Introducción a la programación orientada a objetos con Python

Métodos (Parte II)

{desafío} latam_

Describir los conceptos fundamentales del paradigma de orientación a objetos haciendo la distinción entre Clase y de Objeto.

- 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

¿Qué aprenderás en esta sesión?

- *Explica el concepto de método de una clase haciendo la distinción con el concepto de comportamiento de un objeto.*
- *Identifica los principios de abstracción y encapsulamiento de acuerdo al paradigma de orientación a objetos.*

{desafío} latam_

¿Cuáles son los tipos de métodos?

{desafío} latam_

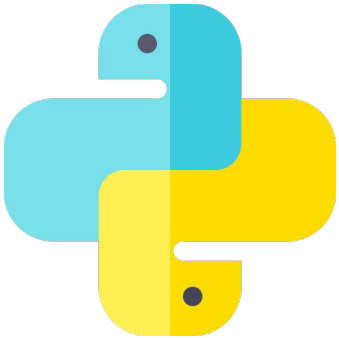
/* Métodos no estáticos */

¿Qué es un método no estático?

- Puede ser llamado sólo desde una instancia de una clase.
- También se les llama métodos de instancia.
- Son capaces de modificar el valor de los atributos de una instancia de la clase
- Son capaces de acceder a los valores de estos atributos en cada instancia específica.
- Sus retornos pueden variar entre distintas instancias de la clase, ya que cada instancia puede tener distintos estados (valores en sus atributos).

¿Cómo definir un método no estático en Python?

- Se debe definir el método al igual que se define cualquier función, pero dentro de una clase.
- Puede o no tener retorno, sin embargo, en cuanto a los parámetros se requiere que como mínimo tenga el parámetro self.
- Hace referencia a una instancia específica de la clase y permite acceder a otros métodos de la clase o a los atributos, de esta forma, se puede también distinguir cuando se hace referencia a un atributo, en lugar de a una variable dentro del alcance local de un método.



¿Cómo definir un método no estático en Python?

Ejemplo

```
class Pelota():

    # Método de instancia que asigna color
    def asigna_color(self, nuevo_color: str):
        self.color = nuevo_color

    # Método de instancia que lee color de la instancia
    def lee_color(self):
        print("El color de esta pelota es {}".format(self.color))

    def lee_color_local_y_atributo(self, color_local: str):
        # Esta variable "color" sólo existe en el alcance del método
        color = color_local

        # Un método de instancia puede llamar a otros métodos
        self.lee_color()
        print("El color {} NO es el color de ESTA pelota".format(color))
```

¿Cómo definir un método no estático en Python?

Ejemplo

```
# Se crea instancia
pelota_multicolor = Pelota()

# Se asigna color a la instancia
pelota_multicolor.asigna_color("rojo")

# Se lee color. La salida será "El color de esta pelota es rojo"
pelota_multicolor.lee_color()

pelota_multicolor.asigna_color("verde")

# Las salidas serán:
#    El color de esta pelota es verde
#    El color amarillo NO es el color de ESTA pelota
pelota_multicolor.lee_color_local_y_atributo("amarillo")
```

Cuando se llama a un método de instancia, solo se debe pasar como argumento los parámetros definidos en el método que sean distintos a `self`., es decir, el argumento correspondiente a self se asigna implícitamente.

Ejercicio guiado

"Crear método que asigna un precio válido"

Crear método que asigna un precio válido

Desde la cadena farmacéutica donde se está desarrollando el software que permite manejar el stock de medicamentos, se te solicita que el programa permita asignar un precio ingresado por el usuario a cada instancia creada. Sin embargo, no es posible usar una asignación simple de valor al atributo, ya que antes de asignar un precio a cualquier medicamento se debe validar que este sea un precio válido (mayor a 0).

- Para cumplir con lo requerido, agrega un método de instancia al código existente hasta el momento, que dentro de su lógica llame al método que valida un precio positivo.
- Si el retorno del llamado es True, se asigna el valor ingresado por parámetro al valor de precio de la instancia de la clase, y si el retorno es False, no se asigna el valor y se muestra un mensaje en pantalla.

Crear método que asigna un precio válido

Solución

Paso 1

Definir clase Medicamento según el ejercicio guiado de la primera sesión.

```
class Medicamento():
    descuento = 0.05
    IVA = 0.18
    @staticmethod
    def validar_mayor_a_cero(numero: int):
        return numero > 0
```

Paso 2

A continuación del código anterior, a nivel de la clase, definir método de instancia que asigna al precio de la instancia el precio ingresado como parámetro del método.

```
def asigna_precio(self, precio_entregado: int):
```

Crear método que asigna un precio válido

Solución

Paso 3

Dentro del método anterior, hacer llamado a método que valida que un número sea mayor a 0, dando como argumento el precio ingresado por parámetro, y almacenar retorno en una variable local.

```
def asigna_precio(self, precio_entregado: int):
        es_valido = self.validar_mayor_a_cero(precio_entregado)
```

Paso 4

Dentro del método asigna_precio, evaluar si es_valido es igual a True. De ser así, asignar el parámetro ingresado a la variable precio de la instancia.

```
if es_valido:
            self.precio = precio_entregado
```

{desafío} latam_

Crear método que asigna un precio válido

Solución

Paso 5

Concatenar al if declarado una cláusula else, dentro de la cual se ejecute un print indicando que el precio ingresado por parámetro no es válido.

```
# archivo medicamento.py
class Medicamento():

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

    def asigna_precio(self, precio_entregado: int):
        es_valido = self.validar_mayor_a_cero(precio_entregado)

        if es_valido:
            self.precio = precio_entregado
        else:
            print("El precio '{}' no es un precio válido".format(
                precio_entregado))
```

{desafío} latam_

Crear método que asigna un precio válido

Solución

Paso 6

En un script ejecucion.py, importar la clase Medicamento desde el archivo medicamento.py

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

Paso 7

Crear una instancia de medicamento

```
medicamento_nuevo = Medicamento()
```

Crear método que asigna un precio válido

Solución

Paso 8

Almacenar en una variable, como entero, el precio ingresado por el usuario

```
precio = int(input("Ingrese precio del medicamento\n"))
```

Paso 9

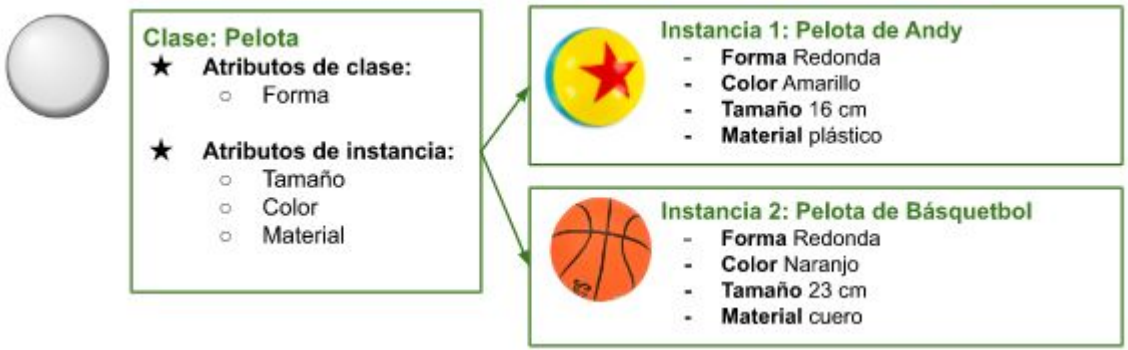
Hacer la asignación en la instancia

```
medicamento_nuevo.asigna_precio(precio)
```

/* Atributo no estático */

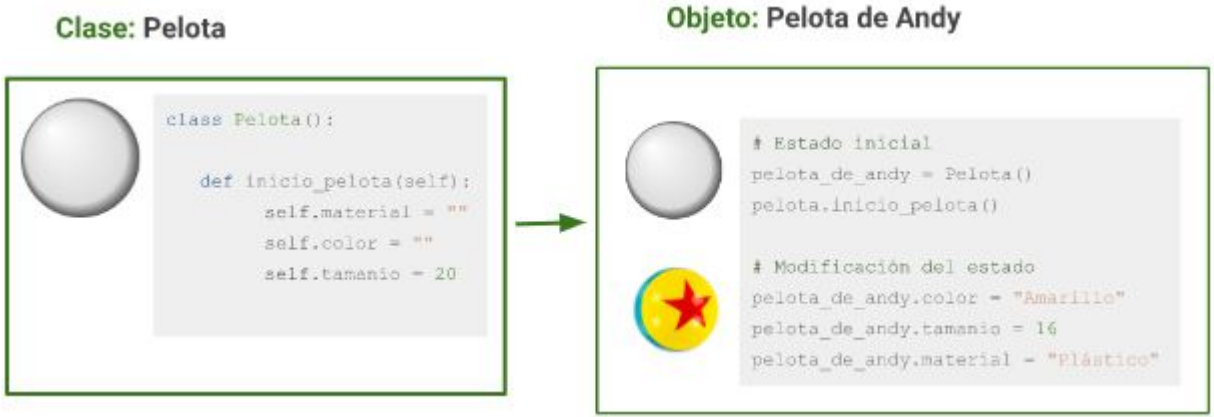
Atributos de instancia o no estáticos

Son aquellos que requieren necesariamente de una instancia de la clase u objeto, para poder acceder a ellos o asignarles un valor. Por ello, se dice que los atributos de instancia (o no estáticos) son únicos para cada instancia.



Estado de un objeto

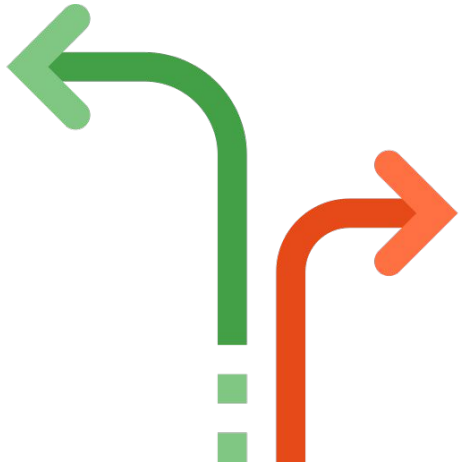
- Se determina por los valores que poseen sus atributos en un momento específico.
- Al crear un objeto, éste se encuentra en su estado inicial y los valores de sus atributos estáticos serán aquellos que se determinen en la definición de la clase, y si se modifican los valores de los atributos del objeto instanciado, entonces se está modificando su estado.



Diferencia entre atributo y estado

- **Atributo:** es el que define una característica específica de un objeto, y contiene un valor.

- **Estado:** corresponde a los valores que tienen el conjunto de atributos de un objeto en un momento dado, es decir, el conjunto de valores de los atributos son los que definen el estado del objeto.



Objetos sin estado en Python

En Python **necesariamente se le debe asignar un valor a un atributo dentro de la definición de su clase** (aunque sea None). Por lo tanto, la única forma de instanciar un objeto sin estado en Python sería creando una instancia de una clase que no posea atributos.

Al igual que para el caso de los métodos no estáticos, para definir o acceder a un atributo de instancia dentro de la definición de la clase, se debe hacer uso del parámetro self. Dado que self es un parámetro de un método de instancia, significa que un atributo de instancia necesariamente debe definirse dentro de un método de instancia.

Definir un atributo que pertenezca a cada instancia

Ejemplo

Se puede ver que el método **asigna_color** realiza exactamente lo mismo que lo visto previamente en el contenido de métodos no estáticos.

El atributo color no se ha definido a nivel de la clase, sino que directamente como atributo de una instancia específica (mediante `self`) dentro del método **asigna_color**.

```
class Pelota():
    def asigna_color(self, nuevo_color: str):
        self.color = nuevo_color
```

Definir un atributo que pertenezca a cada instancia

Ejemplo

De la misma forma, dentro de la definición de la clase, solo se puede acceder al valor de un atributo de instancia dentro de un método de instancia, mediante el parámetro **`self`**.
Por tanto, dentro de la definición de la clase, la forma de acceder a un atributo de clase y a un atributo de instancia es la misma.

```
class Pelota():
    forma = "redondeada"

    def asigna_color(self, nuevo_color: str):
        self.color = nuevo_color

    def lee_color_y_forma(self):
        print("El color de esta pelota es {}".format(self.color))
        print("La forma de esta pelota es {}".format(self.forma))
```

Definir un atributo que pertenezca a cada instancia

Ejemplo

En este caso, no es posible acceder al atributo color sin haber creado una instancia de Pelota y sin haber llamado al método **asigna_color**.

```
from pelota import Pelota
Pelota.color
```

El código anterior produce la siguiente salida:

```
----> 1 Pelota.color

AttributeError: type object 'Pelota' has no attribute 'color'
```

Definir un atributo que pertenezca a cada instancia

Ejemplo

Una vez creada la instancia y asignado el color mediante el método **asigna_color**, se puede leer el atributo color mediante el método **lee_color**, donde la forma correcta de uso sería:

```
from pelota import Pelota

p = Pelota
p.asigna_color("rojo")

# Salida: El color de esta pelota es rojo
p.lee_color()
```

Ejercicio guiado

"Crear método que asigna descuento a un medicamento"

Crear método que asigna descuento a un medicamento

Continuando con el desarrollo de la farmacéutica, se pide esta vez que en el método que asigna el precio válido, se asigne un descuento siguiendo las siguientes reglas:

- 10% de descuento si el medicamento cuesta entre $10.000 y $19.999
- 20% de descuento si el medicamento cuesta entre $20.000 y $29.999
- 30% de descuento si el medicamento cuesta $30.000 o más

El descuento se debe definir como un número decimal (0.1 si es 10%, 0.2 si es 20% y 0.3 si es 30%).

