eip ESCUELA INTERNACIONAL POSGRADOS

Máster Avanzado de Programación en Python para Hacking, BigData y Machine Learning

CERTIFICACIÓN PCAP

LECCIÓN 5

Programación Orientada a Objetos (POO)

ÍNDICE

INTRODUCCIÓN

En esta lección vamos a continuar con el repaso del contenido del examen PCAP centrándonos en el estudio de la programación orientada a objetos y sus principales características.

OBJETIVOS

Al finalizar esta lección serás capaz de:

- Conocer los conceptos clave de los objetos, clases y herencia.
- Conocer cómo se implementan estos conceptos en Python.
- Conocer la diferencia entre variables de instancia y las variables de clase.
- Conocer cómo funciona la herencia en Python y los posibles conflictos que se pueden dar.

Marco teórico

Clase

Conjunto de elementos con caracteres comunes.
Generalización abstracta.

Objeto

Entidades concretas. Instancias de las clases.
Un objeto pertenece a una clase.

Subclase

Subfamilias o subgrupos que se pueden formar dentro de una clase existente.

Marco teórico

Clase padre o superclase

Conjunto de elementos con caracteres comunes.

Generalización abstracta.

Herencia

Proceso mediante el cuál una subclase hereda de una clase padre, compartiendo sus métodos y atributos.

Subclase/Superclase

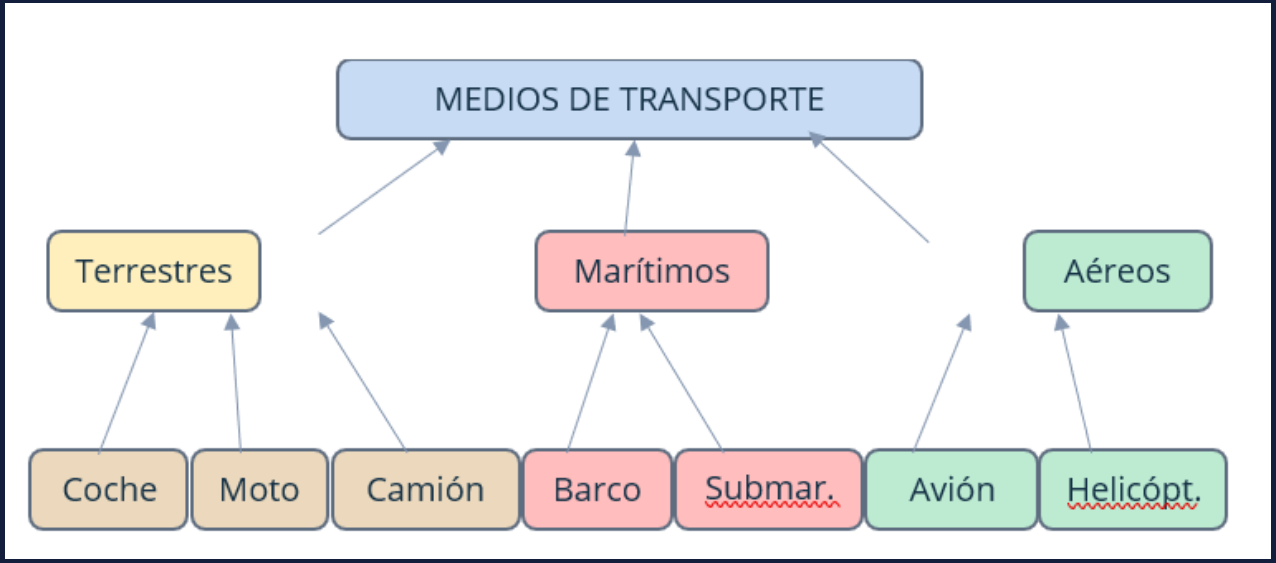
Cada subclase es más especializada que su superclase.

Cada superclase es más general (más abstracta) que sus subclases.

Marco teórico

Herencia

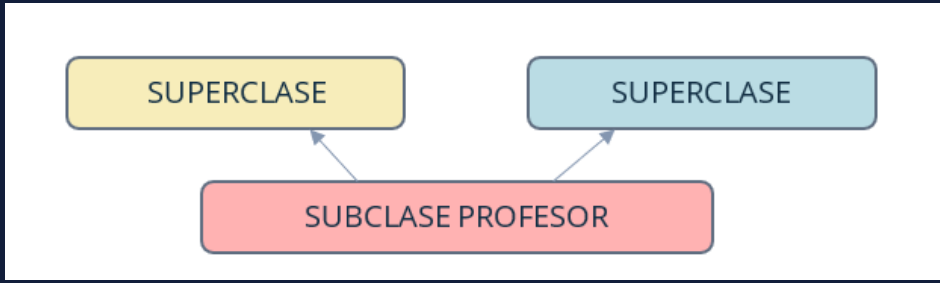
Un objeto hereda todos los rasgos definidos dentro de cualquiera de las superclases.



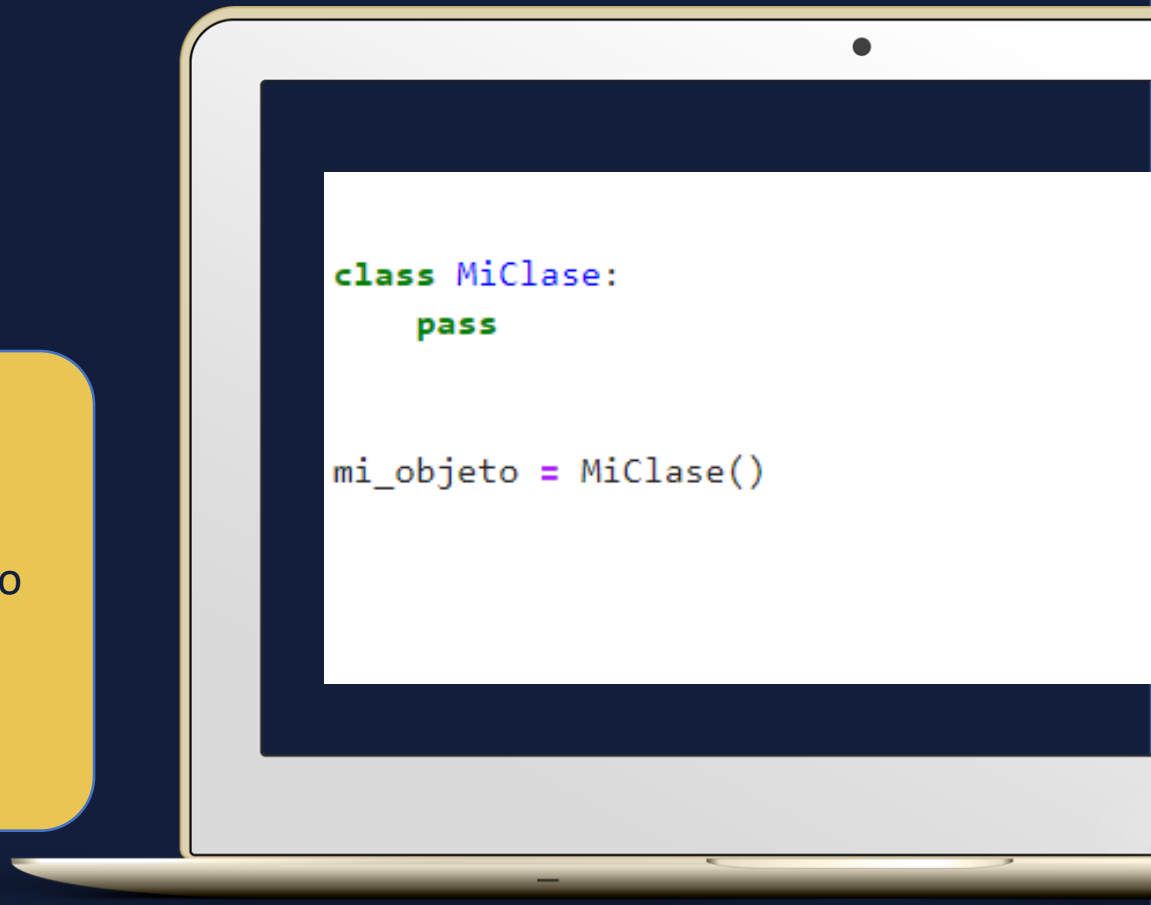
Marco teórico

Herencia múltiple

Un objeto puede tener varias superclases.



Definir una clase



Importante

- **Instanciación**: Procedimiento mediante el cual se crea un objeto utilizando una clase

Definir una clase

Características/atributos de las que podemos dotar a las clases y sus instancias/objetos:

Nombre

Los objetos tienen un nombre que los identifica.

"sustantivo"

Atributos

Los objetos tienen un conjunto de propiedades individuales que los identifican.

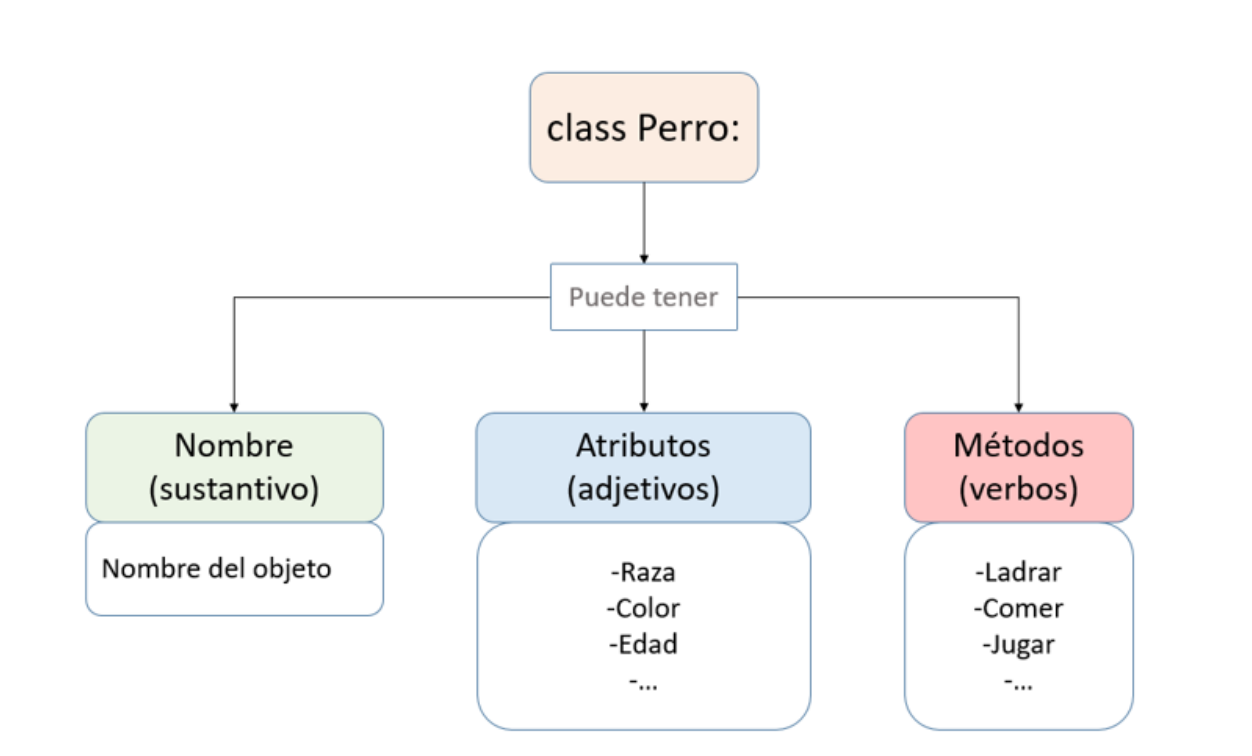
"adjetivos"

Métodos

Los objetos pueden tener una serie de habilidades con las que pueden alterarse a sí mismos o a otros objetos.

"verbos"

Definir una clase



Definiendo atributos en una clase: Inicializando objetos

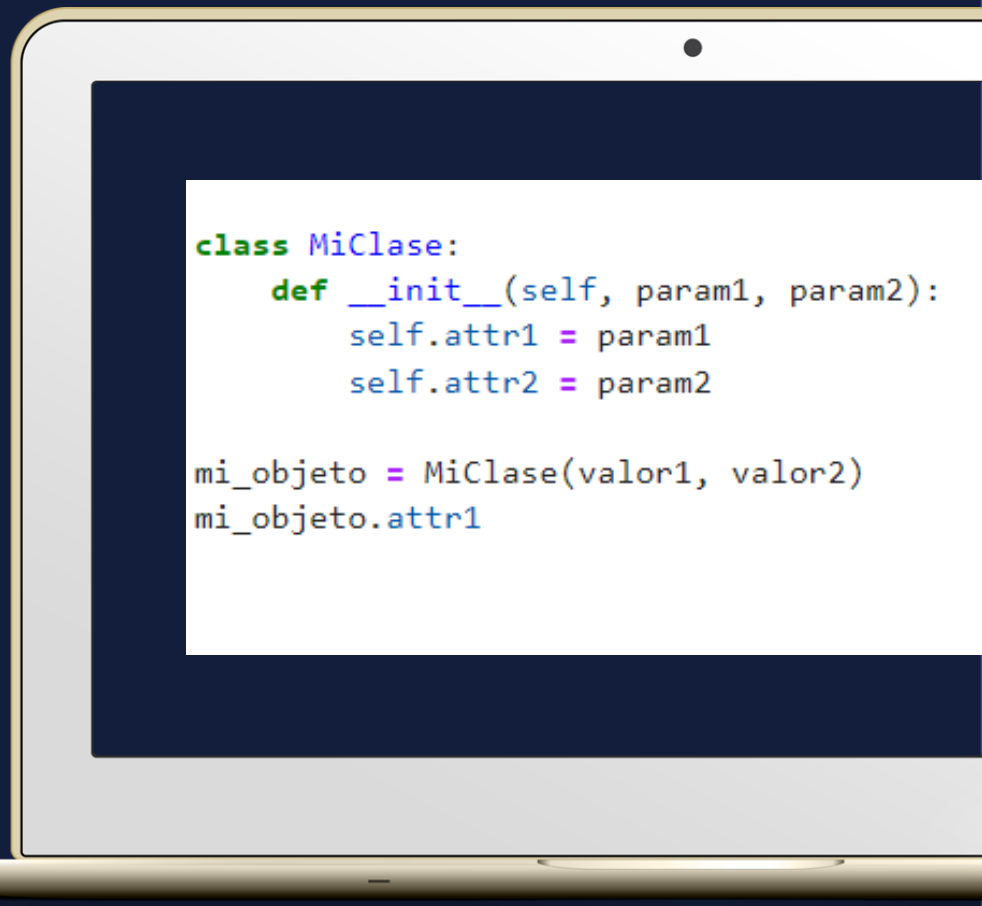
Constructor: Función que permite "inicializar" los objetos.

Importante

- Siempre se llama __init__
- Si no se define utilizará un constructor vacío.

- Debe tener siempre un parámetro: self.
- No debe devolver nada: TypeError

- El constructor se ejecuta "automáticamente" al crear cada objeto.

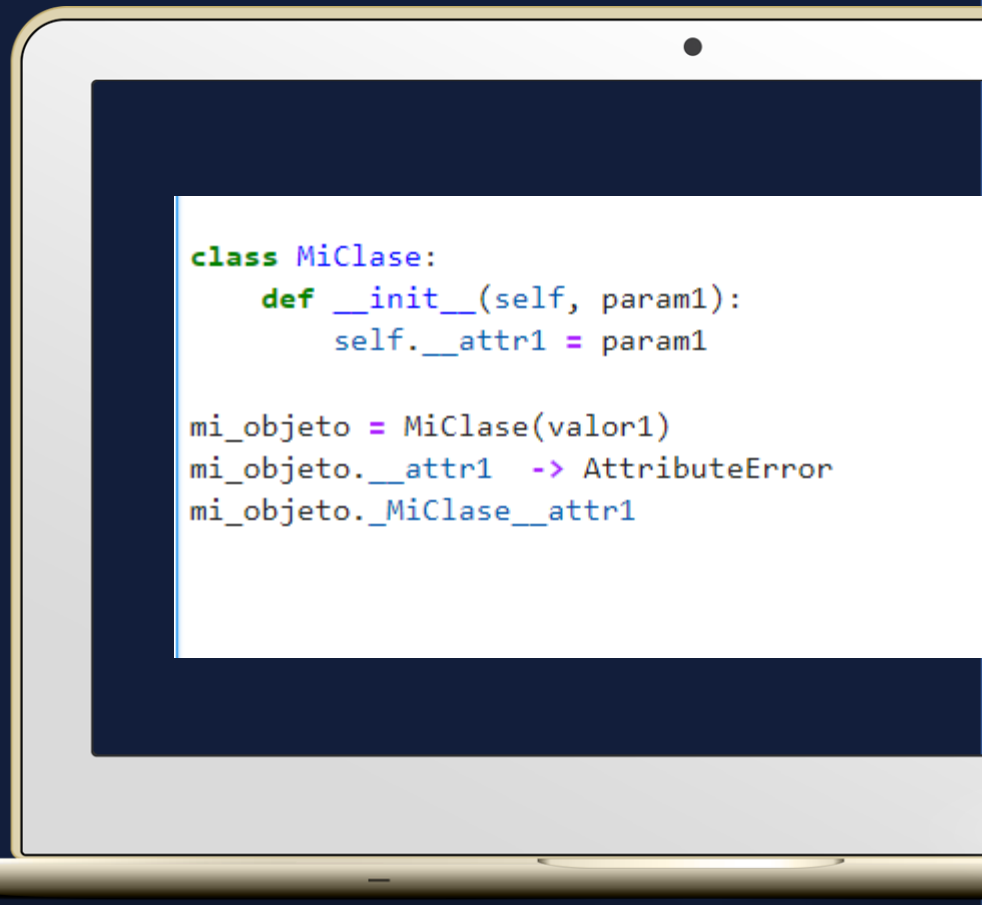


```
class MiClase:
    def __init__(self, param1, param2):
        self.attr1 = param1
        self.attr2 = param2

mi_objeto = MiClase(valor1, valor2)
mi_objeto.attr1
```

Definiendo atributos en una clase: Encapsulación

Importante

- Cualquier elemento definido dentro de la clase precedido de __ solo será accesible desde dentro de la clase.
- Si se intenta acceder AttributeError
- Hay una forma de acceder a ellos ya que Python sólo los renombra.

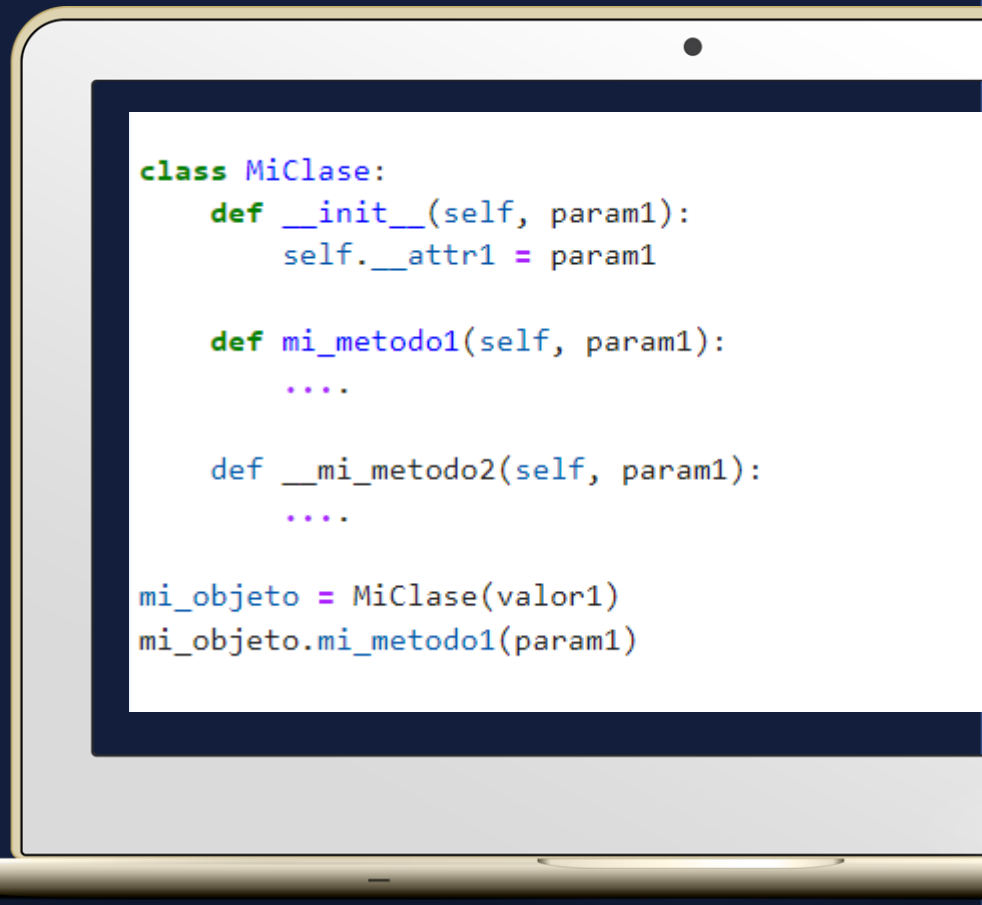


```
class MiClase:
    def __init__(self, param1):
        self.__attr1 = param1

mi_objeto = MiClase(valor1)
mi_objeto.__attr1   -> AttributeError
mi_objeto._MiClase__attr1
```

Definiendo atributos en una clase: Definiendo métodos

Importante

- Deben tener un parámetro self.
- Podemos utilizar un return para devolver un valor.
- Se pueden declarar métodos privados.



```
class MiClase:
    def __init__(self, param1):
        self.__attr1 = param1

    def mi_metodo1(self, param1):
        ....

    def __mi_metodo2(self, param1):
        ....

mi_objeto = MiClase(valor1)
mi_objeto.mi_metodo1(param1)
```

Conocer los atributos: __dict__ y hasattr()

__dict__

- Atributo
- Devuelve un diccionario con los métodos y atributos del objeto o de la clase.

```
objeto.__dict__
MiClase.__dict__
```

hasattr()

- Función
- Permite comprobar si el objeto o clase tiene un atributo.
- Devuelve True o False

```
hasattr(objeto, "nombreatributo")
hasattr(MiClase, "nombreatributo")
```

Variables de instancia vs variables de clase: Variables de instancia

Creación de variables de instancia:

- En el constructor: Todos los objetos las tendrán.
- En un método de la clase: Sólo si el objeto llama al método la tendrá.
- Exteriormente: Sólo el objeto que la cree la tendrá.

Importante

- **Variable de instancia**: Al alterarla surtirá efecto dentro del objeto al que pertenecen.
- Diferentes objetos de la misma clase pueden tener diferentes propiedades.
- Si hacemos un __dict__ :
  - En el objeto aparecerá.
  - En la clase no aparecerá.

Variables de instancia vs variables de clase: Variables de clase

Creación de variables de clase:

- Antes del constructor

No se muestran con __dict__ del objeto, pero si con el de la clase.

Importante

- **Variable de clase**: Esta disponible para todos los objetos.
- Para todos los objetos tendrá el mismo valor. Si se modifica en uno se modificará en el resto.
- Si hacemos un __dict__ :

  En el objeto no aparecerá.

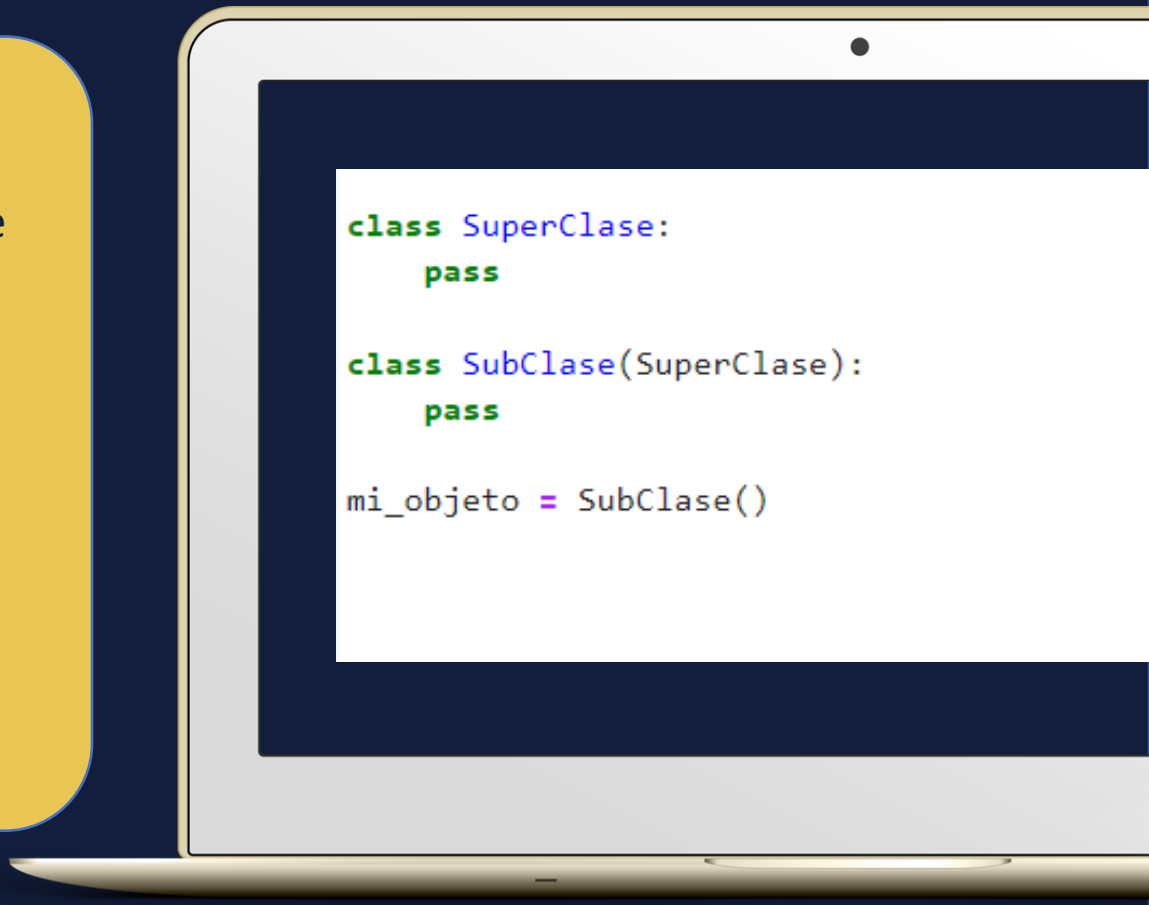
  En la clase aparecerá.

