

# Seminario de Lenguajes - Python

Cursada 2024

Clase 7: conceptos básicos de la POO

Pensemos en la siguiente situación:

- En el trabajo integrador tenemos que registrar los datos de quienes interactúan con la aplicación.
- Podemos pensar en una entidad **Persona\_Usuaría** con datos asociados tales como: - nombre - nick - género - edad
- También podríamos pensar en: - imagen de su avatar - último acceso - paleta de colores elegida

Con lo visto hasta el momento, ¿qué estructura de datos podemos elegir para representar a un usuario?

¿Podríamos utilizar diccionarios para definir las características?

```
In [ ]: user = {'name': 'Janis Joplin',  
               'nick': 'Janis',  
               'gender': 'femenino',  
               'age': 27,  
               'avatar': 'janis.png'}  
}
```

¿Podemos asociar funcionalidades específicas a esta persona?

Por ejemplo, **cambiar su nombre**, **registrar alguna actividad**, **actualizar su avatar**, etc.

Podríamos definir funciones para definir la funcionalidad asociada

```
In [ ]: def update_name(user, new_name):  
        """ Actualiza el nombre del usuario  
  
        user: representa a la persona usuaria con la que queremos operar.  
        new_name: un str con el nuevo nombre.  
        """  
        user['name'] = new_name
```

Pero...

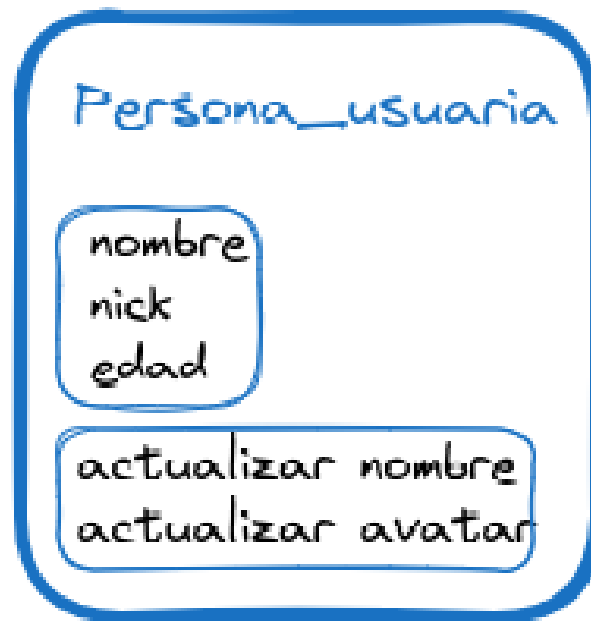
- ¿Podemos modificar a "nuestra persona usuaria" sin utilizar estas funciones?

- ¿Podemos decir que "nuestra persona usuaria" es una **entidad** que **encapsula** tanto su estructura como la funcionalidad para manipularla?

Si...y no...

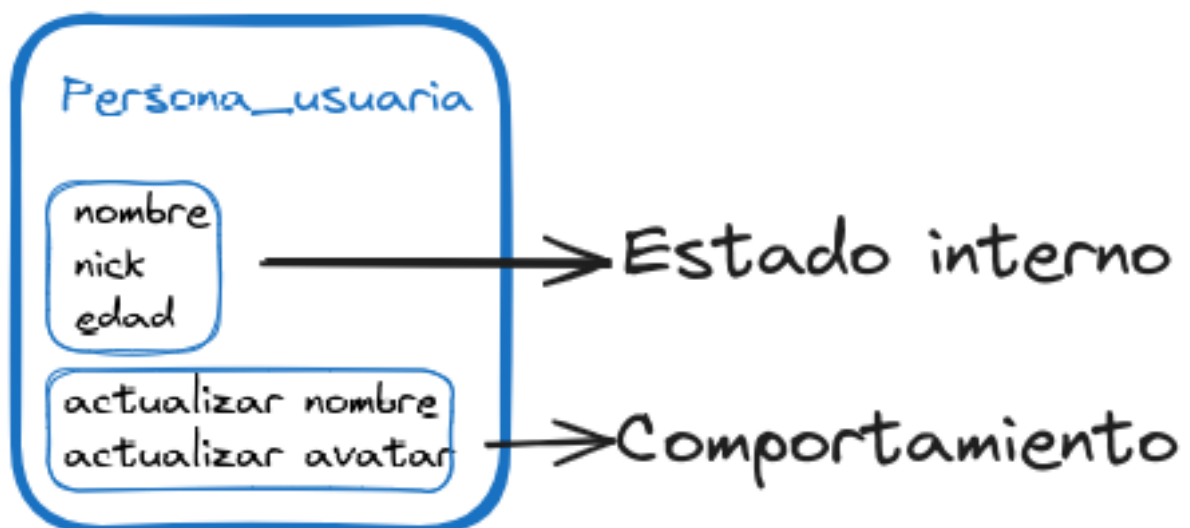
## Hablemos de objetos ...

Un **objeto** es una colección de datos con un comportamiento asociado en una única entidad.



## Objetos

- Son los elementos fundamentales de la POO.
- Son entidades que poseen un **estado interno** y un **comportamiento**.



## Objetos

- Ya vimos que en Python **todos** los elementos con los que trabajamos son objetos.

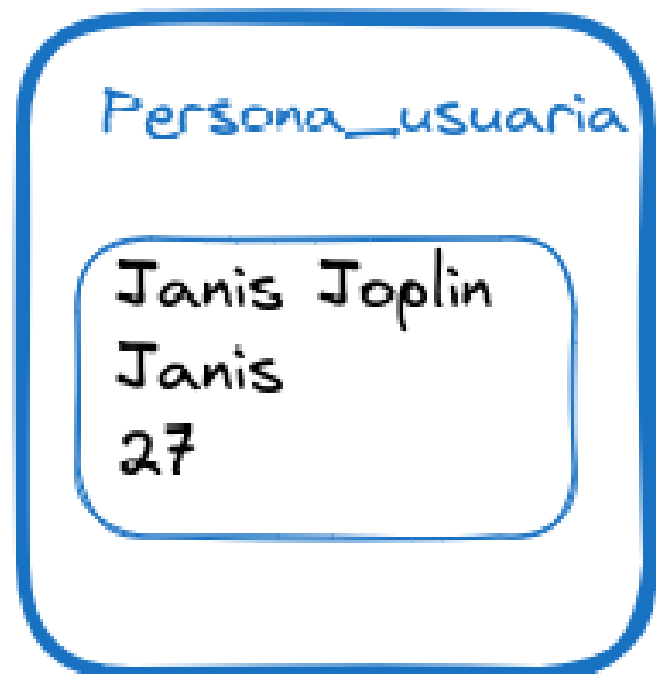
```
sentence = "Hola que tal."  
file = open("archi.txt")  
  
sentence.upper()  
file.close()
```

- **sentence** y **file** referencian a **objetos**.
- **upper** y **close** forman parte del comportamiento de estos objetos: son **métodos**.