Crear método que asigna descuento a un medicamento

Solución

Paso 1

Definir clase Medicamento según ejercicio guiado anterior.

```
# archivo medicamento.py
class Medicamento():
    descuento = 0.05
    IVA = 0.18
    @staticmethod
    def validar_mayor_a_cero(numero: int):
        return numero > 0
    def asigna_precio(self, precio_entregado: int):
        es_valido = self.validar_mayor_a_cero(precio_entregado)
        if es_valido:
            self.precio = precio_entregado
        else:
            print("El precio '{}' no es un precio válido".format(
                precio_entregado))
```

Crear método que asigna descuento a un medicamento

Solución

Paso 2

Si el precio es válido, definir el atributo de instancia “descuento” con valor por defecto 0 (se muestra solo bloque if del método **asigna_precio** del código anterior)

```
if es_valido:
            self.precio = precio_entregado
            self.descuento = 0.0
```

Crear método que asigna descuento a un medicamento

Solución

Paso 3

Aplicar reglas de negocio entregadas para el valor del descuento de acuerdo al valor del precio para la instancia.

```
if es_valido:
                 self.precio = precio_entregado
                 self.descuento = 0.0

                if self.precio >= 10000 and self.precio < 20000:
                    self.descuento = 0.1
                elif self.precio >= 20000 and self.precio < 30000:
                    self.descuento = 0.2
                elif self.precio >= 30000:
                    self.descuento = 0.3
```

Modificar el estado de un objeto mediante __setattr__

Se podría pensar que al modificar el valor mediante una asignación, no se está haciendo uso de un método, sin embargo, se está haciendo uso de un “método mágico” o “método especial” de Python, llamado **`__setattr__`**.

Estos métodos pertenecen a todas las clases que se creen en Python, y tienen un funcionamiento por defecto, normalmente desencadenado al llamar al nombre del método explícitamente.

Modificar el estado de un objeto mediante __setattr__

Ejemplo

Para hacer la asignación, se debe hacer uso de la sintaxis de punto para acceder al atributo de la instancia, al cual se asigna el valor con el símbolo “=”.

```
class Pelota():
    def inicia_color(self):
        self.color = ""

# Se crea objeto y se asigna valor por defecto a atributo color
pelota_de_andy = Pelota()
pelota_de_andy.inicia_color()

# Se modifica estado asignando un nuevo color
pelota_de_andy.color = "Amarillo"
```

Ejercicio guiado

"Tienda de artículos para mascotas"

{desafío} latam_

Tienda de artículos para mascotas

La tienda “Nuestras mascotas” se dedica a la venta de productos para perros, gatos y mascotas exóticas. Actualmente, se está evaluando la idea de realizar una tienda virtual, para lo cual se requiere primero realizar un prototipo en Python que permita ingresar una orden de compra. Cada orden de compra tiene un identificador (un código numérico), un total de productos, un monto, y un código de descuento. El código de descuento, por defecto una cadena vacía, puede ser “10PORCIENTO” si el monto es superior a 10.000, o “20PORCIENTO” si es mayor a 20.000.

Para crear este programa, se debe crear la clase que permita definir la estructura de cada orden de compra creada, y un script que, al ser ejecutado, para ello, cree una instancia de esa clase y asigne a la instancia los valores para el identificador, el total de productos, el monto y el código de descuento (si corresponde). Estos valores deben ser solicitados desde la línea de comandos al momento de ejecutar el script.

Tienda de artículos para mascotas

Solución

Paso 1

Crear archivo orden_compra.py y escribir en él la definición de la clase OrdenCompra.

```
# archivo orden_compra.py
class OrdenCompra():

    # Se crea un método de instancia para definir atributos
    def nueva_orden(self):
        # Se define los atributos de instancia
        self.identificador = 0
        self.total_productos = 0
        self.monto = 0
        self.codigo_descuento = ""
```

Tienda de artículos para mascotas

Solución

Paso 2

Crear archivo generar_orden.py que importa a la clase OrdenCompra desde orden_compra.

```
# archivo generar_orden.py
from orden_compra import OrdenCompra
```

Paso 3

En el script anterior, a continuación, crear objeto de tipo OrdenCompra, y llamar al método nueva_orden.

```
oc = OrdenCompra()
oc.nueva_orden()
```