Herencia: Herencia única

Importante

Se puede usar en los objetos de la subclase:

- Variables de clase
- Métodos



```
class SuperClase:
    pass

class SubClase(SuperClase):
    pass

mi_objeto = SubClase()
```

Herencia: Herencia única

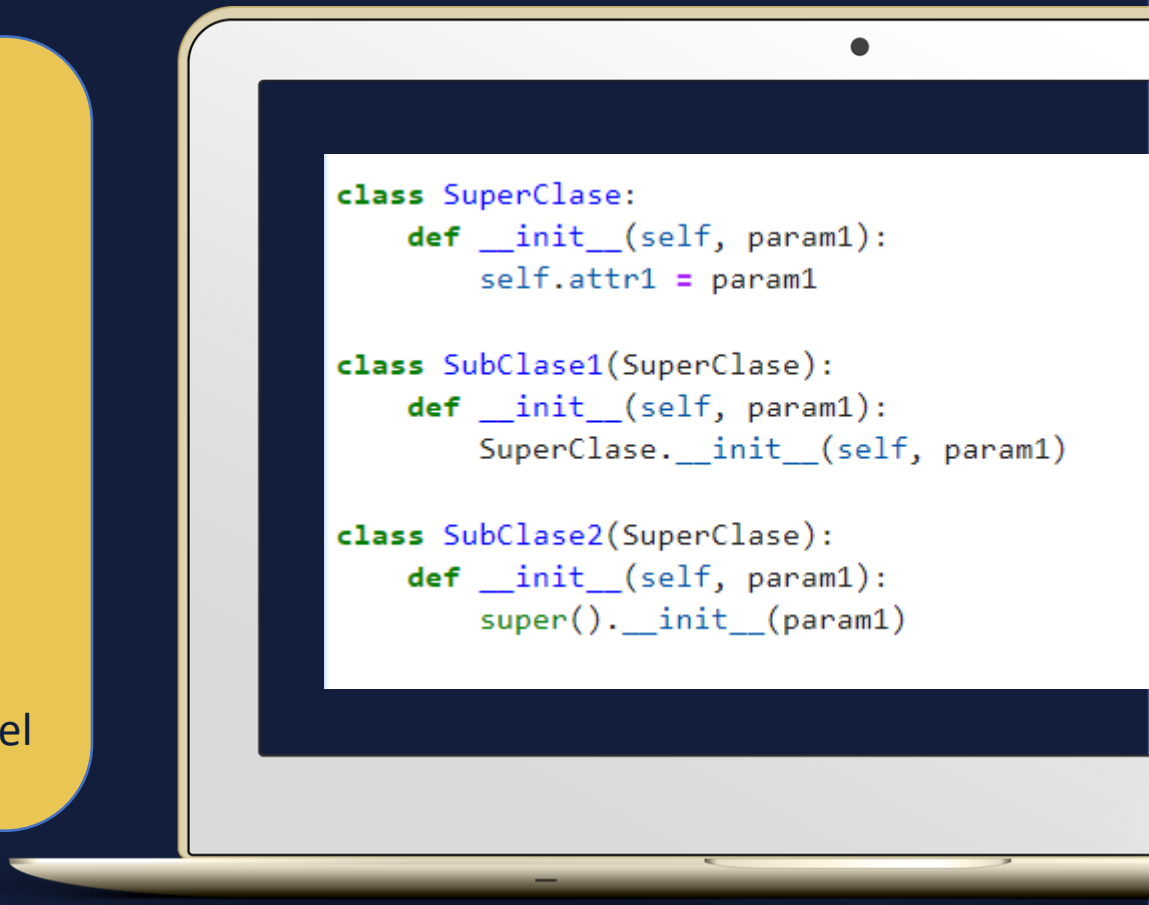
Importante

Para utilizar las variables de instancia hay que llamar al constructor de la SuperClase:

- super().__init__()
- SuperClase.__init__(self)

Ver la diferencia en el uso del self.

Recomendable ponerlo antes del resto de las instrucciones.



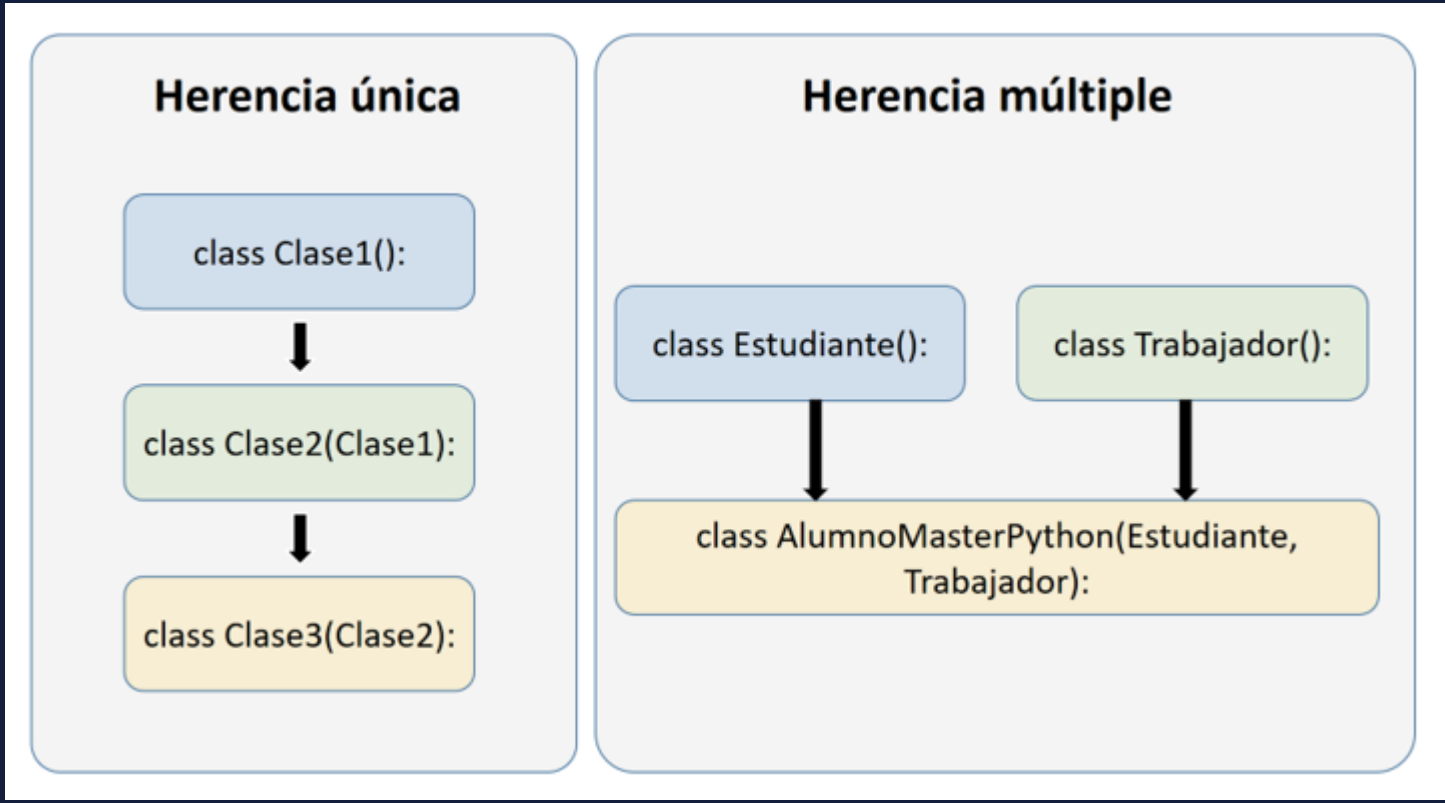
```python
class SuperClase:
    def __init__(self, param1):
        self.attr1 = param1

class SubClase1(SuperClase):
    def __init__(self, param1):
        SuperClase.__init__(self, param1)

class SubClase2(SuperClase):
    def __init__(self, param1):
        super().__init__(param1)
```

Herencia: Herencia única - Conflictos

¿Qué pasa si la SubClase define una variable de instancia, una variable de clase o un método con el mismo nombre que tienen en la SuperClase?

- En el objeto de la subclase se utilizarán los definidos en la subclase.
- En el caso de variables de clase si se modifican en la subclase no se aplicarán los cambios en la superclase.
- Funcionamiento: Python siempre busca primero en la clase del objeto y si no lo encuentra, pasa a su superclase y si no lo encuentra a su superclase y así sucesivamente. Si no lo encuentra en ninguna -> “AttributeError”.

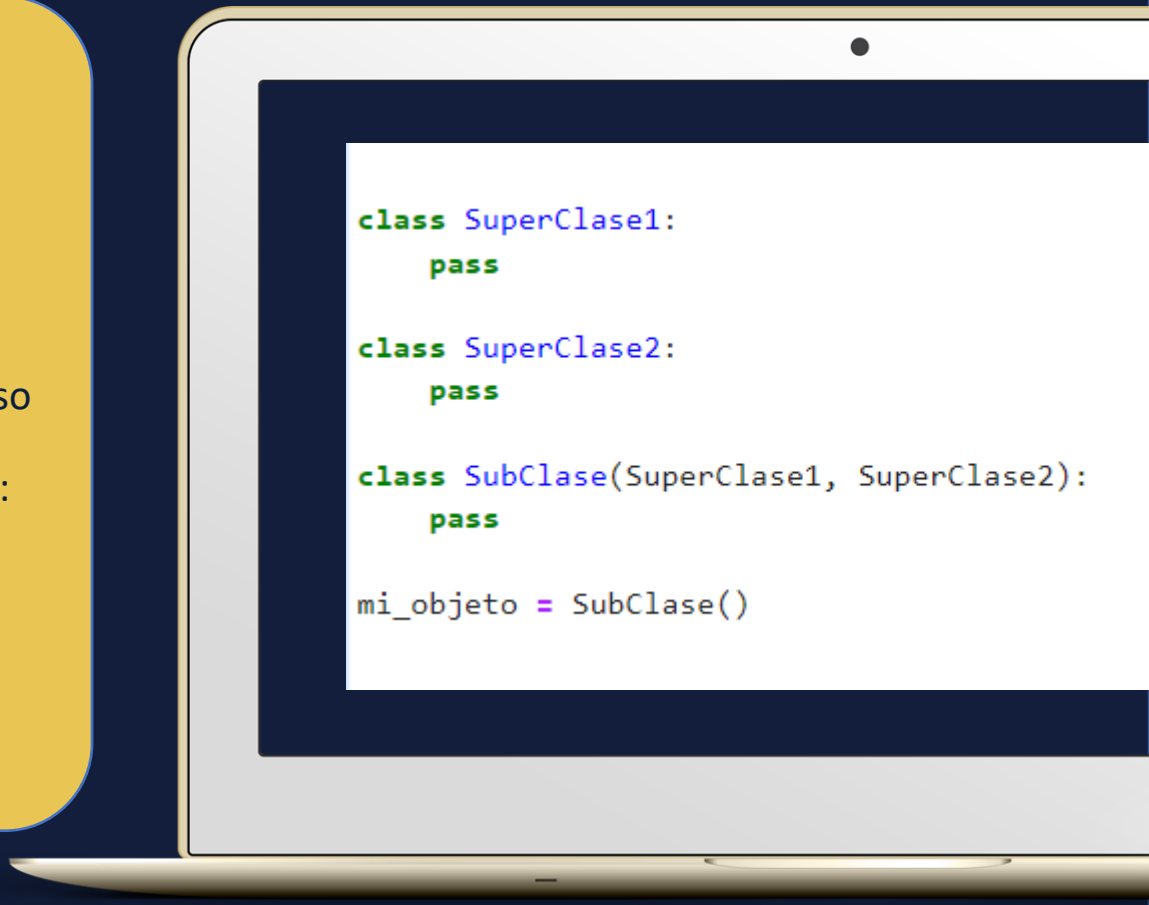
Herencia: única vs múltiple



Herencia: Herencia múltiple

Importante

- La subclase tiene directamente acceso a las variables de clase y métodos de las superclases.
- Para que la subclase tenga acceso a las variables de instancia debe llamar a todos las constructores: SuperClase.__init__(self,…)
- Método issubclass(Subclase, Superclase) -> booleano



```
class SuperClase1:
    pass

class SuperClase2:
    pass

class SubClase(SuperClase1, SuperClase2):
    pass

mi_objeto = SubClase()
```

Herencia: Herencia múltiple - Conflictos

¿Qué pasa si la SubClase define una variable de instancia, una variable de clase o un método con el mismo nombre que tienen en la SuperClase? -> Lo mismo que en la herencia única.

¿Qué pasa si las Superclases definen una variable de instancia, una variable de clase o un método con el mismo nombre? -> La Subclase utilizará los de la clase definida más a la izquierda.

Python a la hora de resolver el valor de las variables de una subclase busca:

- De **abajo hacia arriba en la línea de herencia**
- De **izquierda a derecha cuando hay varias subclases al mismo nivel**.

Es recomendable intentar evitar la utilización de herencia múltiple.

CONCLUSIONES

- Hemos vistos los conceptos de la POO y cómo se implementan en Python.
- Hemos visto la diferencia entre variables de instancia y variables de clase.
- Hemos visto la manera en la que la herencia se define en Python: Herencia única y múltiple. Y los principales conflictos que se pueden dar.

MUCHAS GRACIAS POR SU ATENCIÓN

eip ESCUELA INTERNACIONAL POSGRADOS

tcivera@grupomainjobs.com

Tamara Civera Lorenzo
es.linkedin.com/in/tamara-civera-lorenzo-95962147

twitter.com/eiposgrados

facebook.com/eiposgrados

instagram.com/eiposgrados