Objetos

Clase #23

IIC1103 – Introducción a la Programación

El plan de hoy es...

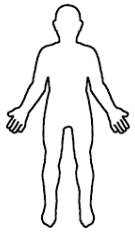
- Orientación a Objetos

- Recordar plazo tarea 2: 31/mayo 18:30

Definiendo tipos propios

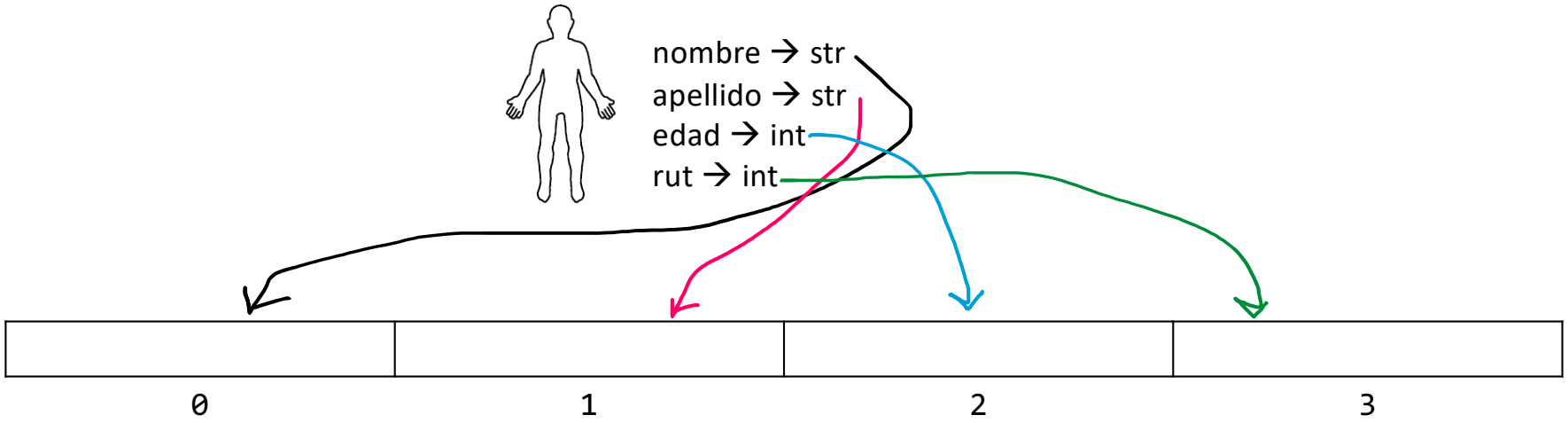
- Tenemos los siguientes tipos:
  - int → representa un número entero
  - float → representa un número real
  - bool → representa un valor de verdad (True/False)
  - str → representa un string (texto)
  - list → representa una lista de elementos

