente:

- Para ingresar un medicamento se debe solicitar nombre y stock, y añadir a un listado de medicamentos ingresados.
- Dos medicamentos se consideran iguales si tienen el mismo nombre (insensible a mayúsculas).
- Solo se debe solicitar (y asignar) el precio (considerando lo mismo del programa anterior en cuanto a IVA, descuento y precio final) en caso de que se ingrese un medicamento que no existe.
- Si se ingresa un medicamento que ya existe, se debe agregar al stock del medicamento existente el del medicamento del nuevo ingreso.
- Luego de cada ingreso (de un medicamento nuevo o existente), se debe informar nombre del medicamento ingresado, precio bruto, descuento (si posee), precio final, stock, y cantidad de medicamentos distintos que tiene ingresados la farmacia.
- El programa debe permitir un nuevo ingreso de medicamento hasta que el usuario indique lo contrario.

Stock de medicamentos

Solución

Paso 1

A partir de la clase Medicamento del archivo **medicamento.py** del ejercicio guiado “Ingreso de medicamento”, sobrecarga el método **__eq__**, de forma que dos instancias de Medicamento se consideran iguales si sus nombres son iguales (independiente de mayúsculas y minúsculas).

```python
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
    @property
    def precio(self):
        return self.precio_final
    @precio.setter
    def precio(self, precio_bruto: int):
        if self.validar_mayor_a_cero(precio_bruto):
            self.precio_bruto = precio_bruto
            self.precio_final = precio_bruto + precio_bruto * self.IVA

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

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

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

    # Este es el método que se debe agregar
    def __eq__(self, other):
        return self.nombre.lower() == other.nombre.lower()
```

{desafío} latam_

Stock de medicamentos

Solución

Paso 2

A continuación del código anterior, sobrecargar el método **`__iadd__`** (permite sobrecargar operación de asignación +=), de forma que, al sumar dos instancias iguales de Medicamento, se añada al stock de la instancia actual el stock de la segunda instancia. El método debe retornar la instancia actual.

```
def __iadd__(self, other):
        if self == other:
            self.stock += other.stock
        return self
```

Stock de medicamentos

Solución

Paso 3

En el archivo **programa.py**, luego de importar la clase Medicamento, consultar mediante input si el usuario desea ingresar un medicamento y almacenar la opción ingresada en una variable. También iniciar una lista vacía donde se irán almacenando los medicamentos ingresados.

```
# archivo programa.py
from medicamento import Medicamento
opcion_ingreso = int(input("¿Desea agregar un medicamento?"
"\n1. Sí\n2. No\n"))
ingresados = []
```

Stock de medicamentos

Solución

Paso 4

A continuación del código anterior, iniciar un ciclo while que tenga como condición de ingreso que la opción ingresada en el paso anterior sea 1. Dentro del ciclo, consultar nombre y stock, y crear una instancia de Medicamento con estos datos.

```
while opcion_ingreso == 1:
    nombre = input("\nIngrese nombre del medicamento:\n")
    stock = int(input("\nIngrese stock del medicamento:\n"))
    m = Medicamento(nombre, stock)
```

Stock de medicamentos

Solución

Paso 5

A continuación del código anterior, consultar si el nuevo medicamento se encuentra ya ingresado (Nota: el operador in desencadena un llamado a __eq__). De ser así, realizar la suma de stocks y reemplazar el medicamento ingresado por el existente.

```
if m in ingresados:
      indice = ingresados.index(m)
      ingresados[indice] += m
```

Stock de medicamentos

Solución

Paso 6

En caso de que no se cumpla la condición anterior, solicitar el precio, asignar y agregar el medicamento a la lista de ingresados.

```
else:
      ingresados.append(m)
      precio_bruto = int(input("\nIngrese precio bruto del medicamento:\n"))
      m.precio = precio_bruto
```

Stock de medicamentos

Solución

Paso 7

Luego de finalizar el ingreso, muestre en pantalla los datos del medicamento ingresado, y la cantidad de medicamentos distintos ingresados.

```
print(f"\n***** DATOS MEDICAMENTO {m.nombre} *****")
   print(f"PRECIO BRUTO: ${m.precio_bruto}")
   if m.descuento:
       print(f"DESCUENTO: {m.descuento*100}%")
   print(f"PRECIO FINAL: ${m.precio_final}")
   print(f"STOCK: {m.stock}")

   print(f"\nLa farmacia cuenta con {len(ingresados)} medicamento(s)\n")
```

Stock de medicamentos

Solución

Paso 8

Por último, al final del ciclo, consultar nuevamente si se desea ingresar o no un nuevo medicamento.

```
opcion_ingreso = int(input("¿Desea agregar un medicamento?"
"\n1. Sí\n2. No\n"))
```

¿A qué se refiere la sobrecarga en Python?

¿Qué permite la sobrecarga de operadores?

Próxima sesión...

- *Desafío guiado.*

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales