

TI 3001 C

Analítica de datos y herramientas de inteligencia  
artificial

## Python orientado a objetos

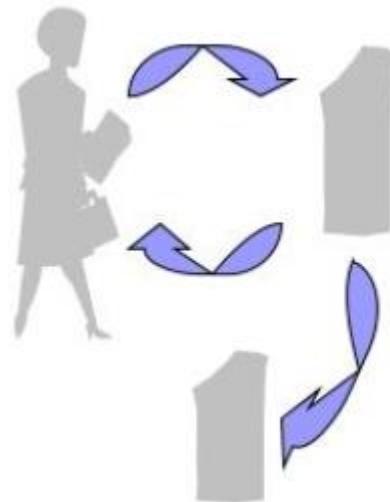
Tecnológico de Monterrey



# Python orientado a objetos

Python es un **lenguaje multiparadigma**: soporta la **programación funcional**, pero también la **programación orientada a objetos**.

Funcional



Retirar, depositar, transferir

Orientada a objetos



Cliente, dinero, cuenta

# Python orientado a objetos

Programación Funcional → Acciones → Función

Programación Orientada a Objetos → Orientada a Objetos → Clase

# Python orientado a objetos

**En Python todo es un objeto.**

Cuando creas una variable y le asignas un valor entero, ese valor es un objeto; una función es un objeto; las listas, tuplas, diccionarios, etc., ... son objetos; una cadena de caracteres es un objeto. Y así podríamos seguir indefinidamente.

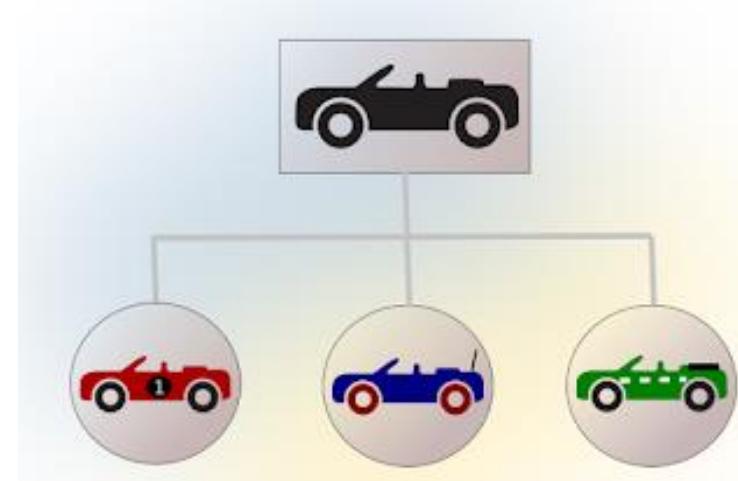


# Programación Orientada a Objetos

Es una técnica de diseño y programación

Algunos términos:

- **Objeto:** Usualmente una persona, lugar o cosa.
- **Método:** Una acción ejecutada por el objeto (un verbo)
- **Clase:** Una categoría de objetos similares (como automóviles)

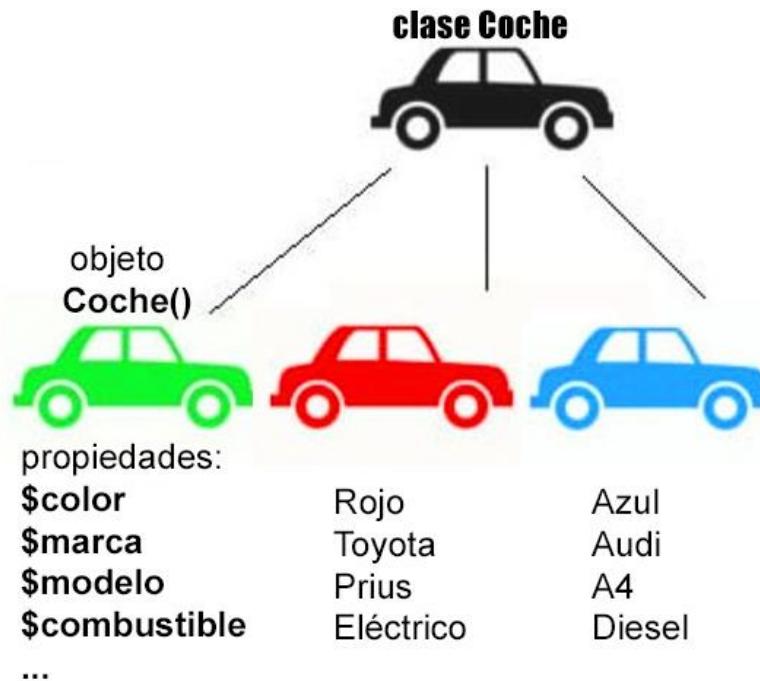


# Programación Orientada a Objetos

Los objetos tienen **datos (atributos)** y **métodos**.

Los objetos de la misma clase tienen los mismos **atributos** y **métodos**.

Los objetos envían y reciben mensajes para invocar acciones.

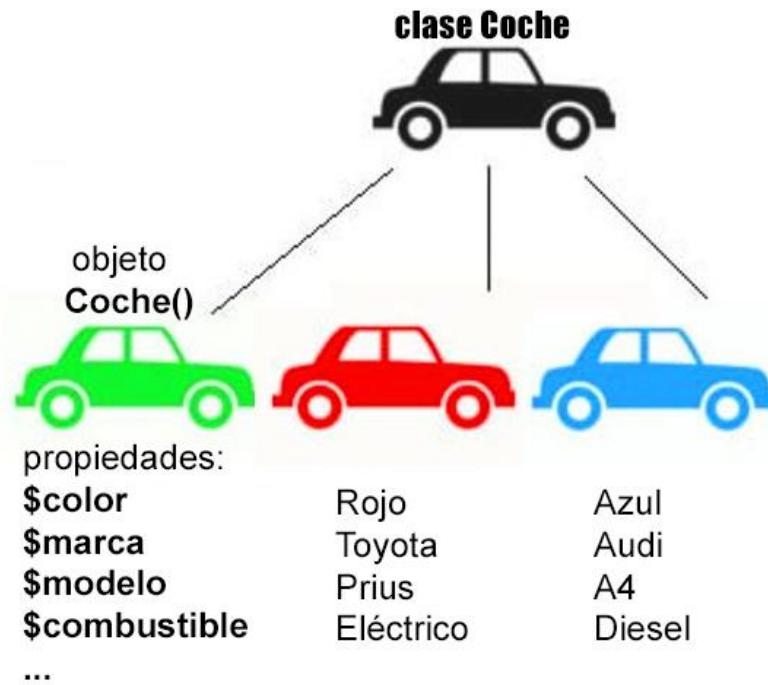


## MÉTODOS

llenarDeposito()  
arrancarMotor()  
frenar()  
acelerar()  
tocarClaxon()  
...

# ¿Qué es una clase?

- Una clase es una generalización de un objeto.
- Una vez que especificamos o creamos una **clase** podemos construir cualquier número de objetos que pertenezca a dicha clase y que por lo tanto tengan los mismos **datos** y el mismo **comportamiento**.



## MÉTODOS

llenarDeposito()  
arrancarMotor()  
frenar()  
acelerar()  
tocarClaxon()

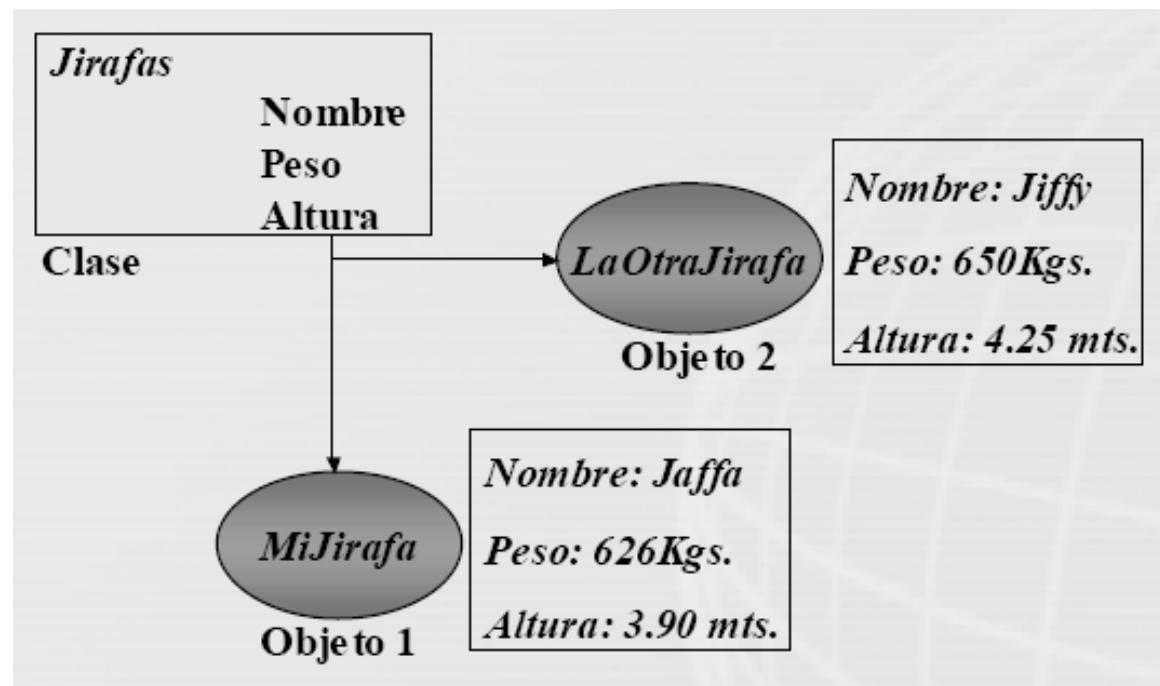
...

# ¿Qué es un objeto?

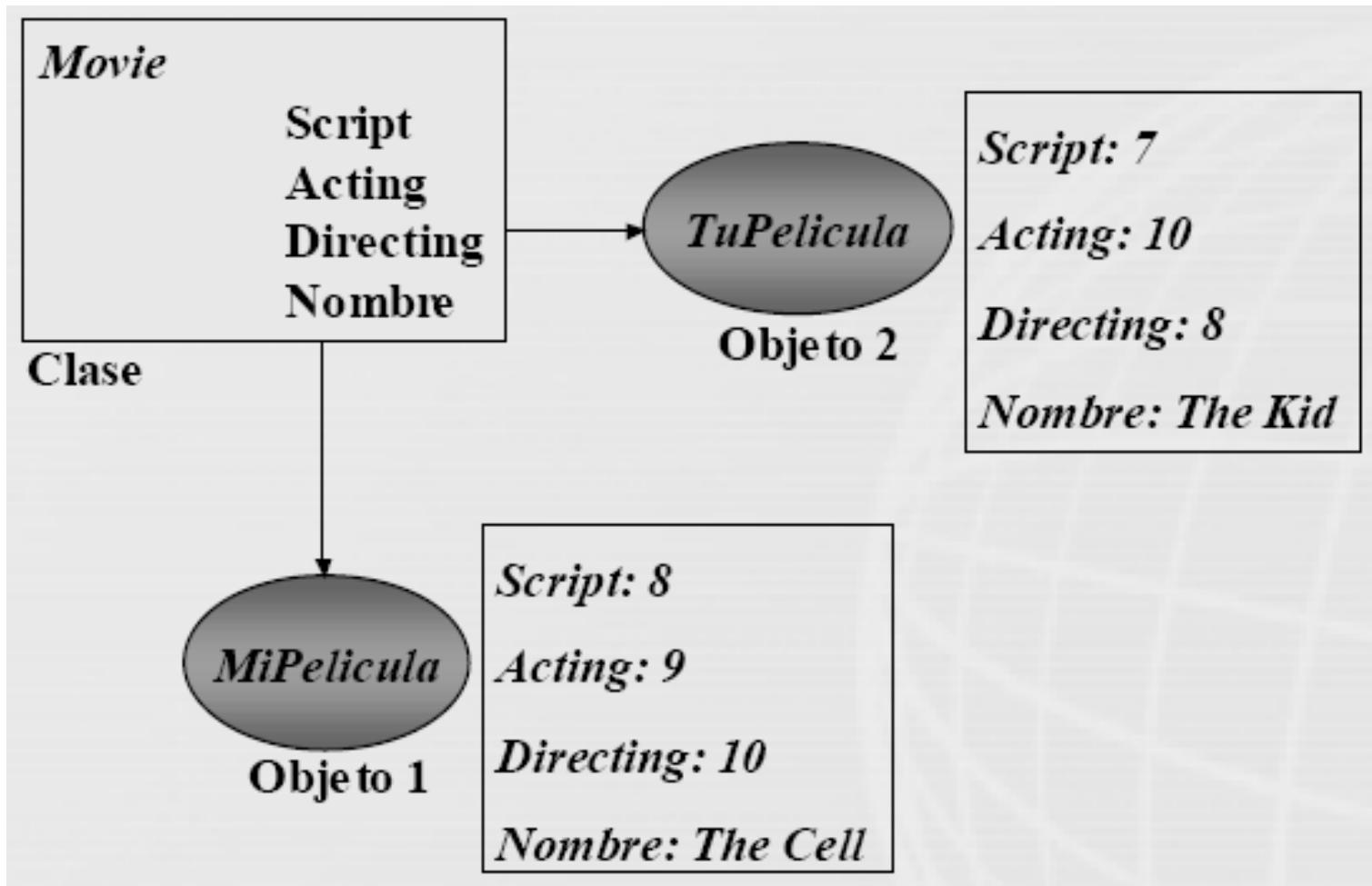
Es una entidad particular de la clase y sus características están definidas con valores únicos para ese objeto.

**Ejemplo:** MiJirafa, LaJirafaDelCircoAtayde, LaJirafaDelZoológico

Nótese que Jirafa es la **clase** que contiene las características propias de todas las jirafas



# Movie



# ¿Qué es un objeto?

**Un objeto es una entidad o todo aquello que se quiere representar (persona, lugar o cosa).**

Los objetos los vamos a representar a través de

- **Atributos:** Características que lo definen. Son variables (int, bool, float, char , etc.) van a tener distintos valores, dependiendo de que objeto se trate.

Si tenemos un objeto **vendedor**, vamos a tener un objeto que represente al vendedor **Juan** y otro objeto que represente a la vendedora **Luisa**. Uno de los atributos de ese objeto puede ser el **nombre**, que puede ser Juan o Luisa. Diferentes valores para un atributo, definen el estado de un objeto.



# ¿Qué es un objeto?

**Un objeto es una entidad o todo aquello que se quiere representar (persona, lugar o cosa).**

Los objetos los vamos a representar a través de

- **Métodos:** **Acciones que puede realizar.** Son similares a las funciones. Pueden recibir parámetros y regresar un valor.
- Los **métodos** hacen o modifican el estado interno del objeto.

En general el objeto es todo aquello que queramos representar y lo vamos a representar a través de **atributos** y **métodos**, de esta forma vamos a identificar las características propias de los objetos.

# Clases y objetos

Para definir una clase y crear objetos de esa clase:

1. Primero debemos definir la **clase**
  - Al crear la clase debemos definir sus variables de instancia o **atributos**. Debemos preguntarnos **¿Qué datos son necesarios para esta clase?**
  - También debemos definir su comportamiento o **métodos**, es decir, **¿qué pueden hacer los objetos que pertenecen a esta clase?**
2. Después debemos crear los **objetos** de la clase que acabo de definir

# Clases y objetos

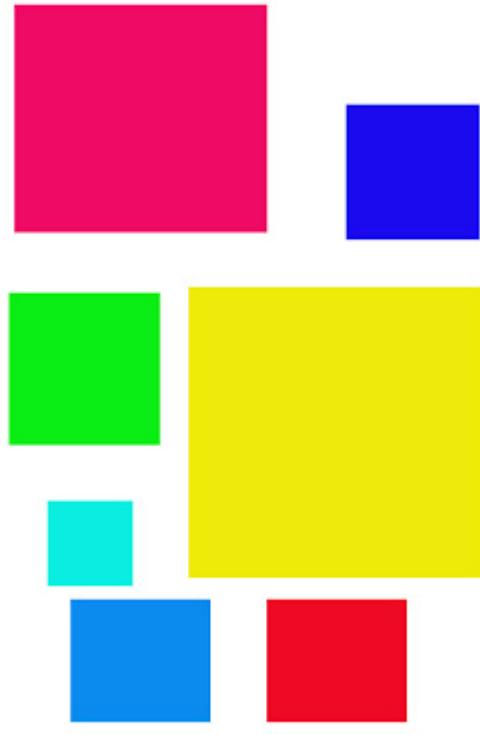
## Ejemplo:

- **Clase: globo** (*Como podrás observar el nombre de la clase lo asociamos con un sustantivo común ya que la clase es la generalización del objeto*)
- **Objetos: globo\_azul, globo\_rojo, globo\_verde** (*El nombre del objeto lo asociamos con un sustantivo propio*)
- **Atributos: x, y, color, diámetro** (*son las características que tienen todos los objetos de una clase determinada*)
- **Métodos o acciones: dibujar\_globo** (*el nombre de los métodos o comportamientos lo asociamos con un verbo*)

# Clases y objetos

Otro ejemplo: clase Cuadrado

- **Clase:** Cuadrado
- **Atributos:** x, y, lado y color
- **Métodos o acciones:**  
`dibuja_cuadrado` y `calcula_area`.

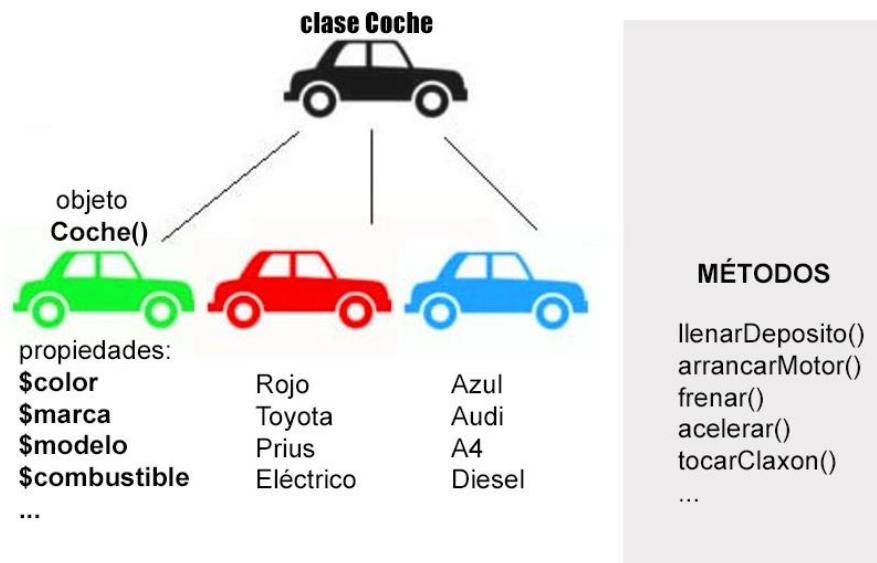


Objetos

# ¿Qué es un objeto?

## Instancia

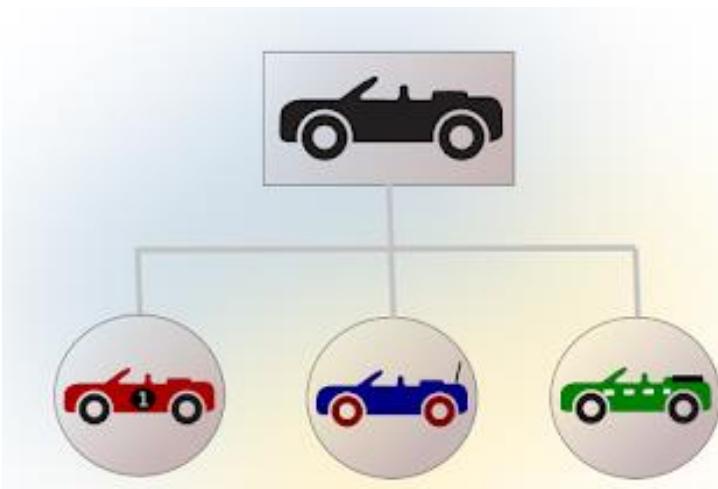
- Cada **coche** está hecho del mismo patrón o modelo y contiene los mismos componentes. En términos de la programación orientada a objetos, se dice que **uno de éstos** carros es una **instancia** de la clase de objetos llamada **Coche**.
- Una **clase** es el modelo o patrón a partir del cual cada objeto individual es creado.



# ¿Qué es una clase?

Un clase es un “molde” para obtener objetos con la misma estructura.

- Para obtener objetos, necesitamos una estructura (molde) que nos permita crear varios objetos con el mismo tipo de atributos y métodos.
- Las clases son una estructura para poder obtener objetos iguales pero con distintos estados. Es decir, vamos a obtener objetos que tengan los mismos atributos y métodos pero con un distinto estado. De esta forma vamos a poder instanciar una clase para poder obtener instancias de ellas que son los objetos.



- “Molde” para obtener objetos con la misma estructura.
- Para obtener un objeto a partir de una clase, se la debe instanciar.
- Objetos de la misma clase: tienen los mismos métodos y atributos, pero los atributos pueden tener distintos valores.



```
class Gato:  
    pass  
  
>>> mi_gato=Gato()
```

# ¿Cómo se crea una clase?

Un clase es un “molde” para obtener objetos con la misma estructura.

`class Nombre_Clase:`

`pass` # Nos dice la clase no está implementada, esta clase está vacía.



`class Gato:`

`pass`

Ya tenemos una clase, ya podemos instanciar objetos de tipo `Gato`.

`mi_gato = Gato()`

A la variable `mi_gato` le asignamos lo que retorna esta operación `Gato()` que es la que instancia un objeto de esta clase.

# Objetos

Vamos a crear varios objetos de tipo **Gato()**

**¿Qué podemos representar de un gato?**

**Atributos:** Nombre, edad, alimentos favoritos

**Métodos:**

- Saber si es adulto.
- Decir si un alimento es de sus favoritos.



# Atributos

- Son propiedades del objeto, que están representadas mediante variables.
- Los atributos definen el objeto y los atributos definen el estado del objeto.
- Puede ser cualquier tipo de dato (incluso otros objetos).
- Podemos tener dos tipos de atributos, dependiendo si el valor es el mismo para todos los objetos o no:
  - **De clase (estáticos)** Son compartidos por todos los objetos de una clase, significa que el valor es el mismo para todos. Estos se utilizan para definir las constantes. Estos valores no cambian para los objetos.
  - **De instancia**
- Pueden crearse de forma **dinámica** durante la ejecución. Python genera un diccionario para los atributos de clase y otro diccionario para los atributos de instancia. Esto por cada uno de los objetos que instanciemos de una clase. La clave de la clase es el nombre del atributo (edad) y el valor del diccionario es el valor del atributo (2 años).

# Métodos

- Dan funcionalidad a los objetos.
- Necesitan de una clase para existir. (Deben estar dentro de una clase)
- Son similares a las funciones (parámetros y valor de retorno)
- Pueden invocarse desde el propio objeto que lo contiene o desde otro objeto, con la sintaxis **objeto.método()**.

**Mi\_gato.nombre\_método**(argumentos del método)

Es importante que tengan nombres representativos que nos digan que hace esa función o método, generalmente se definen con un verbo que identifique la acción que realiza.

# Constructor

Es un método especial que tienen todas las clases  
(se invoca automáticamente)

- Es un método que podemos escribir o no, sino lo escribimos se va a crear uno por **default**.
- Es el método que **nos permite construir objetos de esa clase, es decir, instanciarlos**. Se invoca automáticamente, cada vez que queremos instanciar una clase.
- Da espacio en memoria al nuevo objeto.
- **Nos permite inicializar los atributos de un objeto**. Los atributos que definamos dentro de este método son los atributos de instancia.
- No es obligatorio crearlo ni inicializar todos los atributos dentro de este método, ya que Python nos permite crear atributos dinámicamente. Atributos que no existían que podemos ir creando durante la ejecución.

```
1 class Ejemplo:
2     def __init__(self, parametro1, parametro2):
3         self.atributo1 = parametro1
4         self.atributo2 = parametro2
5
6     def main():
7         un_ejemplo = Ejemplo("un valor", "otro valor")
8         print(un_ejemplo) #Si imprimos el nombre de la variable
9         #podemos ver que se creó un objeto de tipo ejemplo y
10        # que nos dice la dirección de memoria donde está
11        # guardado.
```

Shell >

```
>>> %Run Objetos.py
<__main__.Ejemplo object at 0x040DCBB0>
```

# Constructor

Este constructor recibe dos parámetros.  
El primer parámetro de un método es una referencia del objeto que se está creando.

```
class Ejemplo:  
    def __init__(self, parámetro1, parámetro2):  
        self.atributo1 = parámetro1  
        self.atributo2 = parámetro2  
  
Un_ejemplo = Ejemplo("un valor", "otro valor")
```

Aquí vemos cómo instanciamos un objeto de la clase **Ejemplo**. Cuando ponemos el nombre de la clase automáticamente se está llamando al método **init** y luego pasamos todos los argumentos que el método **init** reciba.

- Cuando mandamos llamar al método, no es necesario poner **self**, se asume que el primer argumento siempre es una **referencia al objeto**.
- El parámetro **self** recibe implícitamente (es decir no necesitamos escribirla) un argumento más con la referencia al objeto que se está creando.
- Una **referencia es una dirección de memoria** (recibe la dirección de memoria del nuevo objeto).

# Constructor

Este constructor recibe dos parámetros.  
El primer parámetro de un método es una referencia del objeto que se está creando.

```
class Ejemplo:  
    def __init__(self, parámetro1, parámetro2):  
        self.atributo1 = parámetro1  
        self.atributo2 = parámetro2  
  
Un_ejemplo = Ejemplo("un valor", "otro valor")
```

Aquí vemos cómo instanciamos un objeto de la clase **Ejemplo**. Cuando ponemos el nombre de la clase automáticamente se está llamando al método **init** y luego pasamos todos los argumentos que el método **init** reciba.

- Cada que queramos hacer referencia a ese objeto, se utiliza **self.atributo** o **self.método** que se quiera invocar.
- Este constructor recibe dos parámetros y luego inicializa dos atributos, dándole el valor de esos dos parámetros. Para poder mencionar a los atributos de ese objeto que acabamos de crear, se utiliza **self**.

# Constructor

Podemos instanciar objetos de tipo Ejemplo:

**Nombre Objeto** = Nombre de la clase (argumentos)

```
1 class Ejemplo:
2     def __init__(self, parametro1, parametro2):
3         self.atributo1 = parametro1
4         self.atributo2 = parametro2
5
6     def main():
7         un_ejemplo = Ejemplo("un valor", "otro valor")
8         print(un_ejemplo) #Si imprimos el nombre de la variable
9             #podemos ver que se creó un objeto de tipo ejemplo y
10            # que nos dice la dirección de memoria donde está
11            # guardado.
```

Shell ×

```
>>> %Run Objetos.py
```

```
<__main__.Ejemplo object at 0x040DCBB0>
```

# Lectura de atributos

Podemos leer los atributos: **Nombre\_variable.atributo**

```
1 class Ejemplo:
2     def __init__(self, parametro1, parametro2):
3         self.atributo1 = parametro1
4         self.atributo2 = parametro2
5
6     def main():
7         un_ejemplo = Ejemplo("un valor", "otro valor")
8         print(un_ejemplo) #Si imprimos el nombre de la variable,
9         #podemos ver que se creó un objeto de tipo ejemplo y
10        # que nos dice la dirección de memoria donde está
11        # guardado.
12        print(un_ejemplo.atributo1)
13        # También podemos leer el valor de los atributos,
14        # poniendo el nombre de la variable punto y el nombre
15        # del atributo o método del objeto.
16        print(un_ejemplo.atributo2)
17
```

Shell ×

```
>>> %Run Objetos.py
<__main__.Ejemplo object at 0x036BCC70>
un valor
otro valor
```

# Constructor

- No necesariamente el constructor debe tener todos los atributos inicializados a partir de parámetros.
- Podemos tener un solo parámetro y que el otro atributo se inicialice con 100.

```
1 class Ejemplo:
2     def __init__(self, parametro1):
3         self.atributo1 = parametro1
4         self.atributo2 = 100
5
6 def main():
7     # Se crea un objeto de tipo ejemplo
8     un_ejemplo = Ejemplo("un dato")
9     print(un_ejemplo) #El objeto se guarda en otra
10    # dirección de memoria.
11    print(un_ejemplo.atributo1)
12    print(un_ejemplo.atributo2)
13
14 main()
```

Shell ×

```
>>> %Run Objetos2.py
<__main__.Ejemplo object at 0x0434CC50>
un dato
100
```

# Atributos dinámicos

En Python, podemos crear atributos de manera dinámica, durante la ejecución.

```
1 class Ejemplo:
2     def __init__(self, parametro1):
3         self.atributo1 = parametro1
4         self.atributo2 = 100
5
6     def main():
7         # Se crea un objeto de tipo ejemplo
8         un_ejemplo = Ejemplo("un dato")
9         print(un_ejemplo) #El objeto se guarda en otra
10        # dirección de memoria.
11        print(un_ejemplo.atributo1)
12        print(un_ejemplo.atributo2)
13        print(un_ejemplo.atributo3)
14
15 main()
```

Shell

```
>>> %Run Objetos2.py
<__main__.Ejemplo object at 0x03D1CCF0>
un dato
100
Traceback (most recent call last):
  File "C:\Users\L00614578\OneDrive\AnaliticaDatos\Presentaciones\Objetos2.py", line 15, in <module>
    main()
  File "C:\Users\L00614578\OneDrive\AnaliticaDatos\Presentaciones\Objetos2.py", line 13, in main
    print(un_ejemplo.atributo3)
AttributeError: 'Ejemplo' object has no attribute 'atributo3'
```

Si intentamos imprimir el **atributo 3**, como no existe marca un error. Si deseo crear el **atributo 3**, solamente tengo que asignarle un valor.

# Atributos dinámicos

Si deseo crear el **atributo 3**, solamente tengo que asignarle un valor.

```
1 class Ejemplo:
2     def __init__(self, parametro1):
3         self.atributo1 = parametro1
4         self.atributo2 = 100
5
6     def main():
7         # Se crea un objeto de tipo ejemplo
8         un_ejemplo = Ejemplo("un dato")
9         print(un_ejemplo) #El objeto se guarda en otra
10        # dirección de memoria.
11        print(un_ejemplo.atributo1)
12        print(un_ejemplo.atributo2)
13        un_ejemplo.atributo3 = "Hola a todos"
14        print(un_ejemplo.atributo3)
15
16 main()
```

Shell ×

```
>>> %Run Objetos2.py
<__main__.Ejemplo object at 0x036CCDD0>
un dato
100
Hola a todos
```

# Ejemplo: Clase Gato

## Atributos:

Nombre  
Edad  
Alimentos favoritos

## Métodos:

`__init__()`  
`verEtapaDeVida()`  
`esAlimentoFavorito()`

```
class Gato:

    especie = "mamífero"

    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad
        self.alimentos = [] # Lista de alimentos favoritos

    def verEtapaDeVida(self):
        if self.edad > 1:
            print(self.nombre, "es adulto")
        else:
            print(self.nombre, "es cachorro")

    def esAlimentoFavorito(self, alimento):
        return alimento in self.alimentos
```

## Atributos:

- **Nombre** (string)
- **Edad** (Numérico entero)
- **Alimentos favoritos** (Lista)

## Métodos:

- **Constructor**
- **verEtapaDeVida** (saber si es adulto o cachorro dependiendo la edad)
- **esAlimentoFavorito** (saber si un alimento es favorito de este gato)

Cada que instanciemos esta clase podremos obtener objetos de tipo **Gato**. Cada objeto tendrá estos atributos y estos métodos.

# Atributos de clase y atributos de instancia

- Los **atributos de clase** son atributos compartidos por todas las instancias de esa clase.
- Los **atributos de instancia** son atributos únicos para cada uno de los objetos pertenecientes a dicha clase.

```
class Gato:  
  
    Atributos de clase        especie = "mamífero"  
  
    Atributos de instancia  
        def __init__(self, nombre, edad):  
            self.nombre = nombre  
            self.edad = edad  
            self.alimentos = [] # Lista de alimentos favoritos  
  
        def verEtapaDeVida(self):  
            if self.edad > 1:  
                print(self.nombre, "es adulto")  
            else:  
                print(self.nombre, "es cachorro")  
  
        def esAlimentoFavorito(self, alimento):  
            return alimento in self.alimentos
```

# Atributos de clase e instancia

- **Atributo de clase:** Le otorga el mismo valor a todos los objetos de esta clase. Se utiliza para valores que son constantes. Python no tiene una definición de constante (una variable cuyo valor no vamos a querer cambiar).
- **Atributos de instancia:** Los atributos dentro del constructor son atributos de instancia.

Para cada uno de los objetos de tipo **Gato** que instanciamos vamos a tener distintos valores dentro de cada uno de estos atributos. Un gato puede tener 3 años, otro 5, etc. Cada gato va a tener un estado distinto.

```
class Gato:  
    especie = "mamífero"  
  
    def __init__(self, nombre, edad):  
        self.nombre = nombre  
        self.edad = edad  
        self.alimentos = [] # Lista de alimentos favoritos  
  
    def verEtapaDeVida(self):  
        if self.edad > 1:  
            print(self.nombre, "es adulto")  
        else:  
            print(self.nombre, "es cachorro")  
  
    def esAlimentoFavorito(self, alimento):  
        return alimento in self.alimentos
```

Este atributo va a ser el mismo para todos los objetos de tipo **Gato** que instanciamos.

# Instanciar objetos

Instanciar objetos de tipo **Gato**.

```
1 class Gato:
2
3     especie = "mamífero"
4
5     def __init__(self, nombre, edad):
6         self.nombre = nombre
7         self.edad = edad
8         self.alimentos = [] # Lista vacía de alimentos favoritos
9
10    def main():
11        p = Gato() #Tengo que pasar los argumentos para el constructor sino
12            # Marca error
13    main()
14
15
16
17
18
19
20
21
22
```

Shell >

```
>>> %Run Clase_Gato.py
Traceback (most recent call last):
  File "C:\Users\L00614578\OneDrive\AnaliticaDatos\Presentaciones\Clase Gato.py", line
  22, in <module>
    main()
  File "C:\Users\L00614578\OneDrive\AnaliticaDatos\Presentaciones\Clase Gato.py", line
  20, in main
    p = Gato() #Tengo que pasar los argumentos para el constructor sino
TypeError: __init__() missing 2 required positional arguments: 'nombre' and 'edad'
```

# Instanciar objetos

Instanciar objetos de tipo **Gato**.

```
1 class Gato:  
2  
3     especie = "mamífero"  
4  
5     def __init__(self, nombre, edad):  
6         self.nombre = nombre  
7         self.edad = edad  
8         self.alimentos = [] # Lista vacía de alimentos favoritos  
9  
19    def main():  
20        p = Gato("Misifu", 1)  
21        print(p.nombre)  
22        print(p.edad)  
23  
24    main()
```

Shell <

```
>>> %Run Clase_Gato.py
```

Misifu

1

# Atributos dinámicos

Python permite agregar atributos de forma dinámica.

```
1 class Gato:  
2  
3     especie = "mamífero"  
4  
5     def __init__(self, nombre, edad):  
6         self.nombre = nombre  
7         self.edad = edad  
8         self.alimentos = [] # Lista vacía de alimentos favoritos  
9  
19    def main():  
20        p = Gato("Misifu", 1)  
21        print(p.nombre)  
22        print(p.edad)  
23        p.raza = "angora"  
24        print(p.raza)  
25  
26    main()
```

```
Shell x  
=>>> %Run Clase_Gato.py  
Misifu  
1  
angora
```

Ejemplo: **raza** no es atributo de la clase **Gato**, pero si le asigno un valor se crea ese atributo para ese objeto **p**. Este objeto **Gato** cuya variable se llama **p** va a tener la raza que le estamos dando.

# Atributos dinámicos

```
1 class Gato:  
2  
3     especie = "mamífero"  
4  
5     def __init__(self, nombre, edad):  
6         self.nombre = nombre  
7         self.edad = edad  
8         self.alimentos = [] # Lista vacía de alimentos favoritos  
9  
10    def main():  
11        p = Gato("Misifu", 1)  
12        print(p.nombre)  
13        print(p.edad)  
14        p.raza = "angora"  
15        print(p.raza)  
16        q = Gato("Pelos", 3)  
17        print(q.nombre)  
18        print(q.raza)  
19  
20    main()
```

```
Shell > ./ejercicios-clase_04.py  
Misifu  
1  
angora  
Pelos  
Traceback (most recent call last):  
  File "C:\Users\L00614578\OneDrive\AnaliticaDatos\Presentaciones\Clase Gato.py", line 29, in <module>  
    main()  
  File "C:\Users\L00614578\OneDrive\AnaliticaDatos\Presentaciones\Clase Gato.py", line 27, in main  
    print(q.raza)  
AttributeError: 'Gato' object has no attribute 'raza'
```

- Si creamos un nuevo objeto de tipo **Gato** y le doy sus propios valores. Si intentamos acceder al atributo **raza** me va a decir que no existe, ya que para cada uno de los atributos lo que hace es guardar un par clave y nos va a marcar error.
- Para el objeto **p**, vamos a tener un diccionario con los atributos: **nombre, edad, alimentos y raza**.
- En cambio el objeto **q**, solamente va a tener un diccionario con los atributos por **default**, los que están dentro del constructor, ya que no le hemos agregado atributos de forma dinámica.

# Ejemplo: Clase Gato

Si le queremos agregar al objeto **p** sus alimentos favoritos.

```
1 class Gato:  
2  
3     especie = "mamífero"  
4  
5     def __init__(self, nombre, edad):  
6         self.nombre = nombre  
7         self.edad = edad  
8         self.alimentos = [] # Lista vacía de alimentos favoritos  
9  
30    p.alimentos.append("pescado") #Agregamos un string a una lista  
31    p.alimentos.append("leche")  
32    print(p.alimentos)  
33  
34 main()
```

# Ejemplo: Clase Gato

Si queremos agregar una lista nueva de alimentos favoritos y borrar la lista anterior.

```
1 class Gato:  
2  
3     especie = "mamífero"  
4  
5     def __init__(self, nombre, edad):  
6         self.nombre = nombre  
7         self.edad = edad  
8         self.alimentos = [] # Lista vacía de alimentos favoritos  
9  
30    p.alimentos.append("pescado") #Agregamos un string a una lista  
31    p.alimentos.append("leche")  
32    print(p.alimentos)  
33    p.alimentos = ["galletas", "leche", "queso"]  
34    print(p.alimentos)  
35  
36 main()  
  
Shell ×  
['pescado', 'leche']  
['galletas', 'leche', 'queso']
```

# Ejemplo: Clase Gato

```
class Gato:

    especie = "mamífero"

    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad
        self.alimentos = [] # Lista de alimentos favoritos

    def verEtapaDeVida(self):
        if self.edad > 1:
            print(self.nombre, "es adulto")
        else:
            print(self.nombre, "es cachorro")

    def esAlimentoFavorito(self, alimento):
        return alimento in self.alimentos
```

Método **verEtapaDeVida**

```
19 def main():
20     # Vamos a instanciar nuestros objetos
21     p = Gato("Misifu", 1)
22     p.verEtapaDeVida() # Este método no tiene un parámetro.
23
24 main()
```

Shell ×  
run\_clase\_udemy\_ejemplos.py

Misifu es cachorro

# Ejemplo: Clase Gato

```
class Gato:

    especie = "mamífero"

    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad
        self.alimentos = [] # Lista de alimentos favoritos

    def verEtapaDeVida(self):
        if self.edad > 1:
            print(self.nombre, "es adulto")
        else:
            print(self.nombre, "es cachorro")

    def esAlimentoFavorito(self, alimento):
        return alimento in self.alimentos

19   def main():
20       # Vamos a instanciar nuestros objetos
21       p = Gato("Misifu", 1)
22       p.verEtapaDeVida() # Este método no tiene un parámetro.
23       p.edad = 5
24       p.verEtapaDeVida()

25
26 main()
```

Shell ×

```
Misifu es cachorro
Misifu es adulto
```

Método **verEtapaDeVida**  
Si cambio la edad cambia  
el resultado del método.

# Ejemplo: Clase Gato

```
class Gato:

    especie = "mamífero"

    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad
        self.alimentos = [] # Lista de alimentos favoritos

    def verEtapaDeVida(self):
        if self.edad > 1:
            print(self.nombre, "es adulto")
        else:
            print(self.nombre, "es cachorro")

    def esAlimentoFavorito(self, alimento):
        return alimento in self.alimentos

19 def main():
20     # Vamos a instanciar nuestros objetos
21     p = Gato("Misifu", 1)
22     p.verEtapaDeVida() # Este método no tiene un parámetro.
23     p.edad = 5
24     p.verEtapaDeVida()
25     q = Gato("Leo", 3)
26     p.alimentos = ["Leche", "Pan", "Huevo"]
27     q.alimentos = ["Galletas", "Cereal"]
28     res = p.esAlimentoFavorito("Pan")
29     print(res)
30     res = q.esAlimentoFavorito("Pan")
31     print(res)
```

Shell

```
True
False
```

Checar alimentos  
favoritos.

# Atributos de clase

```
class Gato:  
  
    especie = "mamífero" ←  
  
    def __init__(self, nombre, edad):  
        self.nombre = nombre  
        self.edad = edad  
        self.alimentos = [] # Lista de alimentos favoritos  
  
    def verEtapaDeVida(self):  
        if self.edad > 1:  
            print(self.nombre, "es adulto")  
        else:  
            print(self.nombre, "es cachorro")  
  
    def esAlimentoFavorito(self, alimento):  
        return alimento in self.alimentos  
  
19 def main():  
20     # Vamos a instanciar nuestros objetos  
21     p = Gato("Misifu", 1)  
22     q = Gato("Geo", 3)  
23     print(p.especie)  
24     print(q.especie)  
25     print(Gato.especie)  
26  
27 main()
```

Shell <

```
mamífero  
mamífero  
mamífero
```

## Atributos de clase:

Este atributo va a ser común para todos los objetos que yo instancie (**especie**).

- Si pregunto la especie de **p**, me va a decir que es **mamífero**.
- Si pregunto la especie de **q**, me va a decir que es **mamífero**.
- Ambos tienen el mismo valor.

**Los atributos de clase se pueden utilizar sin necesidad de una instancia.**

- Poniendo el nombre de la **clase.atributo** de clase me va a dar el valor.

**El atributo de clase es un atributo estático, no es necesario un objeto para poder obtenerlo.**

# Atributos de clase

**Atributos de clase o estáticos.** Si queremos utilizar estos atributos no necesito crear instancias de la clase para accederlos.

## Nombre de la clase. Nombre del atributo de clase

Este diccionario es compartido por todas las instancias de esta clase. Tenemos un solo diccionario con los **atributos de clase**.

Para la clase **Gato** se crea un diccionario donde se van a guardar los atributos y valores de clase.

The diagram illustrates the structure of class attributes for the **Gato** class. It shows three separate dictionaries:

- ATRIBUTOS DE CLASE GATO:** A dictionary where the key is "especie" and the value is "mamífero".
- ATRIBUTOS DEL OBJETO p:** A dictionary with attributes: nombre ("Pelusa"), edad (1), alimentos (["leche", "galletas", "arroz"]), and raza ("Siamés").
- ATRIBUTOS DEL OBJETO g:** A dictionary with attributes: nombre ("Cleo"), edad (3), and alimentos (["pan", "pescado"]).

A red arrow points from the "dinámico" label to the "alimentos" attribute in the object p's dictionary, indicating that this attribute is dynamically added to the class dictionary.

atributo	valor
especie	"mamífero"

atributo	valor
nombre	"Pelusa"
edad	1
alimentos	["leche", "galletas", "arroz"]
raza	"Siamés"

atributo	valor
nombre	"Cleo"
edad	3
alimentos	["pan", "pescado"]

Vamos a tener  
diccionarios propios para  
cada uno de los objetos.

- En el objeto **p**, se creó  
un atributo dinámico.

Cada objeto va a tener  
todos los **atributos de  
instancia** que estén en  
la clase, más los  
**atributos dinámicos** que  
se agreguen en la  
ejecución.

# Método mostrar

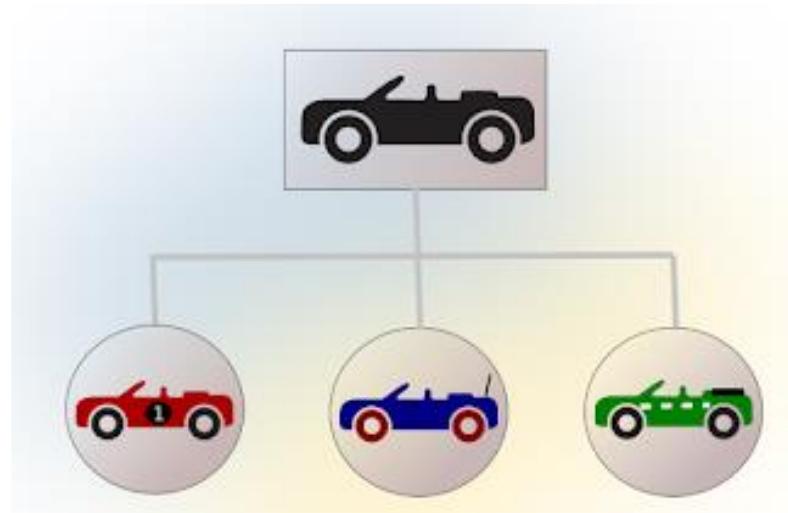
Crea el método **mostrar** que regrese en un string la concatenación de los **atributos de instancia** (nombre, edad y alimentos) y el **atributo de clase** (especie).

```
18
19     def mostrar(self):
20         return "Nombre: " + self.nombre + "\t Edad: " + str(self.edad) + " Alimentos favoritos: " + str(self.alimentos) + "\t Especie: " + Gato.especie
21
22 def main():
23     # Vamos a instanciar nuestros objetos
24     p = Gato("Misifu", 1)
25     p.verEtapaDeVida() # Este método no tiene un parámetro.
26     p.edad = 5
27     p.verEtapaDeVida()
28     q = Gato("Leo", 3)
29     p.alimentos = ["Leche", "Pan", "Huevo"]
30     q.alimentos = ["Galletas", "Cereal"]
31     res = p.esAlimentoFavorito("Pan")
32     print(res)
33     res = q.esAlimentoFavorito("Pan")
34     print(res)
35     print(p.mostrar())
36     print(q.mostrar())
37
38 main()
```

```
Shell >>> %run Clase_GatoMetodos.py
Misifu es cachorro
Misifu es adulto
True
False
Nombre: Misifu    Edad: 5 Alimentos favoritos: ['Leche', 'Pan', 'Huevo']   Especie: Mamífero
Nombre: Leo        Edad: 3 Alimentos favoritos: ['Galletas', 'Cereal']      Especie: Mamífero
```

## Actividad individual

Programación  
orientada a objetos

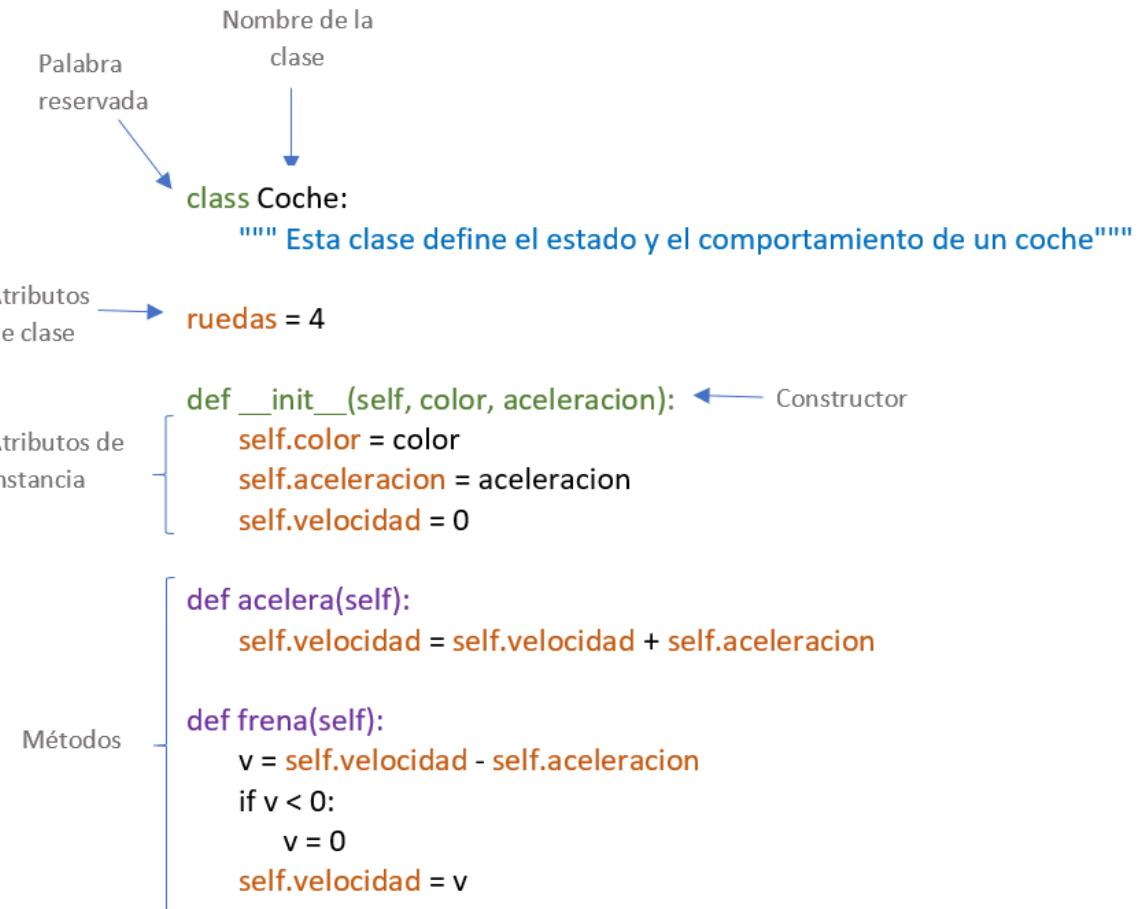


# Clases en Python

Cada vez que se define una clase en Python, se crea a su vez un tipo nuevo (tipo **int**, **float**, **str**, **list**, **tuple**...)

Para definir una clase en Python se utiliza la palabra reservada **class**.

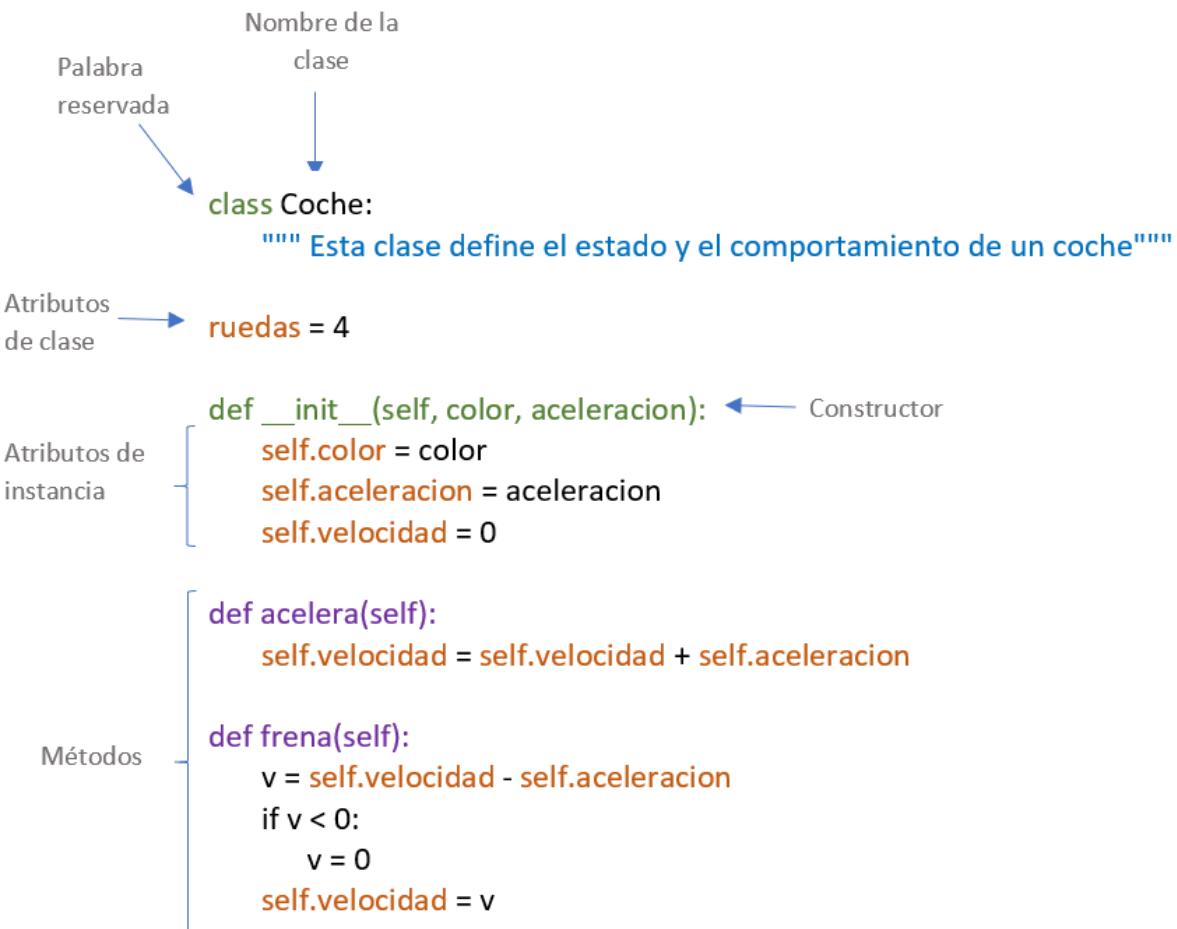
El siguiente esquema visualiza los elementos principales que componen una clase.



# Clases en Python

Este esquema se define la clase **Coche**.

La clase establece una serie de **datos** (atributos), como **ruedas, color, aceleracion** y **velocidad** y las **operaciones** (métodos) **acelera()** y **frena()**.



# Objetos en Python

En el siguiente ejemplo se crean dos objetos de tipo Coche:

**c1 = Coche('rojo', 20)**

print(c1.color)

rojo

print(c1.ruedas)

4

**c2 = Coche('azul', 30)**

print(c2.color)

azul

print(c2.ruedas)

4

```
class Coche:  
    """ Esta clase define el estado y el comportamiento de un coche"""
```

```
ruedas = 4
```

```
def __init__(self, color, aceleracion):  
    self.color = color  
    self.aceleracion = aceleracion  
    self.velocidad = 0
```

```
def acelera(self):  
    self.velocidad = self.velocidad + self.aceleracion
```

```
def frena(self):  
    v = self.velocidad - self.aceleracion  
    if v < 0:  
        v = 0  
    self.velocidad = v
```

**NOTA:**  
**En la función main,**  
**crea los objetos c1 y c2.**  
**Imprime los atributos**  
**color y ruedas de cada**  
**objeto.**

**c1** y **c2** son objetos, objetos cuya clase es **Coche**. Ambos objetos pueden **acelerar** y **frenar**, porque su clase define estas operaciones y tienen un color, porque la clase **Coche** también define este dato. **c1** es de color rojo, mientras que **c2** es de color azul.

# Objetos en Python

Para crear un objeto de una clase determinada, es decir, instanciar una clase, se usa el nombre de la clase y a continuación se añaden paréntesis (como si se llamara a una función).

**obj = MiClase()**

Este código crea una nueva instancia de la clase **MiClase** y asigna dicho objeto a la variable **obj**. Esto crea un objeto vacío, sin estado.

# Constructor de una clase

- Sin embargo, hay clases que deben o necesitan crear instancias de objetos con un estado inicial.
- Esto se consigue implementando el método especial `__init__()`. Este método es conocido como el constructor de la clase y se invoca cada vez que se instancia un nuevo objeto.
- El método `__init__()` establece un primer parámetro especial que se suele llamar `self` y puede especificar otros parámetros como en las funciones.
- En nuestro caso, el constructor de la clase `Coche` es el siguiente:

```
def __init__(self, color, aceleracion):  
    self.color = color  
    self.aceleracion = aceleracion  
    self.velocidad = 0
```

- Además del parámetro `self`, se definen los parámetros `color` y `aceleracion`, que determinan el estado inicial del objeto de tipo `Coche`.

# Constructor de una clase

En este caso, para instanciar un objeto de tipo **Coche**, debemos pasar como argumentos el **color** y la **aceleración** como vimos en el ejemplo:

```
c1 = Coche('rojo', 20)
```

```
class Coche:  
    """ Esta clase define el estado y el comportamiento de un coche """  
  
    ruedas = 4  
  
    def __init__(self, color, aceleracion):  
        self.color = color  
        self.aceleracion = aceleracion  
        self.velocidad = 0
```

# Atributos de datos y métodos

La única operación que pueden realizar los objetos es referenciar a sus atributos por medio del operador (.) punto.

Un objeto tiene dos tipos de atributos: **atributos de datos y métodos**.

- Los **atributos de datos**

definen el estado del  
objeto. En otros

lenguajes son conocidos  
simplemente como  
atributos.

- Los **métodos** son las  
funciones definidas  
dentro de la clase.

Atributos de  
instancia

```
def __init__(self, color, aceleracion): ← Constructor
    self.color = color
    self.aceleracion = aceleracion
    self.velocidad = 0
```

Métodos

```
def acelera(self):
    self.velocidad = self.velocidad + self.aceleracion

def frena(self):
    v = self.velocidad - self.aceleracion
    if v < 0:
        v = 0
    self.velocidad = v
```

# Atributos de datos y métodos

Siguiendo con nuestro ejemplo de la clase `Coche`, vamos a crear el objeto `c1`:

```
1.  >>> c1 = Coche('rojo', 20)
2.  >>> print(c1.color)
3.  rojo
4.  >>> print(c1.velocidad)
5.  0
6.  >>> c1.acelera()
7.  >>> print(c1.velocidad)
8.  20
```

```
def __init__(self, color, aceleracion): ← Constructor
    self.color = color
    self.aceleracion = aceleracion
    self.velocidad = 0

def acelera(self):
    self.velocidad = self.velocidad + self.aceleracion

def frena(self):
    v = self.velocidad - self.aceleracion
    if v < 0:
        v = 0
    self.velocidad = v
```

**NOTA:**  
Imprime el atributo velocidad antes y después de llamar al método acelera del objeto c1.

En la línea 2, el objeto `c1` está referenciando al atributo de dato `color` y en la línea 4 al atributo `velocidad`. Sin embargo, en la línea 6 se referencia al método `acelera()`. Al llamar a este método se modifica el estado del objeto, dado que se incrementa su velocidad. Este hecho lo puedes apreciar cuando se vuelve a referenciar al atributo `velocidad` en la línea 7.

# Atributos dinámicos

A diferencia de otros lenguajes, los atributos de datos **dinámicos** no necesitan ser declarados previamente. Un objeto los crea del mismo modo en que se crean las variables en Python, es decir, cuando les asigna un valor por primera vez.

El siguiente código es un ejemplo de ello:

```
1.  >>> c1 = Coche('rojo', 20)
2.  >>> c2 = Coche('azul', 10)
3.  >>> print(c1.color)
4.  rojo
5.  >>> print(c2.color)
6.  azul
7.  >>> c1.marchas = 6
8.  >>> print(c1.marchas)
9.  6
10. >>> print(c2.marchas)
11. Traceback (most recent call last):
12.   File "<input>", line 1, in <module>
13.     AttributeError: 'Coche' object has no attribute 'marchas'
```

**NOTA:**  
**Crea el atributo dinámico marchas en el objeto c1 con el valor de 6 y realiza las siguientes pruebas en el objeto c1 y c2.**

# Atributos de clase y atributos de instancia

- Los **atributos de clase** son atributos compartidos por todas las instancias de esa clase.
- Los **atributos de instancia** son atributos únicos para cada uno de los objetos pertenecientes a dicha clase.

```
class Coche:
```

```
    """ Esta clase define el estado y el comportamiento de un coche"""
```

Atributos  
de clase →

```
    ruedas = 4
```

Atributos de  
instancia {

```
    def __init__(self, color, aceleracion): ← Constructor  
        self.color = color  
        self.aceleracion = aceleracion  
        self.velocidad = 0
```

# Atributos de clase y atributos de instancia

- Para referenciar a un **atributo de clase** se utiliza el nombre de la clase.
- Al modificar un atributo de este tipo, los cambios se verán reflejados en todas y cada una las instancias.

```
>>> c1 = Coche('rojo', 20)
>>> c2 = Coche('azul', 20)
>>> print(c1.ruedas)
4
>>> print(c2.ruedas)
4
>>> Coche.ruedas = 6
>>> print(c1.ruedas)
6
>>> print(c2.ruedas)
6
```

## NOTA:

- Imprime las ruedas de los objetos c1 y c2.
- Modifica el atributo de clase ruedas e imprime nuevamente las ruedas de los objetos c1 y c2.

```
class Coche:
    """ Esta clase define el estado y el comportamiento de un coche """
    Atributos de clase → ruedas = 4

    Atributos de instancia {
        def __init__(self, color, aceleracion):
            self.color = color
            self.aceleracion = aceleracion
            self.velocidad = 0
```

# Método mostrar

Crea el método **mostrar** que imprime un string con la concatenación de los **atributos de instancia** (color, velocidad y aceleracion) y el **atributo de clase** (ruedas).

```
class Coche:  
    """ Esta clase define el estado y el comportamiento de un coche"""  
  
    ruedas = 4  
  
    def __init__(self, color, aceleracion):  
        self.color = color  
        self.aceleracion = aceleracion  
        self.velocidad = 0  
  
    def acelera(self):  
        self.velocidad = self.velocidad + self.aceleracion  
  
    def frena(self):  
        v = self.velocidad - self.aceleracion  
        if v < 0:  
            v = 0  
        self.velocidad = v
```

## NOTA:

- Crea el método **mostrar**.
- En el main, imprime el resultado del método **mostrar** para el objeto **c1** y **c2**.



# Gracias

