



Jul. 2022

# Unidad 3:



# Objetivos de aprendizaje

Ser capaces de:

- Explicar diferentes conceptos de POO y la importancia de la misma
- Instanciar clases
- Definir métodos de instancia y pasarles argumentos
- Declarar atributos y métodos de clase
- Sobrescribir métodos
- Implementar la herencia múltiple

# Introducción

- La programación orientada a objetos (POO) es un paradigma de programación basado en el concepto de objetos
- Los "objetos" son estructuras de datos que contienen datos en forma de atributos y funciones conocidos como métodos

Kindler, E.; Krivy, I. (2011).

# POO según Steve Jobs

En "[Rolling Stone](#)" 16 de junio de 1994

"Los objetos son como las personas.

Son cosas vivas,  
que respiran,  
que tienen conocimientos en su interior  
sobre cómo hacer las cosas  
y que tienen memoria en su interior  
para poder recordar cosas.

Y en lugar de interactuar con ellos a un nivel muy bajo,  
se interactúa con ellos a un nivel de abstracción muy alto..."

## Steve Jobs in 1994: The Rolling Stone Interview

Even at one of the low points in his career, Jobs still had confidence in the limitless potential of personal computing

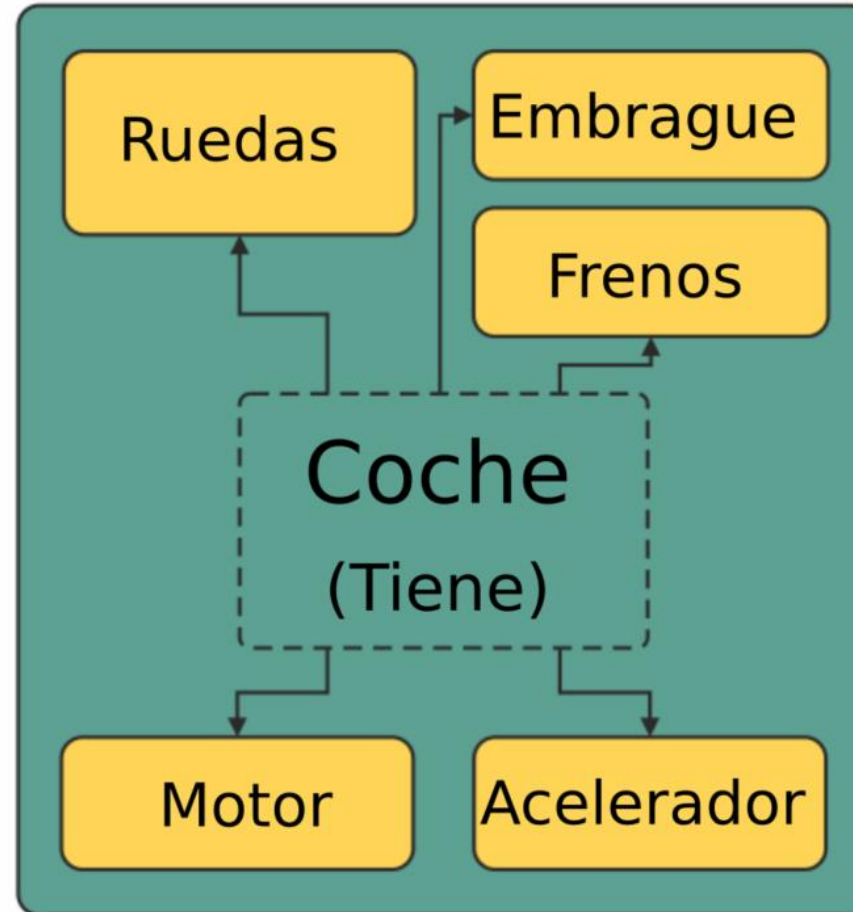
By JEFF GOODELL





# Ejemplo de Objeto

- Múltiples atributos
- Comportamientos específicos
  - Arrancar
  - Acelerar
  - Desacelerar
  - cambiar de marcha



# La Programación Orientada a Objetos (POO)

- Todo en Python es un objeto

La POO forma parte del núcleo de Python

- Python es totalmente compatible con este paradigma

- Los dos protagonistas son los objetos y las clases

- Las clases se utilizan para crear objetos
- Los objetos cuando son creados por una clase
  - heredan los atributos y métodos de la clase

```
<'int'>  
<'str'>  
<'bool'>  
<'list'>  
<'dict'>  
<'function'>
```

# La POO ofrece las siguientes ventajas

- Hace que el código sea reutilizable
- Facilita el diseño/modelado de software
- Facilita las pruebas, la depuración y el mantenimiento
- Los datos están seguros gracias a la abstracción y la ocultación



# Clase

**Definir:** Utilizando la palabra clave de Python *class*, seguida del nombre de la clase.

```
>>> class Persona:  
... pass  
...
```

la palabra clave *pass* se utiliza como marcador de posición

# Objeto

- Instanciar un objeto de una clase es el acto de construir lo que una clase describe

```
>>> juan = Persona()  
>>> pedro = Persona()  
>>> jose = Persona()
```

```
>>> juan is jose  
  
>>> juan = pedro  
>>> juan is pedro
```

# Objeto

- Instanciar un objeto de una clase es el acto de construir lo que una clase describe

```
>>> juan = Persona()  
>>> pedro = Persona()  
>>> jose = Persona()
```

```
>>> juan is jose  
Falso  
>>> juan = pedro  
>>> juan is pedro  
Verdadero
```

Objetos distintos

Objetos iguales

# Añadir atributos a un objeto

- Un atributo es una característica específica de un objeto
- Escribimos el nombre del objeto seguido de un punto (.) el nombre del atributo a añadir y asignándole un valor

```
>>> persona1 = Persona()  
>>> persona1.nombre = "Juan Perez"
```

dict contiene todos los atributos del objeto

# Añadir atributos a un objeto

- Un atributo es una característica específica de un objeto
- Escribimos el nombre del objeto seguido de un punto (.) el nombre del atributo a añadir y asignándole un valor

```
>>> persona1 = Persona()  
>>> persona1.nombre = "Juan Perez"
```

dict contiene todos los atributos del objeto



Establecer atributos de esta manera es una mala práctica

# El método `__init__`

- Constructor: Para añadir de forma adecuada atributos a un objeto

```
>>> class Persona:  
...     def __init__(self, nombre):  
...         self.name = nombre  
...  
>>>
```

**name** se refiere al nombre de la persona

**self** se refiere al objeto que estamos creando

**self** nos permite obtener o establecer los atributos del objeto dentro de nuestra función

Siempre es el primer argumento de un método de instancia.



# E1: Añadir atributos a una clase

1. Crear una clase TelefonoMovil con sus principales atributos
  1. Tamaño de pantalla
  2. Capacidad de la memoria RAM
  3. Sistema Operativo
2. Instanciar los siguientes teléfonos móviles:



**iPhone 13** con pantalla de 5.5, 6GB de RAM, y sistema operativo iOS

**Samsung Galaxy** con pantalla de 6.8, 8GB de RAM y sistema Android



# Actividad 1: Definir una clase y objetos

Para una plataforma de noticias tecnológicas te han pedido que diseñes un sistema de plantillas para sus teléfonos móviles con una estructura similar a:

El nuevo teléfono [] tiene una pantalla de [] pulgadas de pantalla. [] de RAM y funciona con la última versión de []. Su mayor competidor es el teléfono [], que tiene una pantalla de [] pulgadas, [] de RAM y ejecuta Android.

- Genera la plantilla utilizando los valores de los atributos de los objetos creados en el ejercicio anterior

# Definir métodos en una clase

Los objetos están compuestos por comportamientos conocidos como métodos

Reescribir la clase Persona para incluir el método `presentarse()`.

1. Crear un método `presentarse()` en nuestra clase Persona

```
class Persona:  
    # constructor
```

```
def presentarse(self):  
    print(f"Hola, me llamo {self.name} y tengo {self.age} años")
```

2. Instanciar un objeto y llamar al método que hemos definido

```
jorge=Persona("Jorge", 35)  
jorge.presentarse()
```

# Pasar argumentos a los métodos de instancia

- Se puede pasar argumentos a los métodos de una clase al igual que con las funciones normales

```
def saludar(self, persona):  
    if persona.nombre == "Rogelio":  
        print(f"Buenos días {persona.nombre}")  
    else:  
        print(f"Hola {persona.nombre}")
```

- Llamar al método `saludar(persona)`

## E2: Establecer atributos de instancia dentro de métodos de instancia

- Vamos a crear un método `cumpleaños()` que incremente la edad de la persona.
  1. Implementa el método `cumpleaños()`, que toma la edad y la incrementa en uno
  2. Crea una instancia de persona y comprueba la edad
  3. Llama al método `cumpleaños()` y comprueba nuevamente la edad

# Actividad 2: Definir métodos en una clase

- Construye un programa para calcular la circunferencia y el área de círculos
  - La fórmulas de un círculo para calcular

la **circunferencia** es  $2 * \pi * r$   
el **área** es  $\pi * r * r$

1. Definir la clase **Circulo** y el método constructor con el radio
2. Crear el método **área**, que devuelve el cálculo del área del círculo.
3. Crear el método **circunferencia**, que devuelve la circunferencia del círculo.
4. Solicita como **entrada del usuario** el radio.
5. Crea un bucle **while** para que la solicitud de entrada del usuario se ejecute por siempre
6. Redondea la salida a 2 decimales

Radio del círculo : 5  
Area: 78.54  
Circunferencia: 31.42  
...



# Atributos de clase frente a atributos de instancia

- Inicializar un objeto con atributos específicos aplica esos atributos sólo a ese objeto
- Los atributos declarados dentro del constructor se añaden como atributos de la instancia  
La vinculación de esos atributos a la instancia ocurre en el método `__init__` donde añadimos atributos a `self`

# E3: Declarar una clase con atributos de instancia

- Declararemos una clase **NavegadorWeb** que tiene atributos para
    - el historial
    - la página actual y
    - una bandera booleana que muestra si esta de incógnito o no
1. Inicializa los objetos de la clase
  2. Comprueba el atributo **pagina\_actual** en diferentes instancias de la clase NavegadorWeb

# Atributos de clase

- Los atributos de clase están ligados a la propia clase y son compartidos por todas las instancias
- También podemos definir estos atributos
- Cuando cambiamos el atributo de la clase a través de la misma se reflejará en todas las instancias existentes

## E4: Extendiendo la clase con atributos

1. Añadir a la clase `NavegadorWeb` el atributo `conectado` (booleano a `True`)
2. Instanciamos un objeto `NavegadorWeb`.
3. Acceder a los atributos de la clase a través de la propia clase
4. Imprime los atributos `__dict__` de nuestras instancias
5. ¿Por qué no obtenemos un `AttributeError` cuando intentamos recuperar este atributo?

# E5: Implementación de un contador para las instancias de una clase

Vamos a crear un contador que se incrementará cada vez que se instancie un nuevo objeto NavegadorWeb

1. Añade el atributo de clase `número_de_navegadores_web` que servirá de contador y comenzará en 0
2. Modifica el constructor para incrementar el contador cada vez que se crea una nueva instancia añadiendo la línea
3. Comprueba que el contador está en 0
4. Instala un nuevo objeto y comprueba el contador

# Actividad 3: Creando atributos de clase

- Suponga que está diseñando un software para una compañía de ascensores que involucra un mecanismo de seguridad para prevenir que el elevador sea usado cuando se llenen más allá de su capacidad.

1. Declare la clase `ascensor` añadiendo un atributo de clase límite de ocupación de 8 personas
2. Añade el inicializador, que comprobará si se supera el límite de ocupación, e imprimirá un mensaje indicando cuántas personas deben bajar
3. Cree algunas instancias para probar

- La salida debería ser la siguiente:

```
Ocupantes del ascensor 1: 6
Se excede la capacidad del ascensor 2 ocupantes deben salir.
Ocupantes del ascensor 2: 10
```



# E6: Creación de Métodos de Instancia

Para la clase `NavegadorWeb` implementaremos los métodos `navegar()` y `limpiar_historial()`

1. Añade el método `navegar()` a la clase

Si no estamos en modo incógnito cualquier llamada a `navegar` establecerá la página actual del navegador al argumento *nueva\_pagina* y la añadirá al historial

2. Desde una instancia Llamar a `navegar()` debería cambiar `pagina_actual`

3. Crea el método `limpiar_historial` que borrará el historial del navegador hasta el último elemento dejando sólo la página actual en la lista

4. Añade al historial del navegador un par de páginas y luego llama al método `limpiar_historial()` para ver si funciona

# Métodos de clase

- Difieren de los métodos de instancia en que están ligados a la propia clase y no a la instancia
- No tienen acceso a los atributos de la instancia
- Pueden ser llamados a través de la propia clase
- No requieren la creación de una instancia de la clase
- Los métodos de clase el primer parámetro es siempre la propia clase
- Pueden ser utilizados para devolver objetos de un tipo diferente o con diferentes atributos
- La definición de la función comienza con `@classmethod`
  - añade la función que está debajo como un método de clase
- Siguiendo línea declaramos la función que toma el argumento `cls` (clase)

# Método de fábrica

- Un método de fábrica devuelve objetos
- Un caso de uso común para los métodos de clase es cuando se hacen métodos de fábrica
- Pueden ser utilizados para devolver objetos de un tipo diferente o con diferentes atributos

# E7: Probando el método de fábrica

A la clase Navegador vamos a añadir un método de clase llamado `incognito()` que inicializa un objeto del navegador web en modo incógnito:

1. Imprime el método de la clase
2. Crea una instancia de la clase que se inicia en modo incógnito
3. Imprime la página actual de nuestra instancia para comprobar si se ha configurado
4. Confirmar que no se ha rastreado el historial
5. Llamar a los métodos de la clase a través de las instancias para conseguir el mismo efecto

## E8: Accediendo a los Atributos de la Clase desde los Métodos de la Clase

- Los métodos de clase también tienen acceso a los atributos de la clase
- Dado que la mayoría de los navegadores actuales tienen una API de geolocalización simularemos esta funcionalidad en la clase
- En la clase NavegadorWeb crearemos un atributo `geo_coordenadas` que contiene la latitud y la longitud actuales
- También añadiremos un método de clase llamado `cambiar_geo_coordenadas()` que cambiará las coordenadas al ser llamado

# E9: Accediendo a los Atributos de la Clase desde los Métodos de la Clase

1. Añade el atributo de clase `geo_coordenadas` igual a `latitud:39.466667` y `longitud: -0.375`
2. Añade el método de clase `cambiar_geo_coordenadas()` que toma el parámetro `nuevas_coordenadas` que es **un diccionario**.
3. El método de clase comprueba si la `latitud` y la `longitud` proporcionadas en los parámetros son válidas, recuerda que los valores validos para:
  - la `latitud` -> debe estar dentro del rango entre -90 y 90 grados
  - la `longitud` -> debe estar dentro del rango entre -180 y 180 grados
4. Crea `firefox` como una instancia de `NavegadorWeb` para comprobar sus `geocoordenadas`
5. Llamar a `cambiar_geo_coordenadas` en la clase
6. ¿Que sucedió?



# Atributos de instancia VS atributos de clase

- Atributos de Instancia
  - Están vinculados a la instancia
  - Sólo se pueden recuperar a través de la instancia
  - La modificación de este valor sólo lo hace para la instancia actual
- Atributos de Clase
  - Están vinculados la clase
  - Se puede acceder a ellos a través de la instancia y de la clase
  - Al cambiar este valor, se modifica para todas las instancias

# Métodos de instancia VS métodos de clase

- Métodos de Instancia
  - Están vinculados a la instancia
  - Sólo pueden ser llamados a través de una instancia
  - Toman la instancia propia como primer argumento
  - Tienen acceso a los atributos de la instancia y de la clase
- Métodos de Clase
  - Están vinculados a la clase
  - Se puede acceder a través de la instancia y de la clase
  - Toman la clase como primer argumento
  - Sólo tienen acceso a los atributos de la clase

# Encapsulación y ocultación de información

- Uno de los conceptos clave de la POO
- La encapsulación es la agrupación de datos
  - con los métodos que operan sobre esos datos.
- Se utiliza para ocultar el estado interno de un objeto agrupando y proporcionando métodos que pueden obtener y establecer el estado del objeto a través de una interfaz.
- Esta ocultación del estado interno de un objeto se llama ocultación de información.
- En Python, la ocultación de información se realiza marcando los atributos como privados o protegidos
  - **atributos private** sólo deben usarse dentro de la clase y no accedidos externamente.
  - **atributos protegidos** similares a los privados, pueden ser utilizados en contextos muy específicos.
- Por defecto, todos los atributos son públicos.

# Encapsulación y ocultación de información

Python, implementa los modificadores de acceso a los atributos en los propios nombres de los atributos

**Atributos protegidos**, anteponemos al nombre del atributo un **guión bajo**, `_`.

Para indicar que está protegido:

```
self._velocidad = 300  
self._color = "negro"
```

**Atributos privados**, anteponemos al nombre del atributo un **doble guión bajo** `__`.

Esto hace que el atributo sea inaccesible desde fuera de la clase.

El atributo sólo puede ser obtenido y establecido desde dentro de la clase:

```
self.__velocidad = 300  
self.__color = "negro"
```

# Encapsulación y ocultación de información

- Getters y Setters
  - Para cambiar el atributo privado `__velocidad`, debemos utilizar el método **setter** definido `cambiar_velocidad`.
  - Para obtener el atributo de velocidad desde fuera de la clase si es necesario, podemos utilizar el método **getter** “`obtener_velocidad`”

# Actividad 4: Creación de métodos de clase y uso de la ocultación de información

Suponga que trabaja para una compañía de electrónica que tiene un nuevo dispositivo reproductor de música que quiere lanzar al mercado.

El software para este dispositivo necesita soportar actualizaciones over-the-air que permitan a los usuarios escuchar sus canciones favoritas sin problemas.

Crea una clase que represente un reproductor de música portátil, "ReproductorMusica".

La clase ReproductorMusica debe tener un método de reproducción, que establece la primera pista de la lista de reproducción como la que se está reproduciendo actualmente.

La lista de reproducción debe ser un atributo privado.

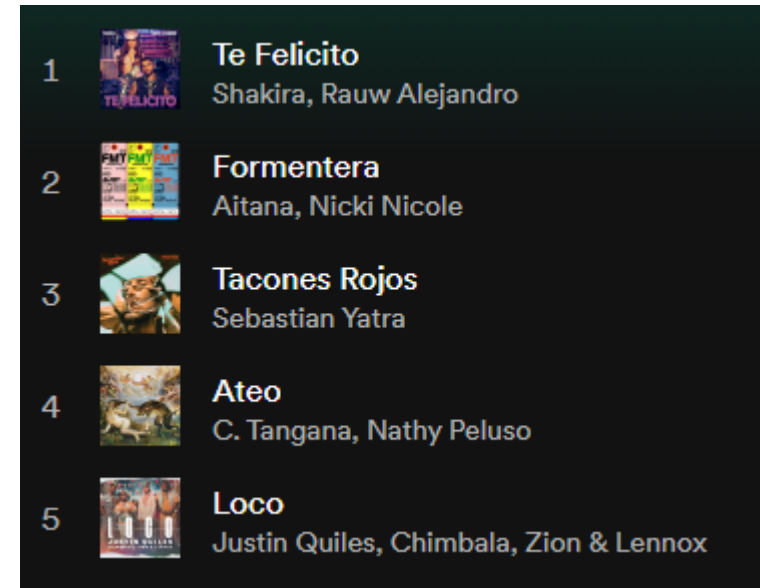
Además, debe tener un atributo de versión de firmware

y un método de clase actualizar\_firmware que actualice la versión del firmware

# Actividad 4: Creación de métodos de clase y uso de la ocultación de información

1. Define la clase ReproductorMusica añadiendo el atributo de clase `version_firmware`

2. Define el método inicializador y pon en la lista de reproducción las siguientes 5 canciones:



3. Asegúrate de que la lista de reproducción es `privada`.

# Actividad 4: Creación de métodos de clase y uso de la ocultación de información

4. Define el método `play`, que establece el atributo `cancion_actual` al primer elemento de la lista de reproducción.
5. Define el método `listar_canciones` que devuelve la lista de reproducción del `ReproductorMusica`.
6. Añadir el método `actualizar_firmware` que comprueba antes de actualizar si la nueva versión del firmware que se proporciona es más reciente que la versión actual
7. Ejecutar el script para probar el reproductor

- Salida de la ejecución del script:

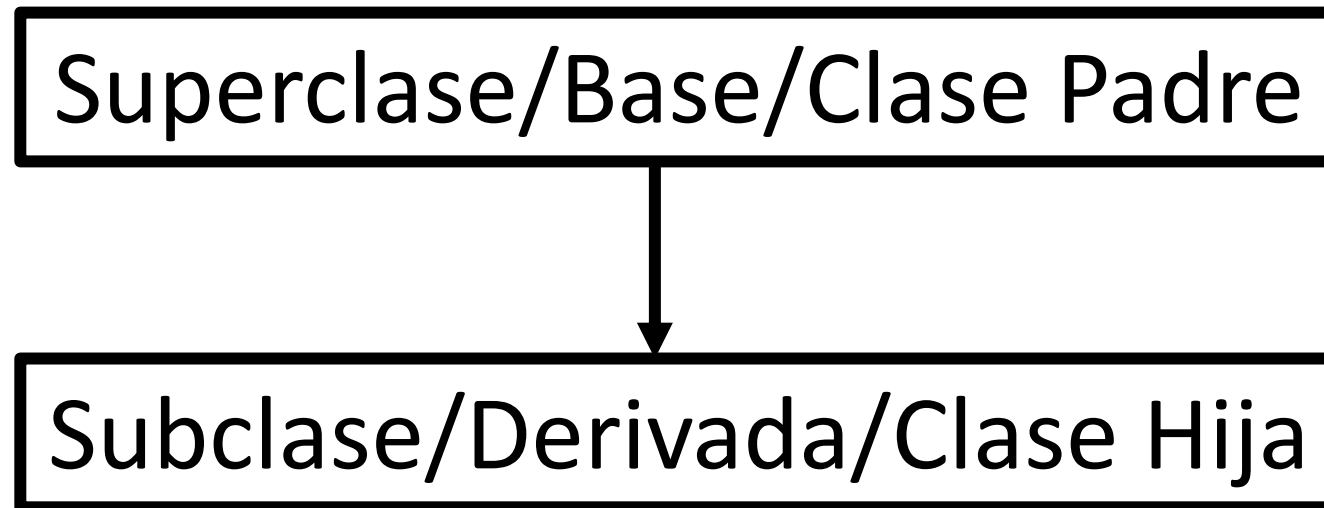
```
Tu lista de reproducción es: ['Te Felicito', 'Formentera', 'Tacones', 'Ateo', 'Loco']  
versión de firmware: 1.0  
versión de firmware: 2.0  
Reproduciendo actualmente: Formentera
```



# Herencia de clases

Una característica clave de la POO.

Es un mecanismo que permite que la implementación de una clase derive de la implementación de otra clase.



# Herencia de clases

- Un ejemplo práctico de herencia en el mundo real es el de los felinos

- son todos felinos que comparten las mismas propiedades
  - peso,
  - vida promedio,
  - velocidad, y
- comportamientos como
  - hacer sonidos y
  - cazar,
  - entre otros.

Los gatos



Los guepardos



los leopardos



los leones



los tigres



# E10: Implementación de la Herencia de Clases

Definiremos la clase Felino de la que derivaremos nuestros felinos.

- La clase tendrá los métodos `hacer_sonido` y `imprimir_datos`, y los atributos `peso`, `vida media` y `velocidad`.
- El método constructor tomará los argumentos `peso`, `vida media` y `velocidad`, a partir de los cuales añadirá los atributos: `peso_en_kg`, `vida media_en_años` y `velocidad_en_kph` al objeto.
- El método `hacer_sonido` imprimirá “Grrrr....” una vocalización no amenazante que es común a varios felinos.
- El método `imprimir_datos` imprimirá datos sobre el felino como por ejemplo:

“El gato pesa 4kg, tiene una vida de 18 años y puede correr a una velocidad máxima de 48kph.”

# E10: Implementación de la Herencia de Clases

Definiremos la clase Felino de la que derivaremos nuestros felinos.

1. Define la clase Felino
2. Instanciar una instancia de Felino e interactuar con los diferentes métodos y atributos que tiene
3. Crea las subclases Leopardo, Gato y León, que heredarán de la clase Felino
4. Instanciar las nuevas clases Leopardo, Gato y León.

# E10: Implementación de la Herencia de Clases

- Si llamamos a `hacer_sonido` a nuestras instancias, todas tienen el mismo comportamiento

Podemos rectificar esto sobrescribiendo el método en nuestra clase.

- **Sobrescribir** significa redefinir la implementación de un método definido en una superclase para añadir o cambiar la funcionalidad de una subclase.
- Sobrescribir el método `sonido` para nuestras subclases:

```
class Gato(Felino):  
    def hacer_sonido(self):  
        print("Miau...")
```

```
class Leon(Felino):  
    def hacer_sonido(self):  
        print("Gr...")
```

# Sobrescritura de `__init__()`

- Muchos felinos grandes tienen un patrón en su pelaje; tienen manchas o rayas. Añadamos esto a nuestra subclase Gato.

# E11: Sobrescribir el método `__init__` para añadir un atributo

Sobreescribiremos el método `__init__` y añadiremos un atributo `pelaje_manchado`

1. Sobrescribe el método inicializador y añade el atributo `pelaje_manchado`
2. Sólo añadirá el atributo `pelaje_manchado` a la instancia.  
Si inicializa la clase Gato recién modificada, debería dar un error al intentar acceder a los atributos originales.
3. Invocar el método `__init__` de la clase Felino dentro del método `__init__` de la subclase Gato antes de añadir el atributo `pelaje_manchado`.

# Otros métodos comúnmente sobrescritos

- Los métodos especiales en las clases de Python siempre llevan el prefijo y el sufijo de doble guión bajo
- Se conocen como métodos Dunder (doble guión bajo) o mágicos.
- Además del método `__init__`, hay otros métodos mágicos en Python que se pueden sobrescribir para personalizar tu clase y añadir funcionalidad personalizada, como cambiar el aspecto de la salida de tu objeto impreso o cómo se comparan tus clases.
- El método que define la salida cuando se llama a `print` en tu objeto, `__str__()`,
- El método que se llama cuando se destruye un objeto `__del__()`.



# El método `__str__()`

Cada objeto en Python tiene el método `__str__()` por defecto. Se llama cada vez que se llama a `print()` sobre un objeto en Python para recuperar la cadena que contiene la representación legible del objeto.

Reemplacemos el método `imprimir_datos()` de la clase `Felino` por este método

# El método `__str__()`

- Ahora, cuando llamemos a `print()` sobre cualquier instancia de Felino o subclase de Felino, debería tener el mismo resultado que cuando llamamos a `imprimir_datos()`:

```
print(guepardo)
```

# El método `__del__()`

- El método `__del__()` es el método destructor.
- El método destructor es llamado cada vez que un objeto es destruido
- Si llamamos a `del` en una instancia de `Tigre`,
- debería imprimir ese mensaje:

```
>>> tigre = Tigre(72, 12, 120)
>>> del tigre
No se ha dañado a ningún animal al borrar esta instancia
>>> tigre
NameError: el nombre 'guepardo' no está definido
```

# Actividad 5: Sobreescritura de métodos

- Suponga que trabaja para una organización de conservación de la vida salvaje. Está trabajando en la creación de un sistema para educar al público en general sobre diferentes animales y hacer que se interesen más en la conservación.
- Cree una clase Tigre que herede de la clase Felino y tenga un nuevo atributo de patrón de pelaje.
- Cambia el comportamiento de las instancias de la clase Tigre para que incluyan este dato de patrón de pelaje cuando se impriman.

# Actividad 5: Sobreescritura de métodos

1. Crear el archivo tiger.py.
2. Definir la clase Felino.
3. Definir la clase Tigre que hereda de la clase Felino.
4. Sobreescibir el inicializador y añadir un atributo patron\_pelaje.
5. Sobreescibir el método `__str__()` modificándolo para incluir la mención del patrón de pelaje.
6. Inicializar una instancia de la clase Tigre.  
Al llamar a print sobre la instancia, debería mostrar datos sobre el tigre.

# Actividad 5: Sobreescritura de métodos

El aspecto debería ser el siguiente:

El tigre pesa 5kg, tiene una vida de 5 años y puede correr a una velocidad máxima de 5 kph  
y el patron del pelaje es a rayas  
No se ha dañado a ningún animal al borrar esta instancia.



# Herencia múltiple

- La herencia múltiple es potente, pero también puede ser muy complicada, por lo que tenemos que asegurarnos de que entendemos lo que sucede cuando la usamos.



# Herencia múltiple

La herencia múltiple es una característica que permite heredar atributos y métodos de más de una clase.

El caso de uso más común para la herencia múltiple son métodos/atributos que están destinados a ser utilizados por otras funciones.

Por ejemplo,  
una clase `Logger` tendría un método `log()`  
que escribe en un archivo de registro,  
y cuando se añade a sus clases les daría esa misma capacidad.

# E12: Implementación de la Herencia Múltiple

- En el mundo real, los leones y los tigres pueden aparearse naturalmente para crear un híbrido conocido como ligre o tigon.
  - Los ligres son mucho más grandes que los leones o los tigres, son sociales como los leones,
  - tienen rayas y, al igual que los tigres, les gusta nadar.
- 
- Vamos a crear una clase Ligre que herede de la clase Leon y Tigre que vamos a definir.
- 
1. Definir las clases León y Tigre
  2. Definir la clase Liger, que heredará de las clases Tigre y León

# E12: Implementación de la Herencia Múltiple

3. Al probarlo, deberíamos ver que la clase Liger ha heredado atributos de las clases León y Tigre.
4. La clase Liger tiene el atributo `patron_pelaje` y el método `nadar()` de la clase Tigre y el atributo `es_social` de la clase León

# Actividad 5: Sobreescritura de métodos

El aspecto debería ser el siguiente:

```
El leon pesa 190kg, tiene una vida de 14 años y puede correr a una velocidad máxima de 80 kph.  
El tigre pesa 310kg, tiene una vida de 26 años y puede correr a una velocidad máxima de 65 kph y el patron  
del pelaje es a rayas  
Salpicar!!!  
El ligre es social? True  
Patrón de pelaje: a rayas
```

# Actividad 6: Practicando la herencia múltiple

- Estamos en el año 2000. Estás trabajando para una empresa de telefonía móvil y te han encargado modelar el software de un teléfono de última generación que tendrá una cámara incorporada: un teléfono con cámara.
- Crea una clase llamada Cámara y una clase llamada TeléfonoMóvil que serán las clases base de una clase derivada llamada CámaraTeléfono.
- La clase CamaraTelefono debe ser inicializada con el atributo de memoria y debe tener un método hacer\_foto() que imprima el mensaje ¡Di queso!

# Actividad 6: Practicando la herencia múltiple

1. Crear una clase Camera que tenga un método hacer\_foto().
2. Crear una clase TelefonoMovil que será inicializada con un atributo de memoria.
3. Crear una clase CamaraTelefono que herede de las clases TelefonoMovil y Camera.
4. Inicializar una instancia de la clase CamaraTelefono.
5. Llamar al método hacer\_foto() en la instancia debería tener una salida como esta. Además, imprime el atributo de memoria:

# Actividad 6: Practicando la herencia múltiple

- La salida debería ser la siguiente:

```
¡Di queso!  
Memoria restante 198k
```





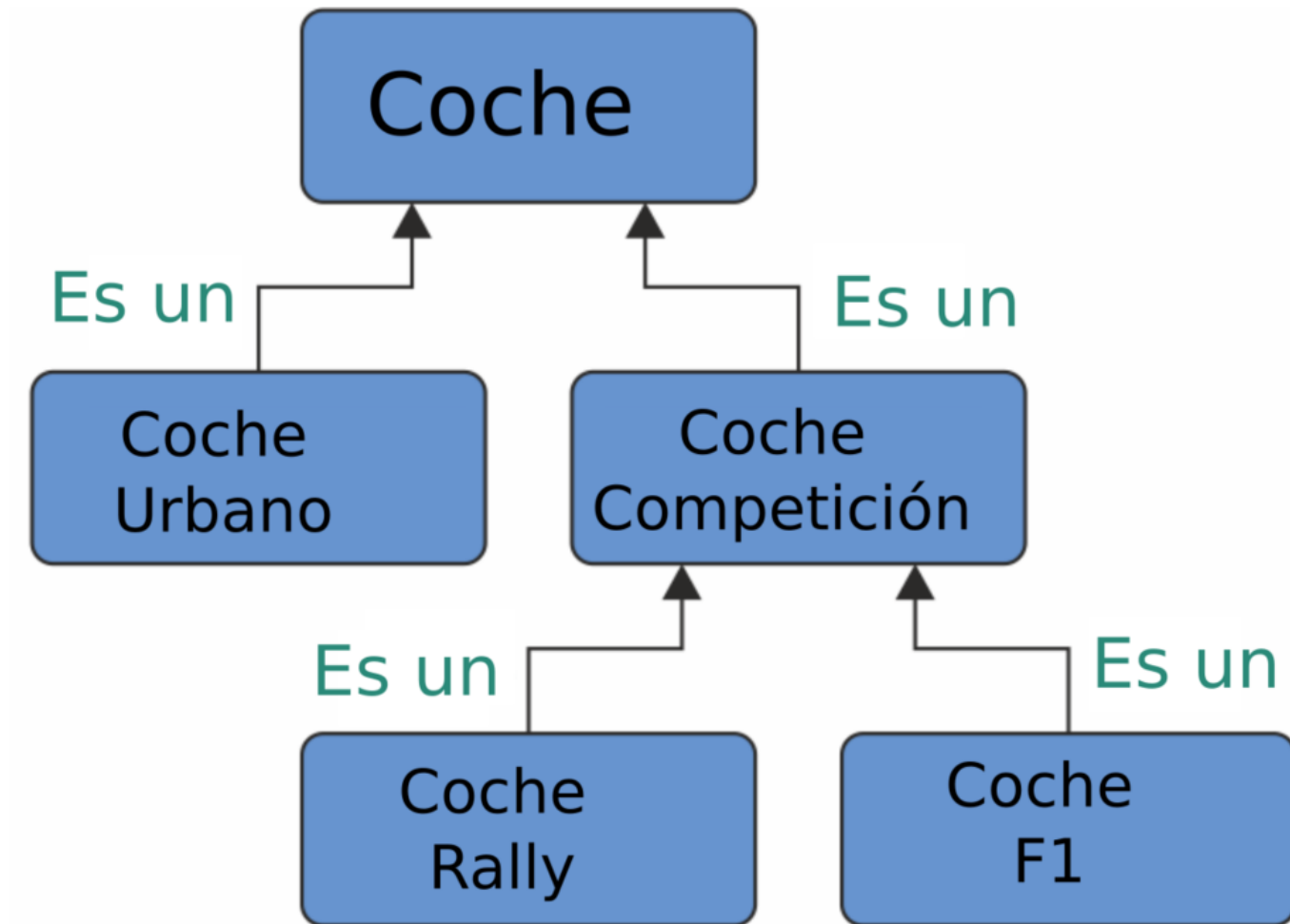
# Resumen

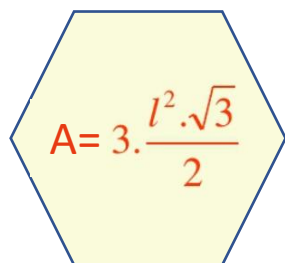
- La POO hace que
  - el código sea más reutilizable;
  - facilita el diseño del software;
  - hace que el código sea más fácil de probar, depurar y mantener; y
  - añade una forma de seguridad a los datos de una aplicación.
- Los comportamientos de un objeto se conocen como métodos, y puedes añadir un método a una clase definiendo una función dentro de ella.
- Para estar vinculada a tus objetos, esta función necesita tomar el argumento self.
- También cubrimos los atributos de la clase y los métodos de la clase en detalle.
- También echamos un vistazo a la encapsulación y a las palabras clave que permiten ocultar información en Python.

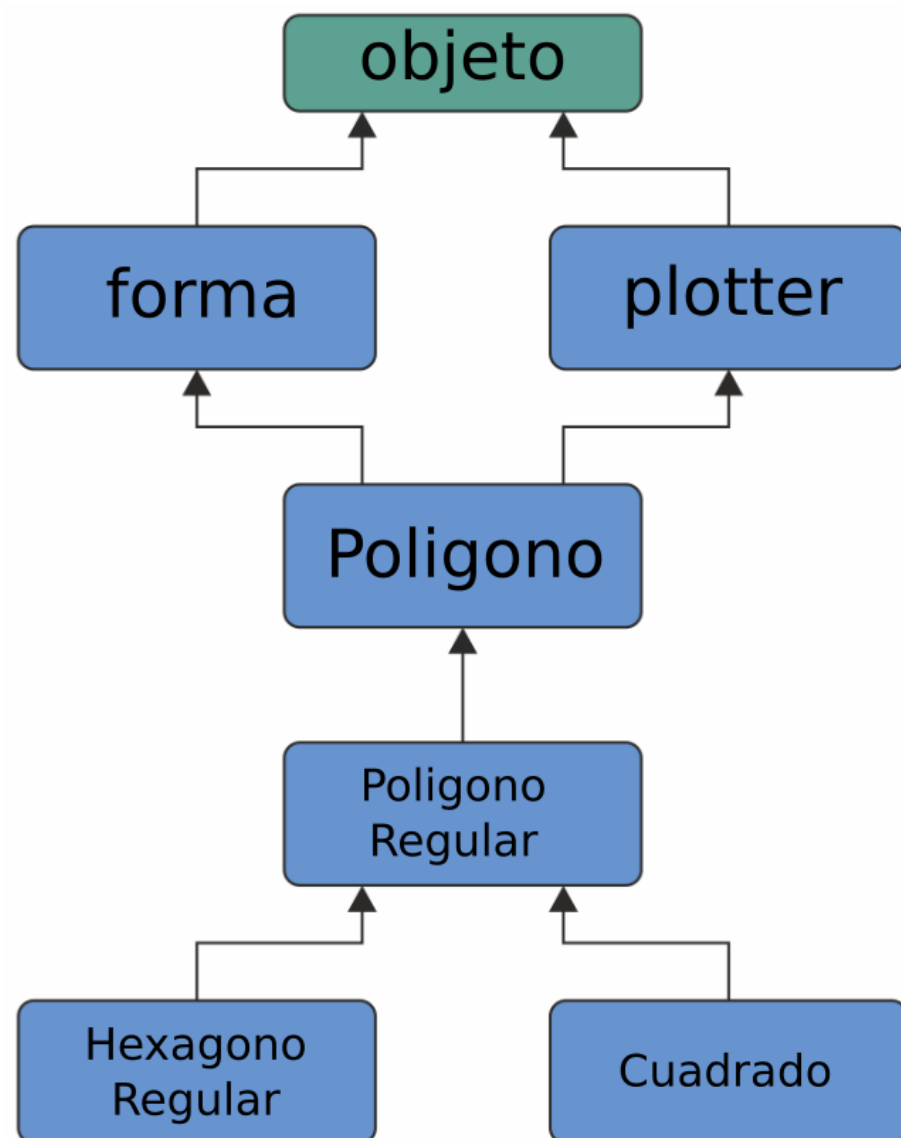
# Resumen

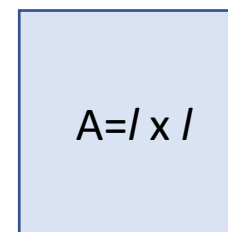
- La ocultación de información se utiliza para abstraer de los usuarios detalles irrelevantes sobre la clase.
- También cubrió la herencia en detalle.
- Vimos cómo hacer que una clase derivada herede de una sola clase base, así como de múltiples clases base.
- También vimos cómo sobrescribir métodos:
  - `__init__()`,
  - `__str__()` y
  - `__del__()`.

# Ejercicios Extras:




$$A = 3 \cdot \frac{l^2 \cdot \sqrt{3}}{2}$$




$$A = l \times l$$