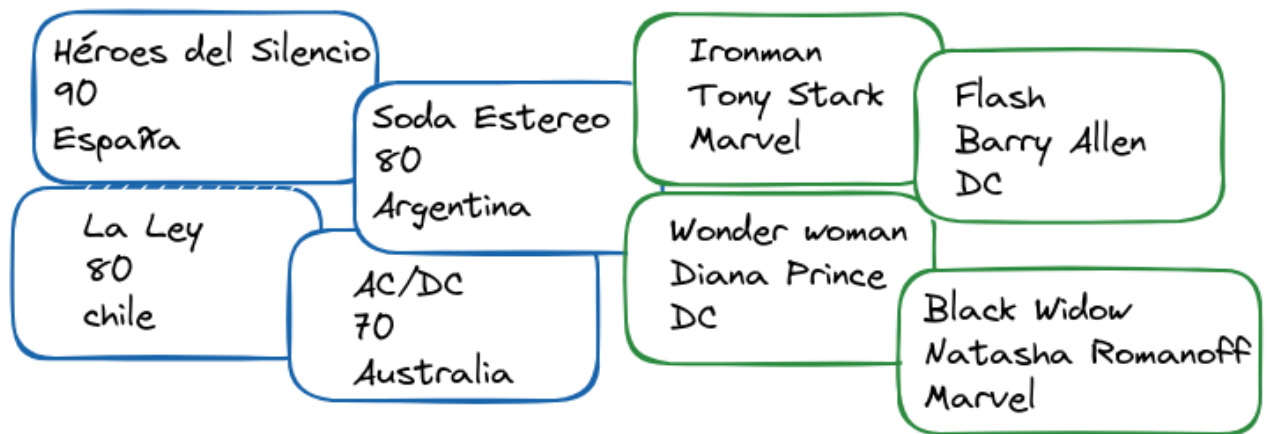
## POO: conceptos básicos

- En POO un programa puede verse como un **conjunto de objetos** que interactúan entre ellos **enviándose mensajes**.
- Estos mensajes están asociados al **comportamiento** del objeto (conjunto de **métodos**).

actualizar avatar  
→



# El mundo de los objetos

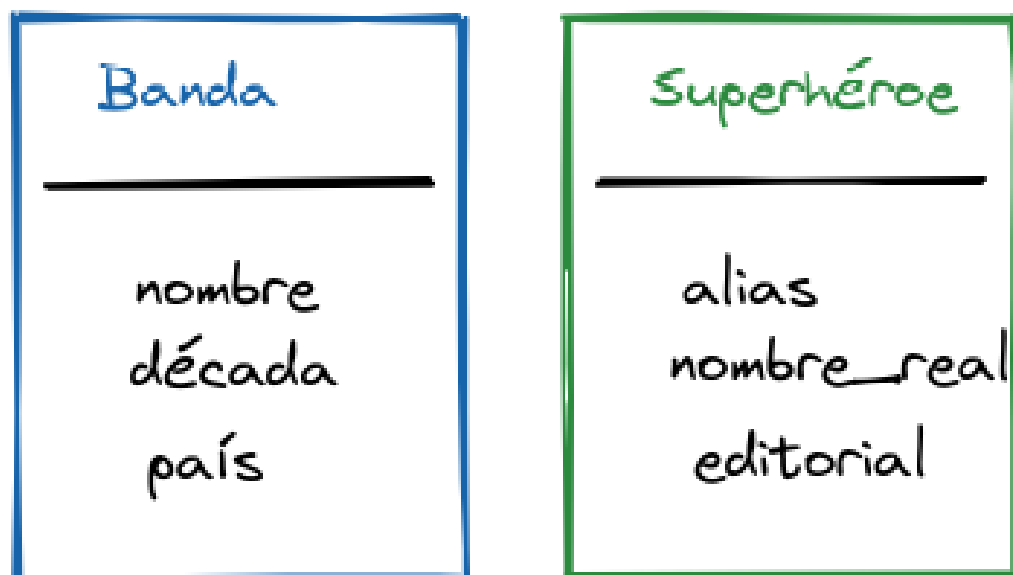


- ¿Qué representa cada objeto?
- ¿Qué podemos decir de cada grupo de objetos?

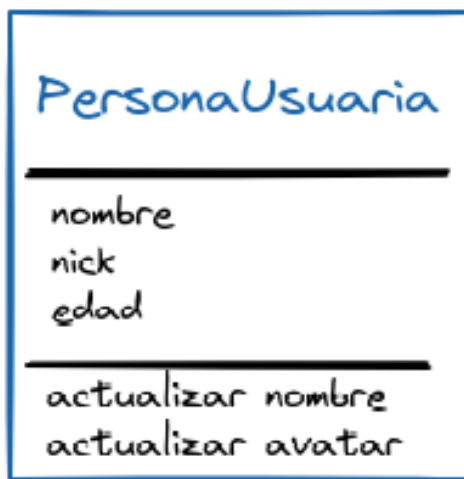
## Objetos y clases

- No todos los objetos son iguales, ni tienen el mismo comportamiento.
- Así agrupamos a los objetos de acuerdo a características comunes.

Una clase describe los atributos o características de los objetos y las acciones o métodos que pueden hacer o ejecutar dichos objetos.



Pensemos en la clase que representa a una persona usuaria



- Cuando creamos un objeto, creamos una **instancia de la clase**.
- Una **instancia** es un **objeto individualizado** por los valores que tomen sus atributos o **variables de instancia**.



- La **interfaz pública** del objeto está formada por las **variable de instancia** y **métodos** que otros objetos pueden usar para interactuar con él.
- ¿Qué pasa si todas las variables de instancia y métodos son privadas? ¿Y si son todas públicas?

## Clases en Python

```
class NombreClase:  
    sentencias  
    ...
```

- La [PEP 8](#) sugiere usar notación CamelCase en el caso del nombre de las clases.
- Al igual que las funciones, las clases **deben** estar definidas antes de que se utilicen.
- Con la definición de una nueva clase se crea un nuevo **espacio de nombres**.

¿Cómo se crea una instancia de una clase?

```
objeto = NombreClase()
```

# La clase User

```
In [ ]: class User():
        """Define la entidad que representa a una persona usuaria en PyTrivia"""

        #Estado interno
        name = 'Janis Joplin'
        nick = 'Janis'
        avatar = None

        #Comportamiento
        def update_name(self, new_name):
            self.name = new_name
```

- ¿self?
- ¿Qué quiere decir que User tiene su propio espacio de nombres?

## Creamos las instancias

```
In [ ]: janis = User()
        print(janis.name)
        janis.update_name("Janis")
        print(janis.name)
```

- Observemos la línea 3: **janis.update\_name("Janis")**
  - Prestar atención a la cantidad de parámetros pasados.
- Cuando creamos otros objetos de clase **User**, ¿qué particularidad tendrán?

```
In [ ]: other = User()
```

```
In [ ]: print(other.name)
```

## Podemos parametrizar la creación de objetos

```
In [ ]: class User():
        """Define la entidad que representa a una persona usuaria en PyTrivia"""

        def __init__(self, name, nick):
            # Estado interno de cada objeto
            self.name = name
            self.nick = nick
            self.avatar = None

        #Comportamiento
        def update_name(self, new_name):
            self.name = new_name
```

```
In [ ]: janis = User('Janis Joplin', 'Janis')
        janis.update_name("Janis")
```

- El método **init()** se invoca automáticamente al crear el objeto.

# ¿Qué pasa si..?

```
In [ ]: other = User()
```

## Podemos inicializar con valores por defecto

```
In [ ]: class User():
        """Define la entidad que representa a una persona usuaria en PyTrivia"""

        def __init__(self, name="Nueva persona", nick=None):
            self.name = name
            self.nick = nick
            self.avatar = None

        #Comportamiento
        def update_name(self, new_name):
            self.name = new_name
```

```
In [ ]: janis = User()
        dibu = User("Emiliano", "Dibu")
        print(janis.name)
        print(dibu.name)
```

## DESAFIO 1

Estamos armando un curso y queremos modelar con clases los distintos recursos con los que vamos a trabajar. Cada recurso tiene un nombre, la URL donde está publicado, un tipo (para indicar si se encuentra en formato PDF, jupyter o video) y la fecha de su última modificación.

Crear la clase para trabajar con estos datos.

```
In [ ]: #Solución
        class Resource:
            ...
```

- ¿Qué debemos pensar?
  - ¿Qué variables de instancia tiene un recurso? ¿Tiene variables de instancia con valores opcionales?
  - ¿Cuál es el comportamiento? ¿Cuáles son los métodos asociados?

## Observemos este código: ¿qué diferencia hay entre villanos y enemigos?

```
In [ ]: class SuperHero():

        villains = []

        def __init__(self, real_name, name):
            self.real_name = real_name
```

```
self.name = name
self.enemies = []
```

- **villains** es una **variable de clase** mientras que **enemies** es una **variable de instancia**.
- ¿Qué significa esto?

## Variables de instancia vs. de clase

Una **variable de instancia** es **exclusiva de cada instancia** u objeto.

Una **variable de clase** es **única** y es **compartida por todas las instancias** de la clase.

## Veamos el ejemplo completo:

```
In [ ]: class SuperHero():
    """ Esta clase define a un superheroe o una superheroína

    villanos: representa a los enemigos de todos los superhéroes
    """
    villains = []

    def __init__(self, real_name, name):
        self.real_name = real_name
        self.name = name
        self.enemies = []

    def get_enemies(self):
        """Retorna los enemigos del superhéroe recetor del mensaje"""
        return self.enemies

    def add_enemy(self, new_enemy):
        """Agrega un enemigo a los enemigos del superhéroe"""
        self.enemies.append(new_enemy)
        SuperHero.villains.append(new_enemy)
```

```
In [ ]: batman = SuperHero( "Bruce Wayne", "Batman")
        ironman = SuperHero( "Tony Stark", "ironman")

        batman.add_enemy("Joker")
        batman.add_enemy("Pinguino")
        batman.add_enemy("Gatubela")

        ironman.add_enemy("Whiplash")
        ironman.add_enemy("Thanos")
```

```
In [ ]: # OJO que esta función está FUERA de la clase
def all_villains(name, villains):
    """Muestra un listado con todos los villanos de name"""
    print("\n"+"*"*40)
    print(f"Enemigos de {name}")
    print("*"*40)
    for villain in villains:
        print(villain)
```

```
In [ ]: all_villains(batman.name, batman.get_enemies())
        all_villains(ironman.name, ironman.get_enemies())
```



```
In [ ]: all_villains("todos los superhéroes", SuperHero.villains)
```

## Volvamos a este código: ¿no hay algo que parece incorrecto?

```
In [ ]: class SuperHero():
        villains = []

        def __init__(self, real_name, name):
            self.real_name = real_name
            self.name = name
            self.enemies = []
```

```
In [ ]: batman = SuperHero("Bruce", "Batman")
        print(batman.name)
```

## Público y privado

- Antes de empezar a hablar de esto ....

"«Private» instance variables that cannot be accessed except from inside an object **don't exist in Python.**"

- De nuevo.. en español..

"Las variables «privadas» de instancia, que no pueden accederse excepto desde dentro de un objeto, **no existen en Python**""

- ¿Y entonces?
- Más info: <https://docs.python.org/3/tutorial/classes.html#private-variables>

## Hay una convención ..

.. que permite **definir el acceso** a determinados métodos y atributos de los objetos, quedando claro qué cosas se pueden y no se pueden utilizar desde **fuera de la clase**.

**Por convención,**

todo atributo (variable de instancia o método) que comienza con "\_" se considera **no público**.

- Pero esto no impide que se acceda. **Simplemente es una convención** que VAMOS a respetar.

## Privado por convención

```
In [ ]: class User():
        """Define la entidad que representa a un usuario en PyTrivia"""

        def __init__(self, name="Sara Connor", nick="mamá_de_John"):
            self._name = name
            self.nick = nick
```

```
self.avatar = None
```

```
def set_name(self, new_name):  
    self._name = new_name
```

```
In [ ]: obj = User()  
print(obj._name)
```

- Hay otra forma de indicar que algo no es "tan" público: agregando a los nombres de las variables o funciones, dos guiones (\_\_) delante.

## Veamos este ejemplo: códigos secretos

```
In [ ]: class SecretCode:  
    '''¿¿¿Textos con clave??? '''  
  
    def __init__(self, plain_text, secret_key):  
        self.__plain_text = plain_text  
        self.__secret_key = secret_key  
  
    def decrypt(self, secret_key):  
        '''Solo se muestra el texto si secret_key es correcta'''  
  
        if secret_key == self.__secret_key:  
            return self.__plain_text  
        else:  
            return ''
```

- ¿Cuáles son las variables de instancia? ¿Públicas o privadas?
- ¿Y los métodos? ¿Públicos o privados?
- ¿Cómo creo un objeto **SecretCode**?

## Codificamos textos

```
class SecretCode:  
    '''¿¿¿Textos con clave??? '''  
  
    def __init__(self, plain_text, secret_key):  
        self.__plain_text = plain_text  
        self.__secret_key = secret_key  
  
    def decrypt(self, secret_key):  
        '''Solo se muestra el texto si secret_key es correcta'''  
  
        if secret_key == self.__secret_key:  
            return self.__plain_text  
        else:  
            return ''
```

```
In [ ]: secret_text = SecretCode("Seminario Python", "stark")
```

```
In [ ]: print(secret_text.decrypt("stark"))
```

# ¿Qué pasa si quiero imprimir desde fuera de la clase: `secret_text.__plain_text`?

```
In [ ]: print(secret_text.__plain_text)
```

Entonces, ¿sí es privado?

## Códigos no tan secretos

- Ahora, probemos esto:

```
In [ ]: print(secret_text._SecretCode__plain_text)
```

- Todo identificador que comienza con "\_\_", por ejemplo `__plain_text`, es reemplazado textualmente por `_NombreClase__`, por ejemplo: `_SecretCode__text_plaino`.
- [+Info](#)

## Entonces... respecto a lo público y privado

### Respetaremos las convenciones

- Si el nombre de una variable de instancia o método comienza con "\_" será considerada privada. Por lo tanto no podrá utilizarse directamente desde fuera de la clase.
- Aquellas propiedades que consideramos públicas, las usaremos como tal. Es decir, que pueden utilizarse fuera de la clase.

## getters y setters

```
In [ ]: class Demo:
    def __init__(self):
        self._x = 0
        self.y = 10

    def get_x(self):
        return self._x

    def set_x(self, value):
        self._x = value
```

- ¿Cuántas variables de instancia?
- Por cada variable de instancia **no pública** tenemos un método **get** y un método **set**. O, como veremos más adelante: **propiedades**.
- ¿Por qué no definimos un get o set para las variables públicas?

## Algunos métodos especiales

Mencionamos antes que los "\_\_\_" son especiales en Python. Por ejemplo, podemos definir métodos con estos nombres:

- `__lt__`, `__gt__`, `__le__`, `__ge__`
- `__eq__`, `__ne__`

En estos casos, estos métodos nos permiten comparar dos objetos con los símbolos correspondientes:

- `x<y` invoca `x.__lt__(y)`,
- `x<=y` invoca `x.__le__(y)`,
- `x==y` invoca `x.__eq__(y)`,
- `x!=y` invoca `x.__ne__(y)`,
- `x>y` invoca `x.__gt__(y)`,
- `x>=y` invoca `x.__ge__(y)`.

```
In [ ]: class Band():
        """ Define la entidad que representa a una banda .. """

        def __init__(self, name, genres="rock"):
            self.name = name
            self.genres = genres
            self._members = []

        def add_member(self, new_member):
            self._members.append(new_member)

        def __lt__(self, other_band):
            return len(self._members) < len(other_band._members)
```

- ¿Qué implementa el método `__lt__`?
- ¿Cuándo una banda es "menor" que otra?

```
In [ ]: soda = Band("Soda Stereo")
soda.add_member("Gustavo Cerati")
soda.add_member("Zeta Bosio")
soda.add_member("Charly Alberti")

bangles = Band("The Bangles", genres="pop-rock")
bangles.add_member("Susanna Hoffs")
bangles.add_member("Debbi Peterson")
bangles.add_member("Vicki Peterson")
bangles.add_member("Annette Zilinskas")
```

```
In [ ]: menor = soda.name if soda < bangles else bangles.name
menor
```

## El método `__str__`

Retorna una cadena de caracteres (**str**) con la representación que querramos mostrar del objeto.

```
In [ ]: class Band():
        """ Define la entidad que representa a una banda .. """

        def __init__(self, name, genres="rock"):
            self.name = name
```

```
self.genres = genres
self._members = []

def add_member(self, new_member):
    self._members.append(new_member)

def __str__(self):
    return (f"{self.name} está integrada por {self._members}")
```

```
In [ ]: bangles = Band("The Bangles", genres="pop-rock")
bangles.add_member("Susanna Hoffs")
bangles.add_member("Debbi Peterson")
bangles.add_member("Vicki Peterson")
bangles.add_member("Annette Zilinskas")

print(bangles)
```

[-Info](#)