Tienda de artículos para mascotas

Solución

Paso 4

A continuación de la línea anterior, solicitar al usuario el valor para el identificador, mediante la función input().

```
oc.identificador = int(input("Ingrese identificador de la OC:\n"))
```

Paso 5

En las dos líneas siguientes, repetir el paso anterior para el total de productos y el monto.

```
oc.total_productos = int(input("Ingrese total de productos:\n"))
oc.monto = int(input("Ingrese monto:\n"))
```

Tienda de artículos para mascotas

Solución

Paso 6

Determinar si se debe aplicar un código de descuento. El valor del código de descuento no se debe solicitar al usuario, sino que lo determina el programa según lo ingresado por el usuario en el atributo monto.

```
if oc.monto > 20000:
    oc.codigo_descuento = "20PORCIENTO"
elif oc.monto > 10000:
    oc.codigo_descuento = "10PORCIENTO"
```

Ejemplo de salida tras una ejecución del archivo generar_orden.py:

```
Ingrese identificador de la OC:
112233
Ingrese total de productos:
100
Ingrese monto:
5000
```

Ejercicio guiado

"Tienda de artículos para mascotas"

Tienda de artículos para mascotas

Refactorización de clase OrdenCompra

Desde la tienda Nuestras Mascotas, te solicitan refactorizar la clase OrdenCompra, de tal forma, que la lógica que aplica un código de descuento se ejecute cada vez que se asigne un monto, lo cual se debe realizar mediante un método propio, en lugar del operador símbolo “=”. De la misma forma, te solicitan que solo las instancias que tengan un monto asignado mediante este método cuenten con el atributo codigo_descuento. El script que instancia la clase y solicita los valores al usuario, debe llamar al nuevo método creado al momento de asignar el monto, en lugar de hacer uso del símbolo “=”.

Tienda de artículos para mascotas

Solución

Paso 1

Del ejercicio guiado anterior, eliminar el atributo **codigo_descuento**, y definir método de instancia **asigna_monto**.

```
class OrdenCompra():

    def nueva_orden(self):
        self.identificador = 0
        self.total_productos = 0
        self.monto = 0

    def asigna_monto(self, nuevo_monto: int):
```

Tienda de artículos para mascotas

Solución

Paso 2

Dentro del método **asigna_monto**, asignar el monto ingresado al atributo, y definir el código de descuento como una cadena vacía.

```
def asigna_monto(self, nuevo_monto: int):
        self.monto = nuevo_monto
        self.codigo_descuento = ""
```

Paso 3

Dentro del mismo método, incluir lógica que asigna descuento (tomada del script **generar_orden.py**)

```
if nuevo_monto > 20000:
           self.codigo_descuento = "20PORCIENTO"
        elif nuevo_monto > 10000:
           self.codigo_descuento = "10PORCIENTO"
```

{desafío} latam_

Ejercicio guiado

"Tienda de artículos para mascotas"

{desafío} latam_

Tienda de artículos para mascotas

Refactorización de script generar_orden.py

Desde la tienda Nuestras Mascotas, te solicitan refactorizar el script **generar_orden.py**, de tal forma que se pueda modificar el valor del monto mediante el método **asigna_monto**. Por ahora, el identificador y el total de productos deben mantener su forma de asignación.

Se solicita además que una vez asignado el monto se muestre en pantalla el código de descuento.

Tienda de artículos para mascotas

Solución

Paso 1

A partir del script **generar_orden.py** desarrollado anteriormente, elimina la asignación del monto y del código de descuento.

```
# archivo generar_orden.py
from orden_compra import OrdenCompra
oc = OrdenCompra()
oc.nueva_orden()
oc.identificador = int(input("Ingrese identificador de la OC:\n"))
oc.total_productos = int(input("Ingrese total de productos:\n"))
```

Paso 2

Almacena en una variable el ingreso del usuario para el valor del monto.

```
monto = int(input("Ingrese monto:\n"))
```

Tienda de artículos para mascotas

Solución

Paso 3

Desde la instancia oc, llama al método asigna_monto, usando como argumento la variable monto.

```
oc.asigna_monto(monto)
```

Paso 4

Utiliza la instancia oc para mostrar en pantalla el valor del atributo codigo_descuento.

```
print(oc.codigo_descuento)
```

¿Cómo se puede llamar a un método de instancia?

{desafío} latam_

¿Cuándo se modifica o cambia el estado de un objeto?

Próxima sesión...

- *Desafío evaluado.*

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales