



PYTHON

Fundamentos de Python 2
Programación orientada a objetos

PROGRAMACIÓN ORIENTADA A OBJETOS

FUNDAMENTOS

PROGRAMACIÓN ORIENTADA A OBJETOS

- Clase
- Objeto o instancia
- Atributos (se pueden agregar o eliminar en cualquier momento con la instrucción **del**)
- Métodos
- Constructores
- Relaciones
 - Herencia – Superclase - Subclase

PROGRAMACIÓN ORIENTADA A OBJETOS

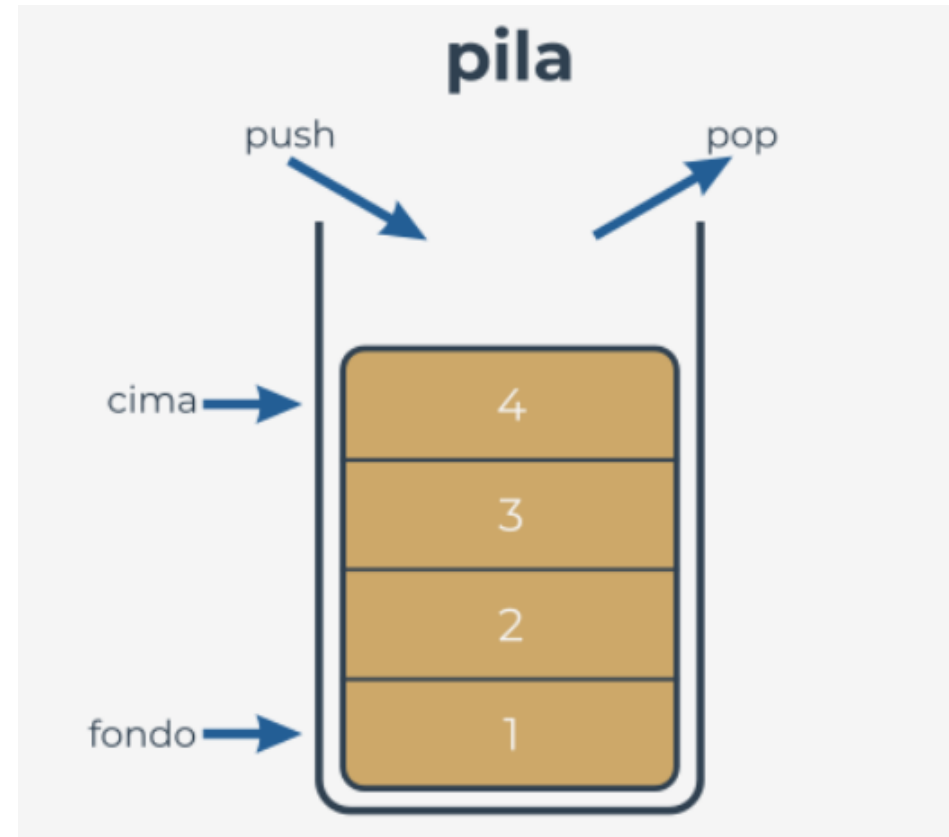
- Instanciación
- Notación punteada (*self.elemento*).
 - La referencia *self*
 - Acceso a atributos
 - Invocación de métodos
- Uso de referencias entre objetos.

PROGRAMACIÓN ORIENTADA A OBJETOS

- Encapsulación:
 - Por defecto, todo público.
 - Precedido de __, privado (con “trampa”) → __atributo es visible como __Clase__atributo.

PROGRAMACIÓN ORIENTADA A OBJETOS

- Ejercicio: implementar una pila de números enteros en **procedimental vs OO**
- **LIFO vs FIFO**



PROGRAMACIÓN ORIENTADA A OBJETOS

- Herencia:

- `class Subclase(Superclase)`
- Invocación al constructor de la superclase: `Superclase.__init__(self)`
 - Alternativamente utilizar **`super().__init()`**. En este caso no se pasa la referencia a `self`.

- Ejercicio: mediante herencia, crear una clase que herede de la pila anterior y que sepa calcular el acumulado de los elementos que contiene.

- Sobrecribir los métodos `push` y `pop` para que calculen el acumulado, invocando a los métodos existentes de la superclase.

PROGRAMACIÓN ORIENTADA A OBJETOS

- Variables de instancia:
 - Son propias del objeto.
 - Se pueden crear o eliminar en cualquier momento: PELIGRO.
 - También desde fuera de la clase → la nomenclatura de atributo privado no tiene efecto.
 - Método `__dict__` proporciona el diccionario con TODOS los atributos y valores de un objeto → Permite verificar la existencia de atributos.

PROGRAMACIÓN ORIENTADA A OBJETOS

- Variables o atributos de clase:
 - Tiene un valor compartido.
 - Se declaran fuera del constructor: *nombre_atributo=valor*
 - Pueden ser privadas
 - Se accede con *NombreClase.nombre_atributo* o *nombre_objeto.nombre_atributo*.
 - **No se muestran en `__dict__` sobre el objeto; sí se muestran en `dir()`**
 - **Sí se muestran en `__dict__` sobre la clase.**

PROGRAMACIÓN ORIENTADA A OBJETOS

- Comprobación de existencia de atributo:
 - `AttributeError`.
 - Soluciones:
 - Try-except
 - Función **hasattr**(*nombre_objeto*, "*nombre_atributo*")
 - La función **hasattr** aplicada a una clase permite preguntar por atributos de clase y de instancia.
 - La función **hasattr** aplicada a una clase permite preguntar por atributos de clase, pero no de instancia.

PROGRAMACIÓN ORIENTADA A OBJETOS

- Atributos internos:

- `__dict__`
- `__name__`. Contiene el nombre de la clase. Sólo es aplicable a la clase → Para objetos utilizar **`type()`**.
- `__module__`. Indica en el módulo en el que está la clase. Se puede utilizar sobre la instancia y sobre la clase.
- `__bases__`. Sobre una clase, proporciona una tupla con las clases **directas** de las que hereda (atención a la herencia múltiple).
 - Por defecto, todas las clases heredan de la clase **`object`**.

PROGRAMACIÓN ORIENTADA A OBJETOS

- **Introspección.** Capacidad de un objeto de analizar en tiempo de ejecución su composición.
- **Reflexión.** Capacidad de un objeto de manipular en tiempo de ejecución su composición.
- Atributo `__dict__` permite acceder a todos los atributos de un objeto: es un diccionario con los métodos `keys()`, `values()` e `items()`.
- Funciones *built-in* **`getattr(objeto, atributo)`** y **`setattr(objeto, atributo, valor)`** permiten leer y modificar atributos

PROGRAMACIÓN ORIENTADA A OBJETOS

- Herencia.
 - Se indica entre paréntesis junto al nombre de la subclase: Subclase(Superclase).
 - Función *built-in* **issubclass**.
 - Función *built-in* **isinstance**.
 - Método `__str__`
 - Operador **is** vs **==**. Atención al comportamiento de **is** con cadenas nuevas vs cadenas modificadas.
 - Invocaciones a constructores o métodos de las superclases a través de **super()**: `super().__init__()`; `super().calcular()`;...

PROGRAMACIÓN ORIENTADA A OBJETOS

- Herencia múltiple.
 - Una clase hereda de varias.
 - Notación: Subclase(Superclase1, Superclase2,...)
 - Búsqueda: de abajo a arriba y de izquierda a derecha. Resolución de problemas de ambigüedad predecible.
 - Hay que intentar evitarla: viola el principio de responsabilidad única.
 - MRO (Orden de Resolución de Métodos).
 - En herencia múltiple, se genera error si una clase C, hereda de A y B (en ese orden, en el contrario funciona bien), y B a su vez hereda de A. Provoca TypeError.

PROGRAMACIÓN ORIENTADA A OBJETOS

- Polimorfismo.
 - Un método tiene diferente comportamiento en las distintas clases de la jerarquía.
 - Al método redefinido se le llama **virtual**.
- ¿Método abstracto?:
 - Método sin implementar (contiene pass).

PROGRAMACIÓN ORIENTADA A OBJETOS

■ Composición.

- Una clase está compuesta por instancias de otras clases, que le proporcionan la funcionalidad.

```
import time

class Tracks:
    def change_direction(self, left, on):
        print("pistas: ", left, on)
class Wheels:
    def change_direction(self, left, on):
        print("ruedas: ", left, on)
class Vehicle:
    def __init__(self, controller):
        self.controller = controller
    def turn(self, left):
        self.controller.change_direction(left, True)
        time.sleep(0.25)
        self.controller.change_direction(left, False)

wheeled = Vehicle(Wheels())
tracked = Vehicle(Tracks())

wheeled.turn(True)
tracked.turn(False)
```

Fuente del ejemplo: Edube Interactive

PROGRAMACIÓN ORIENTADA A OBJETOS

- Las excepciones son clases:
 - Método `__subclasses__()`
 - Atributo `args` de la excepciones → Todos los argumentos pasados al constructor

```
def print_exception_tree(thisclass, nest = 0):  
    if nest > 1:  
        print("  |" * (nest - 1), end="")  
    if nest > 0:  
        print(" +---", end="")  
  
    print(thisclass.__name__)  
  
    for subclass in thisclass.__subclasses__():  
        print_exception_tree(subclass, nest + 1)
```

Fuente del ejemplo: Edube Interactive

```
print_exception_tree(BaseException)
```

PROGRAMACIÓN ORIENTADA A OBJETOS

- Construcción de excepciones propias.
 - Heredan de excepciones existentes.
 - Invocando a sus constructores con `super().__init__`
 - Añadiendo nuevos atributos y métodos