Introducción a la programación orientada a objetos con Python

Métodos (Parte I)

{desafío} latam_

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

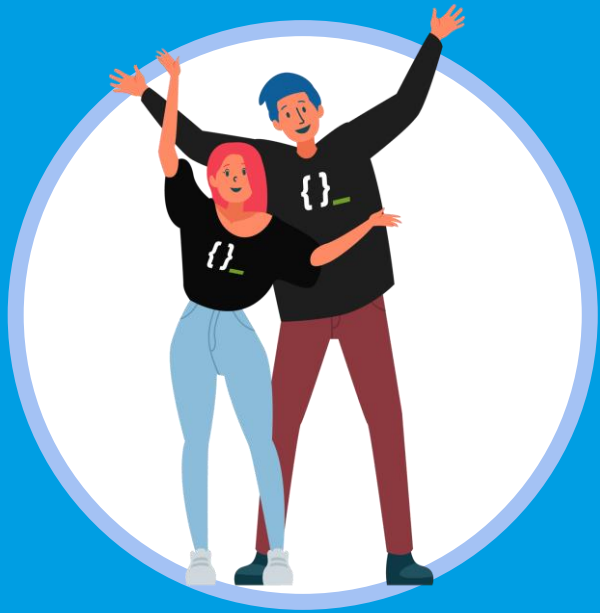
- Unidad 1:
  Introducción a la programación orientada a objetos con Python
- Unidad 2:
  Tipos de métodos e interacciones entre objetos
- Unidad 3:
  Herencia y polimorfismo
- Unidad 4:
  Manejo de errores y archivos

¿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_

¿Qué entendemos por métodos?



{desafío} latam_

/* Métodos */

Los métodos

Corresponden a un bloque de código que **permite realizar una tarea específica**, los que pueden tener o no un retorno y pueden ser con o sin parámetros.

Tipos de métodos

- Estático
- No estático

Un método por definición, pertenece a una clase, por lo que hablar de "**método de una clase**" puede ser redundante y, por eso, simplemente se habla de "métodos".

Los métodos

Ejemplo

En la siguiente imagen, se ilustra cómo para la **clase Pelota**, se puede establecer dos métodos, uno estático y otro no estático.

- El método no estático no altera los valores de los atributos de la clase Pelota.
- El método estático sí afecta el atributo posiciones.

Clase: Pelota

★ **Atributos:**
- `posiciones`: Lista de números

★ **Método estático:**
- `crear_rebote`: Genera lista de posiciones (números) que permiten generar un rebote válido (el primer número debe ser mayor a 0, y luego intercala 0 con números mayores a 0, cada vez menores, hasta llegar a 0).

★ **Método no estático:**
- `rebotar`: Asigna lista de posiciones válidas (que generan un rebote), a una pelota específica en su atributo `posiciones`.

Métodos de una clase y diferencia con funciones

Los métodos se definen igual que las funciones, haciendo uso de la palabra reservada **def**, dando un nombre en **snake_case**, y definiendo los parámetros en caso de ser necesario.

La diferencia con las funciones está dada, porque las **funciones no se definen dentro de una clase** y, por ende, **no están asociadas a una clase ni a un objeto específico**.

Además, en el caso de los métodos, a diferencia de las funciones, se debe hacer uso del decorador **@staticmethod** cuando se define un método estático, y se debe incluir el parámetro **self** para métodos no estáticos.

Métodos de una clase y diferencia con funciones

Entonces…

En el siguiente cuadro, encontrarás 5 características que describen a métodos y funciones. Marca con un ticket cuando la característica se cumple y con una cruz cuando no se cumple.

Característica	Funciones	Métodos
Se definen con la palabra reservada `def`		
Su nombre por convención es en `snake_case`		
Pueden o no tener parámetros*		
Pueden o no tener retorno		
Se definen dentro de una clase		

Definir un método estático

- Un método estático es aquel que se puede llamar directamente desde la clase, sin que se requiera crear una instancia de ella para hacer uso de él.
- Para definir un método estático, se requiere hacer uso del decorador **`@staticmethod`**.

En la línea siguiente al decorador **`@staticmethod`**, se debe definir el método, de la misma forma que una fu

```
# archivo pelota.py
class Pelota():
    posiciones = [3, 0, 2, 1, 0]

    @staticmethod
    def crear_rebote():
        return [5, 0, 4, 0, 3, 0, 2, 0, 1, 0]
```

Definir un método estático

Decorador `@staticmethod`

- Un decorador corresponde a un texto que comienza por el símbolo “@” y se escribe en la línea anterior a la definición de un método o una función.

- El texto que acompaña al símbolo “@”, corresponde al nombre del decorador y lo que desencadena, es el llamado a una función, que toma como argumento la función definida a continuación del decorador.

- Para que un método sea estático, además de tener el decorador @staticmethod, dentro de su lógica no puede modificar los valores de los atributos de la clase

¿Cómo hacer el llamado a un método estático?

Hacer uso de la sintaxis de punto . directamente desde el nombre de la clase

Ejemplo:

```
# desde otro archivo, se importa la clase
from pelota import Pelota

# Se llama al método directamente desde la clase, sin haber creado un objeto o instancia de pelota
print(Pelota.crear_rebote())

# La salida del print anterior será
# [5, 0, 4, 0, 3, 0, 2, 0, 1, 0]
```

¿Cómo hacer el llamado a un método estático?

Hacer uso de la sintaxis de punto . directamente desde el nombre de la clase

Ejemplo:

```
# archivo pelota.py
class Pelota():
    posiciones = [3, 0, 2, 1, 0]

    @staticmethod
    def crear_rebote():
        posiciones = [5, 0, 4, 0, 3, 0, 2, 0, 1, 0]
        return posiciones

    @staticmethod
    def imprimir_posiciones():
        Pelota.crear_rebote()
        print(Pelota.posiciones)
```

```
# desde otro archivo, se importa la clase
from pelota import Pelota

# La salida será [5, 0, 4, 0, 3, 0, 2, 0, 1, 0]
print(Pelota.crear_rebote())

# La salida será [3, 0, 2, 1, 0]
Pelota.imprimir_posiciones()
```

Ejercicio guiado

"Uso de la clase del Medicamento"

Uso de la clase del Medicamento

Desde la cadena farmacéutica donde se está desarrollando el software que permite manejar el stock de medicamentos, te informan que se requiere agregar una nueva funcionalidad al programa, y esta consiste en que cada medicamento creado no puede tener un precio igual o menor a cero.

- Para ello, agregue dentro de la clase Medicamento un método estático que permita hacer la validación de que un argumento ingresado sea mayor a 0 o no, devolviendo True de cumplir con esta condición, y False en caso contrario.
- Te solicitan además crear un script que al ejecutarse solicite al usuario ingresar un precio, y mediante el método solicitado se informe en pantalla si es o no un precio válido. El script debe además, validar que todos los medicamentos tengan el mismo IVA y descuento. Para ello, crea dos instancias de la clase, y corrobore que ambas instancias tienen los mismos valores en estos atributos. De ser así, muestre en pantalla los valores de descuento e IVA, haciendo referencia directamente a la clase.

Uso de la clase del Medicamento

Solución

Paso 1

Definir clase Medicamento según el ejercicio guiado de la sesión anterior, en el archivo **`medicamento.py`**

```python
# archivo medicamento.py
class Medicamento():
    descuento = 0.05
    IVA = 0.18
```

Paso 2

Definir método estático validar_mayor_a_cero.

```python
# archivo medicamento.py
class Medicamento():
    descuento = 0.05
    IVA = 0.18

    @staticmethod
    def validar_mayor_a_cero(numero: int):
```

Uso de la clase del Medicamento

Solución

Paso 3

Implementar lógica solicitada dentro del método **`validar_mayor_a_cero`**.

```python
# archivo medicamento.py
class Medicamento():
    descuento = 0.05
    IVA = 0.18

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

Paso 4

En un archivo llamado **programa.py** realiza el **import** de la clase Medicamento.

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

Uso de la clase del Medicamento

Solución

Paso 5

Solicitar al usuario un precio a validar y almacenar el valor en una variable.

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

precio = int(input("Ingrese un precio a validar:\n"))
```

Paso 6

Llamar al método para validar precios y almacenar retorno en una variable.

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

precio = int(input("Ingrese un precio a validar:\n"))

es_valido = Medicamento.validar_mayor_a_cero(precio)
```

Uso de la clase del Medicamento

Solución

Paso 7

Mostrar en pantalla si el precio es válido o no.

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

precio = int(input("Ingrese un precio a validar:\n"))

es_valido = Medicamento.validar_mayor_a_cero(precio)

if es_valido:
    print("El precio ingresado es válido")
else:
    print("El precio ingresado no es válido")
```

Uso de la clase del Medicamento

Solución

Paso 8

A continuación del código anterior, crear dos instancias de la clase.

```
m1= Medicamento()
m2 = Medicamento()
```

Paso 9

Hacer validación de que ambos atributos son iguales en ambas instancias.

```
m1 = Medicamento()
m2 = Medicamento()

if m1.IVA == m2.IVA and m1.descuento == m2.descuento:
```

Uso de la clase del Medicamento

Solución

Paso 10

Dentro de la condición anterior, agregar la salida por pantalla de los valores de estos atributos.

```
m1 = Medicamento()
m2 = Medicamento()

if m1.IVA == m2.IVA and m1.descuento == m2.descuento:
    print("Todas las instancias tienen igual descuento e IVA")
    print("El valor del IVA es: ", Medicamento.IVA)
    print("El valor del descuento es:", Medicamento.descuento)
```

Uso de la clase del Medicamento

Solución

La salida al ejecutar el script programa.py, e ingresando un precio de 1000, es la siguiente:

```
Ingrese un precio a validar:
1000
El precio ingresado es válido
Todas las instancias tienen igual descuento e IVA
El valor del IVA es:  0.18
El valor del descuento es: 0.05
```

/* Objetos */

Objetos

Recordando..

En Programación Orientada a Objetos, un **objeto** tiene características y acciones que realiza. Las características se denominan **atributos**, los cuales se definen dentro de la clase, y las acciones que realiza el objeto se definen en los **métodos**.

Los métodos pueden o no alterar el estado de un objeto, es decir, modificar los valores de los atributos en una **instancia** específica de la clase, en un momento dado. En general, la capacidad que tiene el objeto de realizar una acción determinada es lo que se conoce como **comportamiento**.

Comportamiento de un objeto

- **Acciones** que pueden ocurrir dentro de un objeto; estas acciones son comunes a todos los objetos de un mismo tipo, por lo que se definen dentro de una clase como métodos.
- Puede o no modificar el estado del objeto, pero siempre estará relacionado con acciones características del objeto, por ello, se definen en el “molde” de un objeto que corresponde a la clase.

Clase: Pelota

★ **Atributos:**
- `posiciones`: Lista de números

★ **Método estático:**
- `crear_rebote()`

★ **Método no estático:**
- `rebotar()`

Comportamientos de la Pelota:

- `crear_rebote`: Los rebotes se asocian a todas las pelotas
- `rebotar`: Todas las pelotas pueden rebotar

Diferencia entre método y comportamiento

- **Método**: corresponden a los bloques de código definidos dentro de una clase que permiten realizar un comportamiento una vez que son llamados.
- **Comportamiento**: es la acción en sí que realiza el objeto, mientras que el método es el código que define dicha acción. Se puede decir además que un comportamiento corresponde también cuando se realiza uno o más llamados de métodos, dando así un resultado.

Clase: Pelota

★ **Métodos:**

```
def crear_rebote():
    return [2, 0, 1, 0]

def rebotar(self):
    self.posiciones = self.crear_rebote()
```

★ **Comportamiento:**

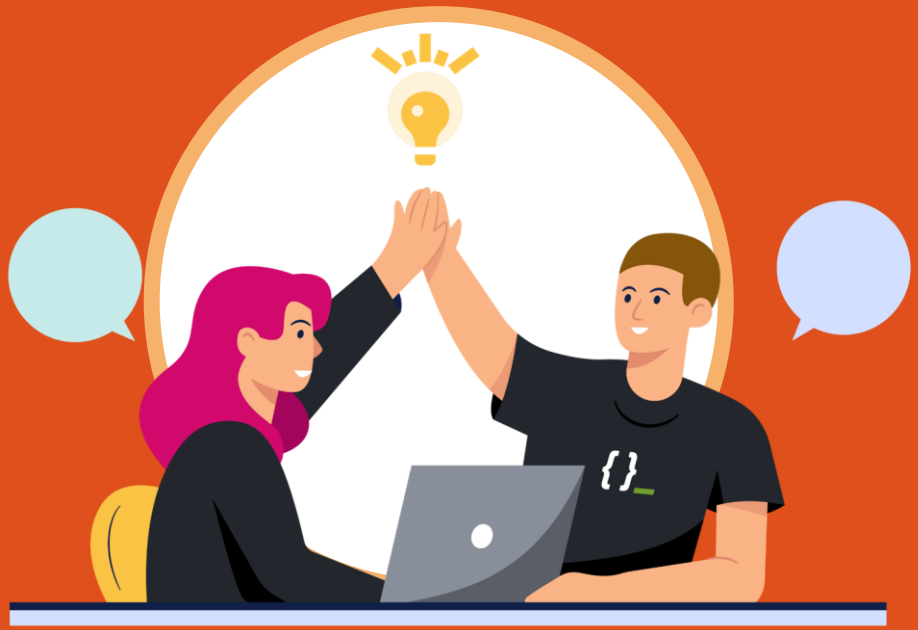
- Crea rebotes
- Rebota en posiciones

Diferencia entre método y comportamiento

En el siguiente cuadro encontrarás 3 características que describen a método y comportamiento, marca con un ticket cuando la característica se cumple y con una cruz cuando no se cumple.

Característica	Método	Comportamiento
Se relaciona con las acciones que realizan todos los objetos de una clase específica		
Ejecución de la acción definida para un objeto		
Define una acción que realiza un objeto		

El comportamiento del objeto, mediante métodos no estáticos, permite también modificar los valores de los atributos de una instancia, es decir, modificar el estado de un objeto.



¿Qué diferencia hay entre método y función?

¿Qué convención de escritura se usa para dar nombre a los métodos en Python?

Próxima sesión...

- *Desafío guiado.*

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales