

# **PROGRAMACIÓN III**

## **Clase 5**

# Clases y objetos



## Clase 4

### Funciones

- ✓ **Funciones. Concepto.**
- ✓ **Llamada a función.**
- ✓ **Retorno y envío de valores.**
- ✓ **Parámetros, argumentos, valor y referencia.**
- ✓ **Parámetros mutables e inmutables.**
- ✓ **Parámetros por defecto**
- ✓ **Docstring.**
- ✓ **Funciones Lambda/Anónima.**

## Clase 5

### Clases y objetos

- ✓ Paradigmas de programación.
- ✓ Programación estructurada vs POO.
- ✓ Clases, objetos y atributos.
- ✓ Métodos de clase y métodos especiales: init, del y str.

## Clase 6

- ✓ **Colaboración entre clases y Encapsulamiento**
- ✓ Mensajes y Métodos.
- ✓ Colaboración entre clases.
- ✓ Variables de clase.
- ✓ Objetos dentro de objetos.
- ✓ Encapsular atributos y métodos.
- ✓ Decorators.

# Programación orientada a objetos

En el **paradigma de programación orientada a objetos (POO)** se utilizan entidades que representan elementos del problema a resolver y tienen **atributos** y **comportamientos** (pueden almacenar datos y realizar acciones).

Estas entidades se denominan objetos, y Python proporciona soporte para este paradigma.

La POO permite que el desarrollo de grandes proyectos de software sea más fácil e intuitivo, al representar en el software objetos del mundo real y sus relaciones.

La programación orientada a objetos surge en los 70s y tiene un gran auge en los 90. Uno de los lenguajes destacados de este nuevo paradigma es Java.

Por supuesto, el concepto de la POO excede a Java y Python ya que se aplica a muchos lenguajes..

# Objetos y clases

A diferencia de la programación estructurada o funcional, que está centrada en las funciones, la **programación orientada (POO)** se basa en la definición de **clases** y **objetos**.

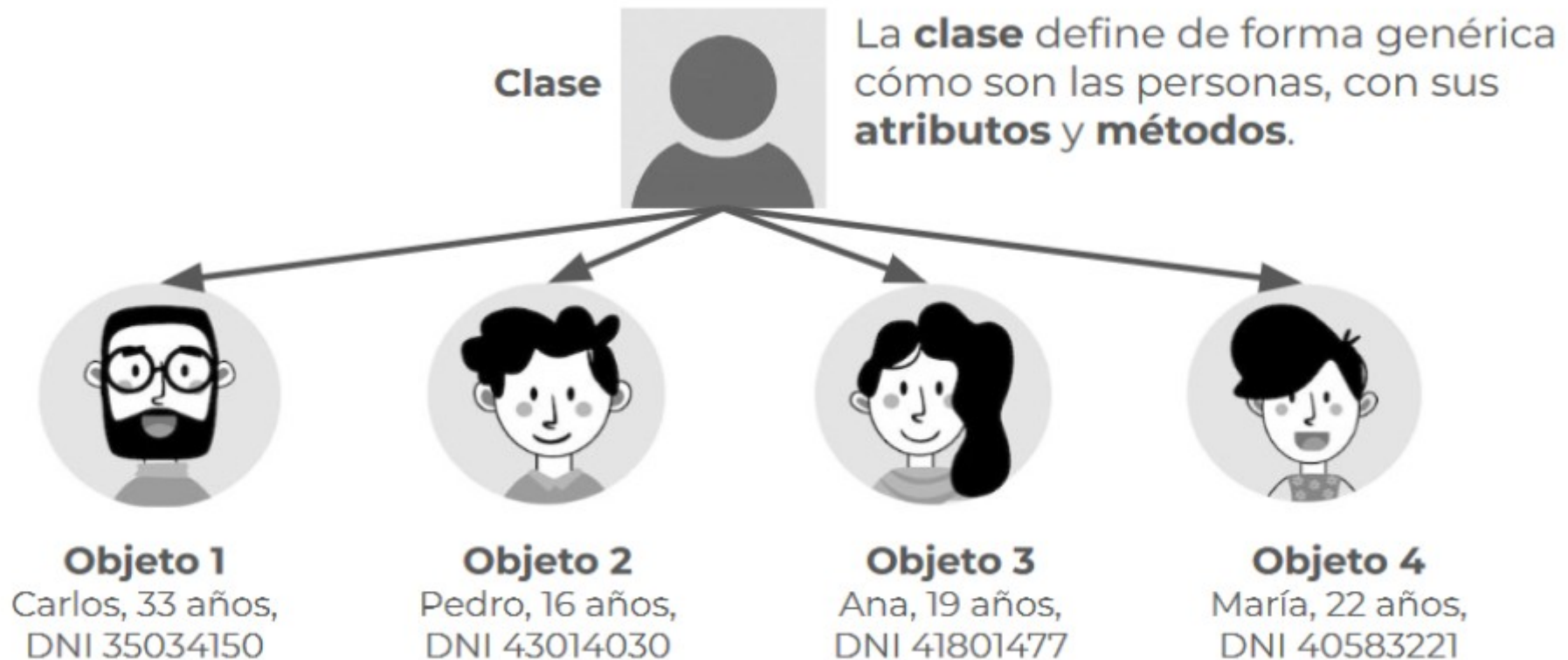
Podemos pensar en las clases como plantillas. Definen de manera genérica cómo van a ser los objetos de determinado tipo. Por ejemplo, una clase para representar a las personas puede llamarse Persona y tener una serie de **propiedades** como Nombre, Edad o Nro de DNI (similares a variables), y una serie de **comportamientos**, como Hablar(), Caminar() o Comer(). Estos comportamientos se implementan como **métodos** (similares a funciones).

# Objetos y clases

Una clase no es más que un concepto, sin entidad real. Para poder utilizarlas en un programa hay que **instanciarla**, es decir, **crear un nuevo objeto concreto** de la misma. Un objeto es una entidad concreta que se crea a partir de la plantilla que es la **clase**. Este nuevo objeto tiene "existencia" real, puesto que ocupa memoria y se puede utilizar en el programa. Así un objeto puede ser una persona que se llama Ivana, de 37 años y DNI nro 32456822, que en nuestro programa podría hablar, caminar o comer, que son los comportamientos que están definidos en la **clase**.

Una **clase** equivale a la **generalización de un tipo específico de objetos**. Y una **instancia** es la concreción de una clase en un objeto.

# OBJETOS Y CLASES



Los **objetos** son personas concretas, cada una con sus características propias.

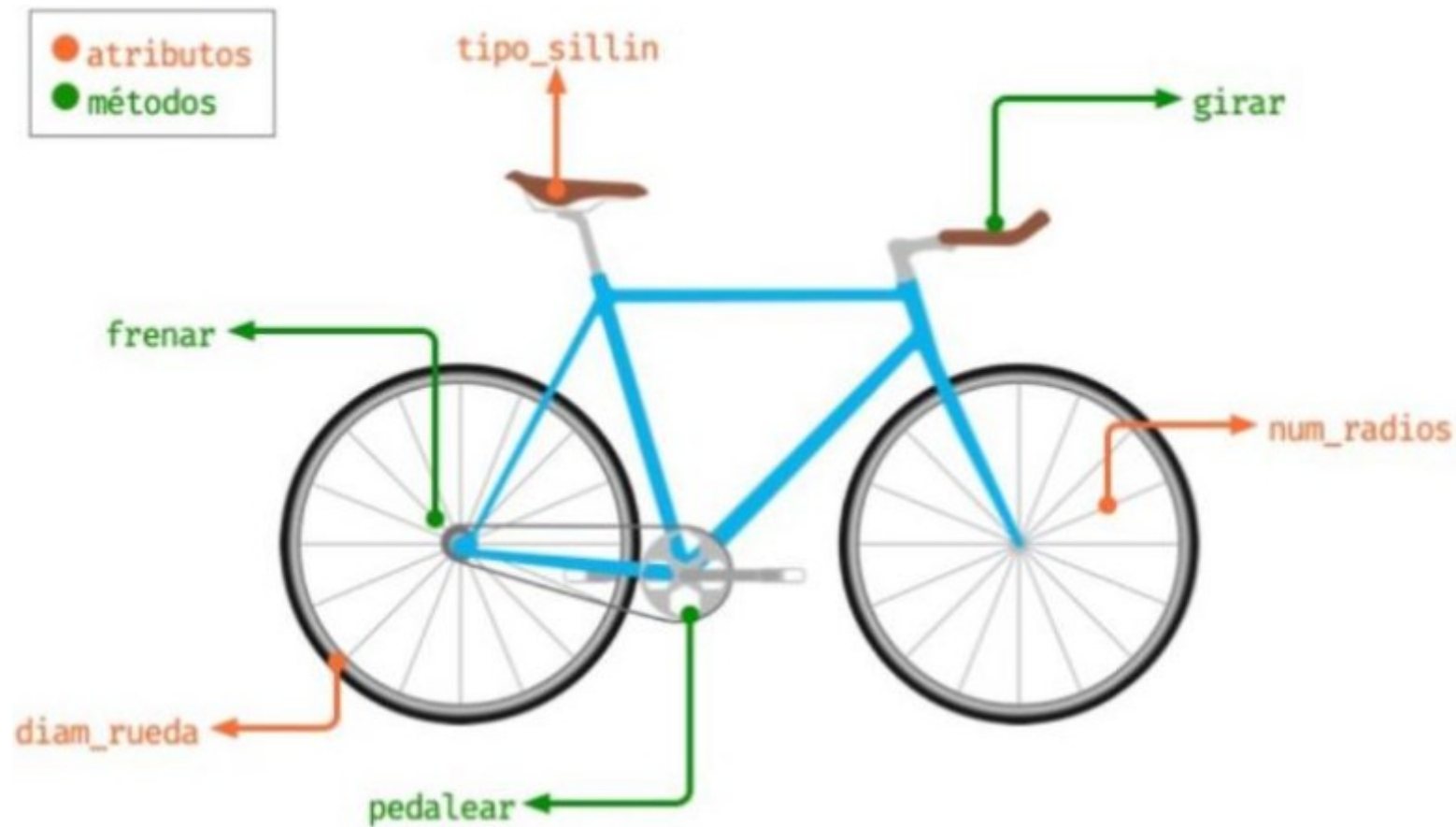
# Objetos y clases

Conceptos relacionados con clases y objetos:

- ✓ **Atributos:** Son datos que caracterizan al objeto, almacenan datos relacionados con su estado.
- ✓ **Métodos:** Caracterizan el comportamiento del objeto. Son las acciones que el objeto puede realizar por sí mismo, como responder a solicitudes externas o actuar sobre otros objetos. Pueden depender de, o modificar los valores de un atributo.
- ✓ **Identidad:** Cada objeto tiene una identidad que lo distingue de otros objetos, sin considerar su estado. Por lo general, esta identidad se crea mediante un identificador que deriva naturalmente de un problema (por ejemplo: un producto puede estar representado por un código, un automóvil por un número de modelo, etc.).



# Objetos | Atributos y métodos - Ejemplo



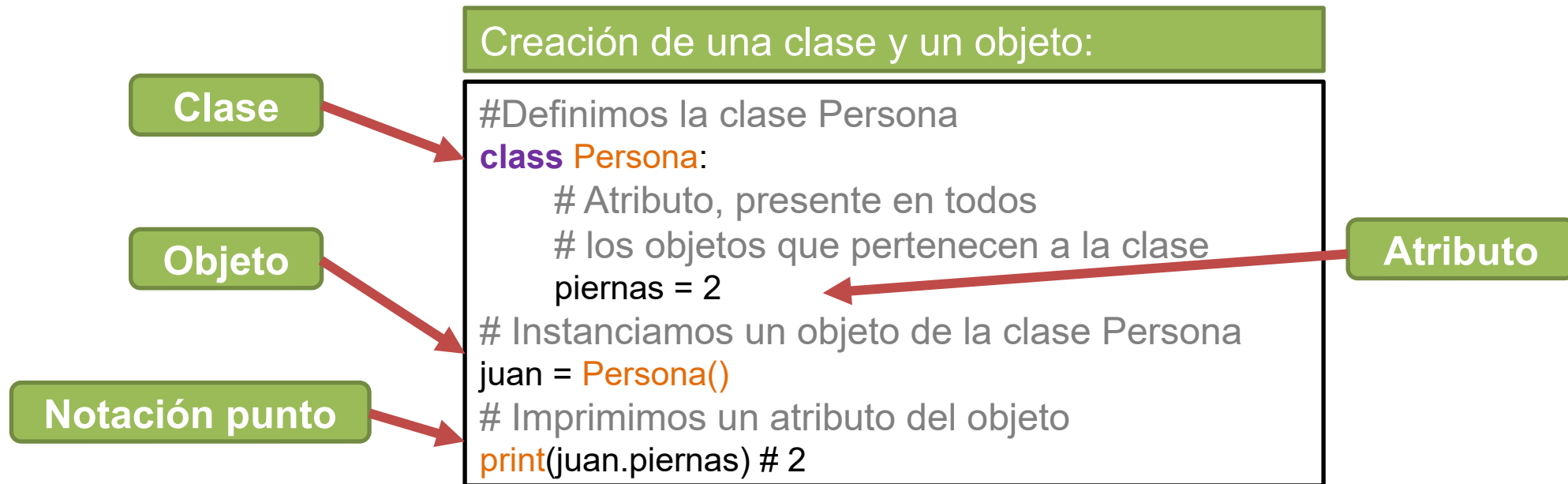
# Objetos | Atributos y métodos – Ejemplo

Los objetos de la clase Bicicleta comparten atributos y métodos:

- ✓ Los **atributos** tipo\_sillin, num\_radios y diam\_rueda están presentes en todos los objetos de la clase, pero posiblemente sus valores varían de un objeto Bicicleta a otro.
- ✓ Los **métodos** frenar, girar y pedalear son compartidos por todas las instancias que se crean a partir de la clase bicicleta. Pero en cada instancia se invocan cuando sea necesario: no todas las instancias van a frenar o acelerar al mismo tiempo.  
Resumiendo, estas características están presentes en todas las bicicletas creadas a partir de la “plantilla” Bicicleta.

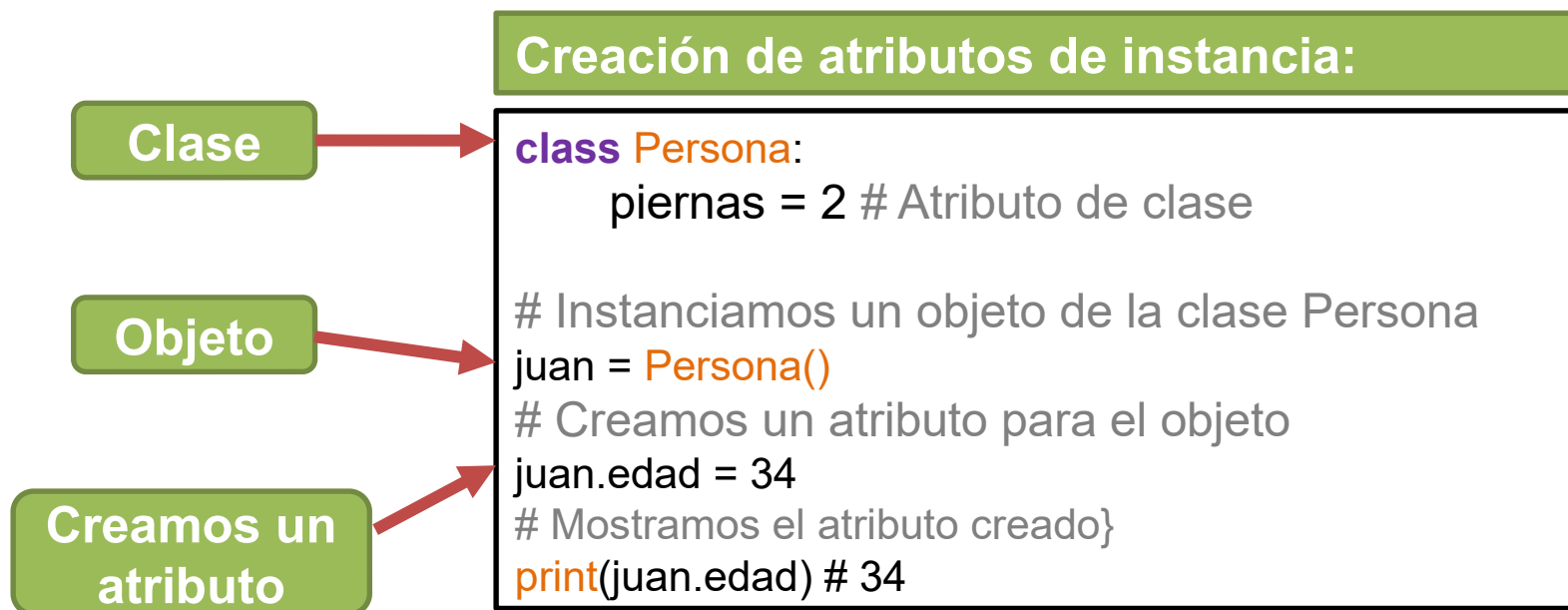
# Clases | Definición

Los nombres de las clases se escriben camelCase. Se definen con la palabra clave **class**, seguida del nombre de la clase y dos puntos. Los objetos se declaran como variables, y se accede a sus atributos utilizando la notación punto.



# Clases | Definición

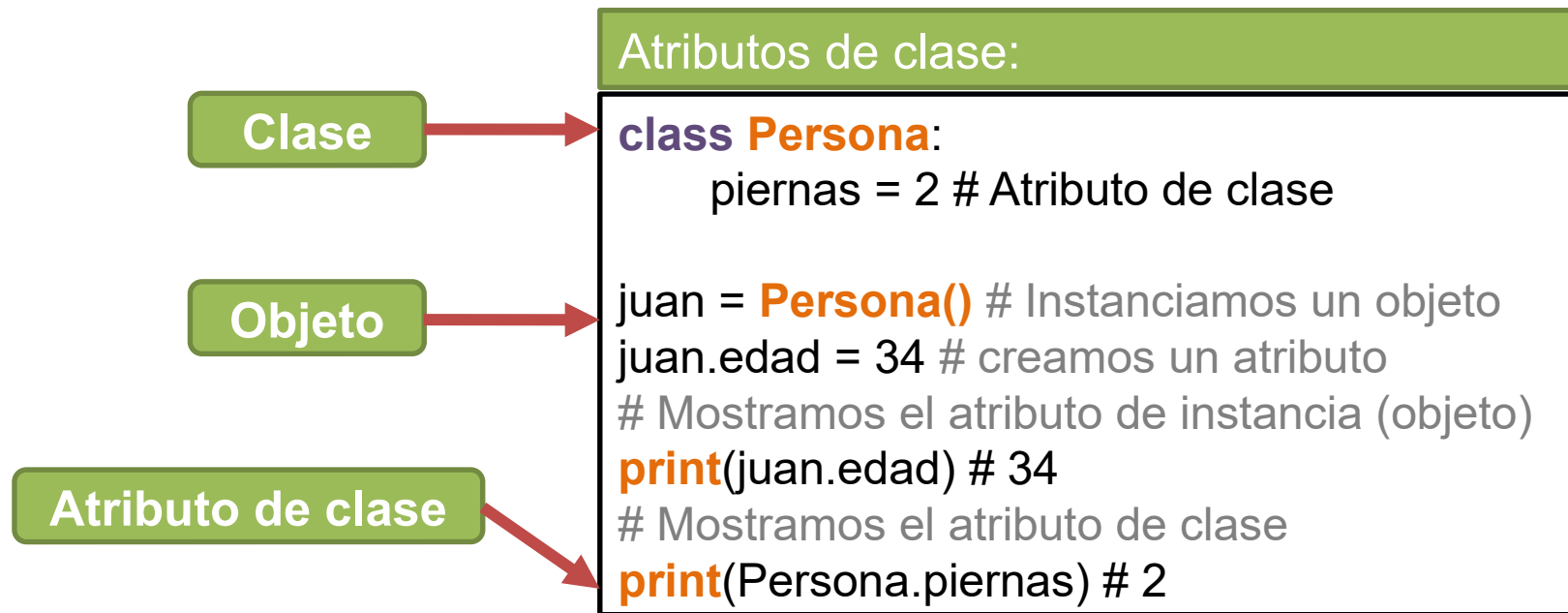
Los objetos pueden tener sus propios atributos, llamados **atributos de instancia**. Una manera de crearlos es usar directamente la notación punto:



Pero veremos que existe una manera más ordenada de hacer esto.

# Clases | Atributos de clase

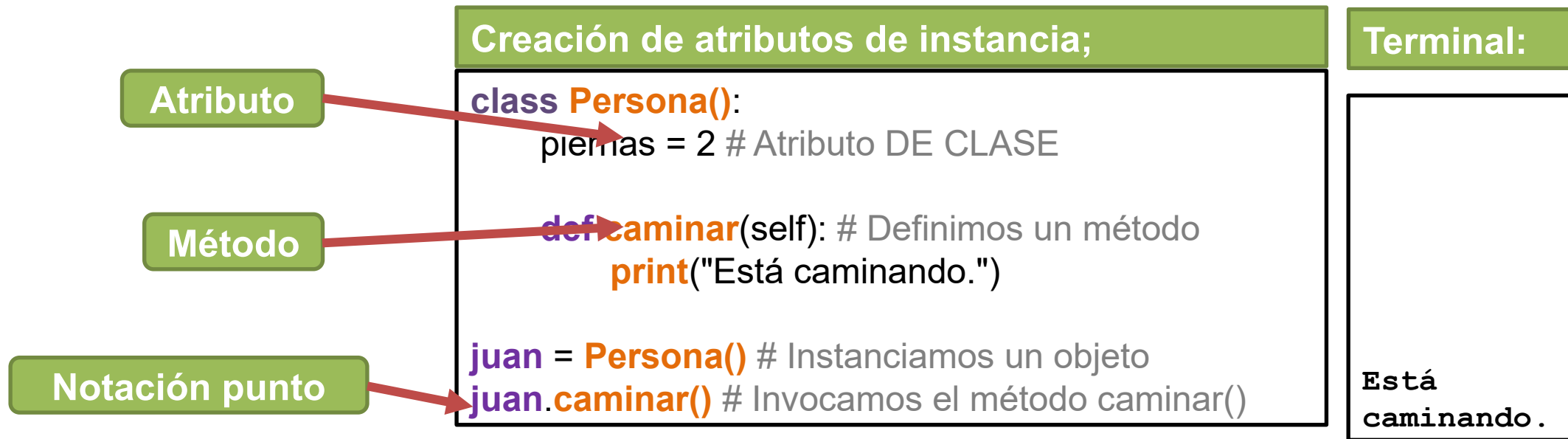
Las variables dentro de la clase (**atributos de clase**) son compartidas por todos los objetos instanciados. Se definen dentro de la clase pero fuera de sus métodos:



Podemos acceder a ellos mediante la notación punto: clase.atributo

# Clases | Métodos

Los **métodos** permiten a los objetos de una clase realizar acciones. Se declaran con **def:**, como las funciones, dentro de la clase. Reciben parámetros y uno de ellos, el primero (**self**) es obligatorio:



**self** hace referencia a la instancia perteneciente a la clase.

# Clases | Métodos

En el ejemplo los **métodos** caminar() y detener() modifican el valor del **atributo** caminando. Necesitamos utilizar **self** para referirnos al atributo del objeto, ya que **self** hace referencia a la instancia del objeto. Si creamos varios objetos de la misma clase, cada uno posee su propio valor en el atributo caminando.

Atributos de clase:

```
class Persona():  
    def caminar(self): # Método caminar  
        self.caminando = True # Atributo  
        print("Estoy caminando.")  
  
    def detener(self): # Método detener  
        self.caminando = False  
        print("Estoy detenido.")  
  
juan = Persona() # Instanciamos  
juan.caminar() # Estoy caminando  
print(juan.caminando) # True  
juan.detener() # Estoy detenido  
print(juan.caminando) # False
```

# Clases | Método constructor

Un **constructor** es un **método** que permite a la objetos asignar valores a los atributos. Su primer parámetro es **self**, y los demás los requeridos para la inicialización.

Luego de **instanciar** el objeto, establecemos los valores de sus **atributos** mediante el constructor, y ya podemos utilizarlos normalmente.

Veamos un ejemplo:

## Constructor:

```
class Persona():
    # Método constructor
    def constructor(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad
    def identificarse(self): # Método normal
        print(f" Hola. Soy {self.nombre} y tengo {self.edad} años.")

persona1 = Persona() # Instanciamos
persona1.constructor("Juan", 42)
persona1.identificarse()
persona1.edad = 43 # Modificamos la edad
persona1.identificarse()
```



# Clases | Método constructor

En el ejemplo anterior tenemos una clase con **dos métodos** y **dos atributos** (nombre y edad). El valor de los atributos para cada objeto se establecen luego de la instancia, mediante el constructor. El método constructor puede tener cualquier nombre.

Es muy importante el uso de **self**. El constructor crea los atributos, cuyo nombre comienza por **self** y copia en ellos los valores pasados mediante los parámetros. Los atributos y los parámetros suelen tener el mismo nombre, pero esto no es obligatorio.

Python ofrece un **método especial** denominado **`__init__()`** que simplifica el proceso de instancia y asignación de valores a los atributos.

# Clases | Método `__init__()`

`__init__()` permite que en el momento de la instancia se asignen valores a los atributos.

El ejemplo de la derecha muestra cómo utilizar `__init__()`. En el momento de instanciar el objeto `persona1` pasamos como argumentos los valores del nombre y la edad, para que el constructor los asigne a la instancia creada.

Atributos de clase:

```
class Persona():  
    # Método constructor  
    def __init__(self, nombre, edad):  
        self.nombre = nombre  
        self.edad = edad  
    def identificarse(self): # Método normal  
        print(f" Hola. Soy {self.nombre} y tengo {self.edad} años.")  
  
# Instanciamos  
persona1 = Persona("Juan", 42)  
persona1.identificarse()  
persona1.edad = 43 # Modificamos la edad  
persona1.identificarse()
```

# Clases | Ejemplo

La clase **Cuadrado** incluye un constructor que en la inicialización establece el valor del **atributo** lado.

Posee métodos para calcular el área y el perímetro ( `calcular_area()` y `calcular_perimetro()`). Es posible modificar el valor del **atributo** lado mediante la notación punto, y los **métodos** mencionados devuelven los valores recalculados

## Ejemplo de la clase Cuadrado

```
class Cuadrado:
    def __init__(self, lado):
        self.lado = lado

    def calcular_area(self):
        return self.lado ** 2

    def calcular_perimetro(self):
        return self.lado * 4

cuad1 = Cuadrado(15) # Instanciamos
print(cuad1.calcular_area()) # 60
print(cuad1.calcular_perimetro()) # 225
cuad1.lado = 12 # Modificamos el atributo
print(cuad1.calcular_area()) # 48
print(cuad1.calcular_perimetro()) # 144
```

# Clases | Método `__str__()`

Para mostrar objetos, Python provee otro método especial, llamado `__str__`, que debe devolver una cadena de caracteres con lo que queremos mostrar. Este método se invoca cada vez que se llama a la función `str`, por ejemplo, al imprimir el objeto.

El método `__str__` tiene un solo parámetro, `self`.

`__init__()`

# Creamos la clase "Alumno":

**class** Alumno:

**def** `__init__`(**self**, nombre, nota):

**self**.nombre = nombre

**self**.nota = nota

**def** `__str__`(**self**):

**return** f " La nota de {**self**.nombre} es  
{**self**.nota}"

alumno1 = Alumno("Pedro", 7)

**print**(alumno1) # La nota de Pedro es 7

alumno1.nota = 10

**print**(alumno1) # La nota de Pedro es 10

# Clases | Método `__del__()`

El método especial `__del__()` se invoca automáticamente cuando el objeto se elimina de la memoria. Se puede utilizar para realizar alguna acción especial cuando tiene lugar este evento. Su sintaxis es la que vemos en el ejemplo, y tiene como único parámetro **self**. Los objetos se borran con **del**, o se eliminan al finalizar el programa.

## Método `__del__()`

```
class Perro:
    def __init__(self, nombre, raza):
        self.nombre = nombre
        self.raza = raza
    def __del__(self):
        print('Objeto eliminado.')

perro1 = Perro("Lassie", "Collie")
print(perro1.nombre) # Lassie
print(perro1.raza) # Collie
del perro1
```

# Clases | Ejemplo de uso de clases y objetos

El siguiente ejemplo implementa todo lo explicado hasta aquí:

- ✓ Se implementa una clase llamada Alumno, que posee un atributo de clase (nro\_alumnos) que lleva la cuenta de los objetos instanciados.
- ✓ Cada objeto posee un nombre y una nota.
- ✓ Se definen métodos para inicializar sus atributos, imprimir el estado del objeto, procesar su eliminación de la memoria y para mostrar un texto con su estado. El estado es “regular” (nota menor o igual a 4), “bueno” (nota mayor a 4 y menor que 9) o “excelente” (nota mayor que 9).
- ✓ En el programa principal se instancian dos objetos de la clase Alumno y se muestran algunas de sus características. Al salir del programa se ve como son eliminados de la memoria.

# Clases | Ejemplo de uso de clases y objetos

## Clase Alumno (parte I )

```
class Alumno: # Creamos la clase
nro_alumnos = 0 # Cantidad de legajos existentes
#Constructor
def __init__(self,nombre,nota):
    self.nombre = nombre
    self.nota = nota
    Alumno.nro_alumnos += 1 # Agregamos un legajo
# Mostrar datos del objeto
def __str__(self):
    return f"Nombre: {self.nombre} (nota: {self.nota})"
# Damos de baja el alumno
def __del__(self):
    Alumno.nro_alumnos -= 1 # Restamos un legajo
print("Alumno dado de baja.")
print(f"{Alumno.nro_alumnos} legajos restantes.")
```

## Clase Alumno (parte II )

```
def mostrar_estado(self): # ¿está aprobado?
print(f"El estado de {self.nombre} es ",end="" )
if self.nota <= 4:
    print("regular")
elif self.nota < 9:
    print("bueno")
else:
    print("excelente")
# Programa principal
alumno1 = Alumno("Aldo López", 8)
alumno2 = Alumno("Juana Martín", 3)
print(alumno1) # Nombre: Aldo López (nota: 8)
print(alumno2) # Nombre: Juana Martín (nota: 3)
alumno1.mostrar_estado() # El...de Aldo López es bueno
alumno2.mostrar_estado() # El...Juana Martín es regular
input("Pulse enter para salir")
```

# **MATERIAL EXTRA**



# Artículos de interés

## Material extra:

- ✓ [Conceptos fundamentales sobre POO](#) explicados de manera simple
- ✓ [Guía para Principiantes de la POO](#), en kinsta
- ✓ [Clases y objetos](#), en la documentación de Python
- ✓ [La función map\(\)](#), en Hektor Docs

## Videos:

- ✓ Objetos [parte I](#), [parte II](#), [parte III](#) y [parte IV](#) en Píldoras informáticas
- ✓ [Clases y Objetos](#) en Python, por yacklyon