# Resumen para 1° Parcial

## Unidad 2: POO, Clases y Objetos

### Métodos en Python

**@classmethod:** Los métodos de clase realizan acciones que no involucran los valores de un objeto. Utiliza **cls** cómo primer parámetro del método.

Según cómo lo entendí, los métodos convencionales (o sea métodos de instancia) trabajan con una **instancia** de la **clase** (por eso su primer parámetro es ***self*** (el propio objeto), EN CAMBIO, un método de clase opera sobre la propia **clase**, y **no** sobre una instancia específica. ***cls*** refiere a la propia clase, y permite acceder y modificar atributos o métodos de la clase.

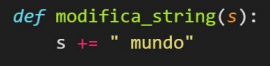
**@staticmethod:** **sin parámetro**, no accede a atributos de clase ni instancia. Utilizado para funciones de “utilidad”.

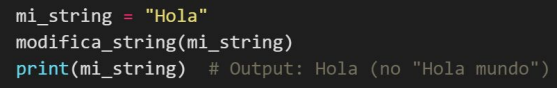
### Pass-by-object

Cuando se pasa un **objeto inmutable** cómo parámetro se crea una nueva referencia en el alcance de la función, pero cualquier modificación en este resulta en la creación de **otro objeto**.

Ejemplos de objetos inmutables: **int**, **float**, **str**, **None**, etc.

Básicamente, esto significa que si yo meto cómo parámetro, por ejemplo, un **str** en un método y lo modifico dentro del mismo, se podría crear un nuevo objeto pero **no se alteraría** el original





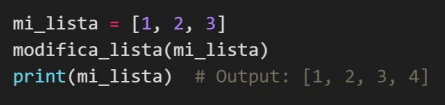
Sigue imprimiendo “Hola” porque no alteró ‘mi\_string’



En cambio, para un **objeto mutable** se pasa una referencia, por lo que el objeto original puede ser modificado.

Ejemplos de objetos mutables: Lists, Dictionaries, Sets, etc.







### Constructores

Se invoca automáticamente luego de que se crea la instancia, setea el estado inicial del objeto. **\_\_init\_\_(self)**

Existe un método que se llama antes de construir el objeto, por lo que no tiene ***self***, pero no suele ser utilizado

**\_\_new\_\_()**

En Python **no existe** la sobrecarga del constructor.

### Destructores

Para operaciones de limpieza, cómo cerrar archivos o liberar recursos

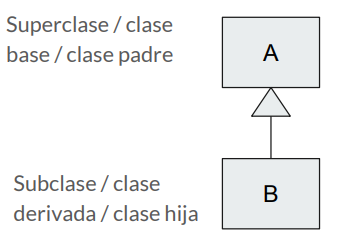
**\_\_del\_\_(self)**

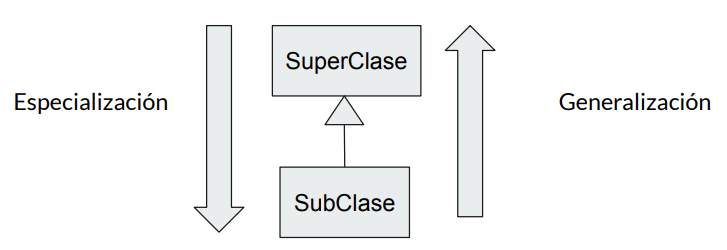
## Unidad 3: Herencia y Polimorfismo

**Reutilización:** Existen funcionalidades que ya fueron implementadas en otros sistemas y que no son necesarias de implementar nuevamente, para esto se **reutiliza** código.

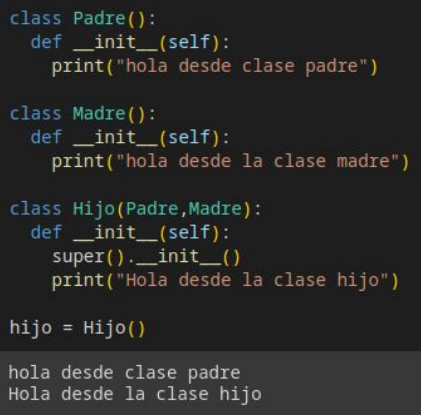
**Extensibilidad:** Permite extender el código sin afectarlo.

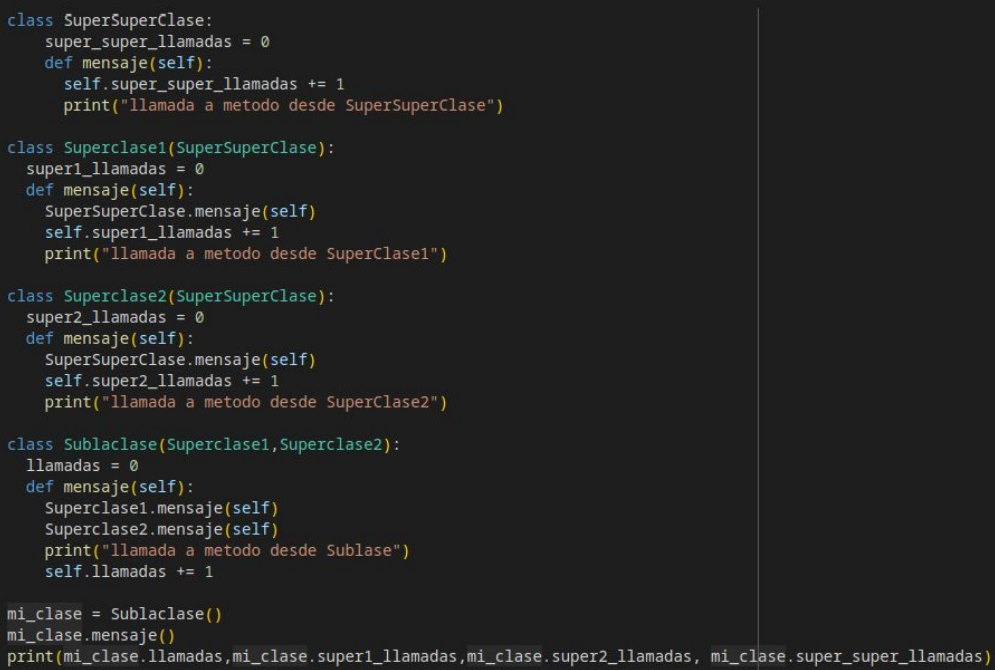
#### Herencia





### Herencia múltiple





llamada a metodo desde SuperSuperClase

llamada a metodo desde SuperClase1

llamada a metodo desde SuperSuperClase

llamada a metodo desde SuperClase2

llamada a metodo desde Subclase

1 1 1 2

### Mixin

**Clase** que proporciona métodos para ser utilizados cómo una extensión opcional para otras clases. Es heredada junto con otras clases para proporcionar comportamientos específicos.

#### Polimorfismo

Tipos de Polimorfismo:

1. **OVERLOADING, SOBRECARGA, o ESTÁTICO**

Cuando el nombre de un método tiene varias implementaciones. Se distingue en tiempo de compilación, por los **tipos** y **cantidad** de **parámetros**.

1. **OVERRIDING, SOBREESCRITURA, o DINÁMICO**

Mismo: nombre de método, cantidad y tipos de parámetros (misma “signatura”), y además con una relación de herencia entre clases.

1. **VARIABLE POLIMÓRFICA**

Se **declara** de un tipo, pero **contiene** otro diferente.

1. **GENÉRICO o TEMPLATE**

Crean herramientas de propósito general para hacerlas más específicas en ciertas situaciones.

### Duck Typing

Si el objeto sabe responder a los métodos que necesito, es suficiente.

### Interfaz

De tipo **formal** o **informal**, definen un conjunto de métodos que un objeto debe cumplir, pero **no cómo**.

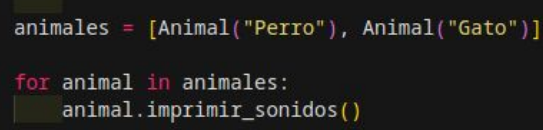
## Unidad 3: SOLID

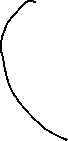
#### Single responsibility

Una clase solo debe ser responsable por una cosa. Si tiene más de una responsabilidad se acopla.

#### Open – Closed

Las entidades **SW** (Clases, módulos, funciones) deben ser abiertas para extensión y no para modificación.





‘Imprimir\_sonidos’ está cerrada para nuevos animales, si se suma otro es necesario agregar otro if. En este caso una solución puede ser hacer clases para cada animal y heredar de class Animal.



#### Liskov Substitution

Una subclase debe ser sustituible por su superclase.

Es decir, la subclase debe poder asumir el lugar del padre sin errores y sin tener que hacer comprobación de tipos.

#### Interface Segregation

Las interfaces tienen que ser específicas. Las clases que las implementan no deben verse obligadas a implementar métodos que **no necesitan**.

#### Dependency Inversion

Las dependencias deben ser sobre abstracciones, y no sobre concreciones.

Módulos de alto y bajo nivel deben depender de abstracciones.

Los detalles deben depender de las abstracciones.