- Quiero definir un tipo propio para representar una Persona



nombre → str
apellido → str
edad → int
rut → int

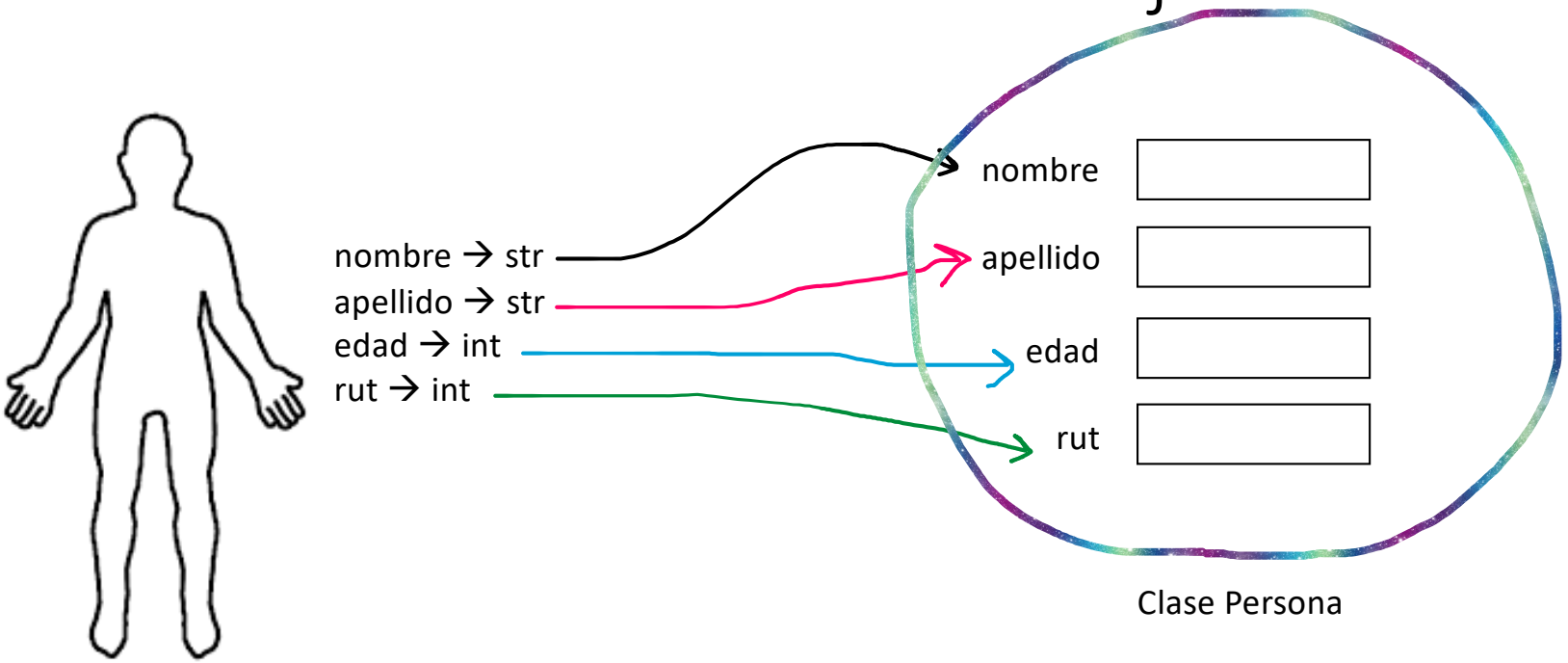
Es cierto que podría hacer una lista…



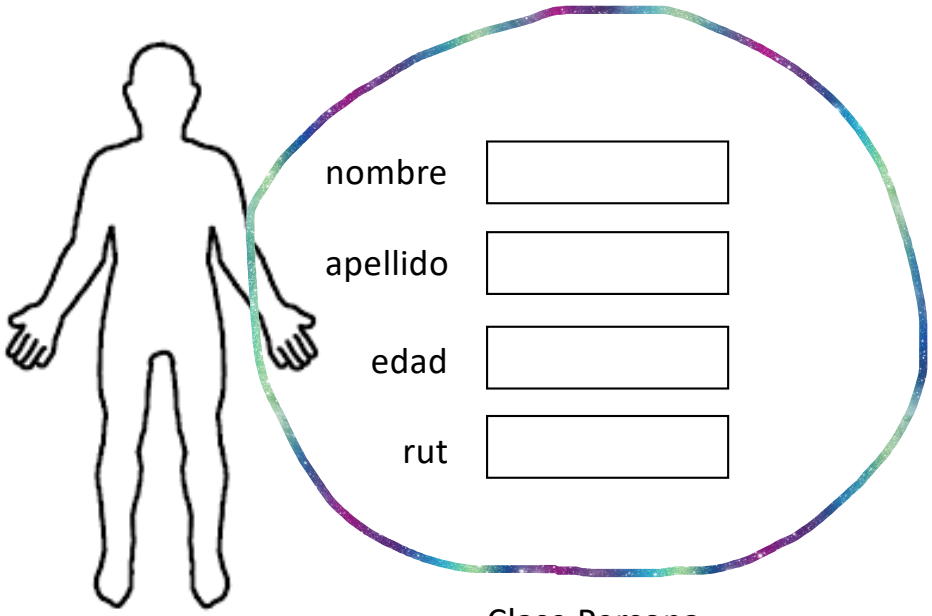
Por ejemplo

Margaret	Hamilton	83	123456
0	1	2	3

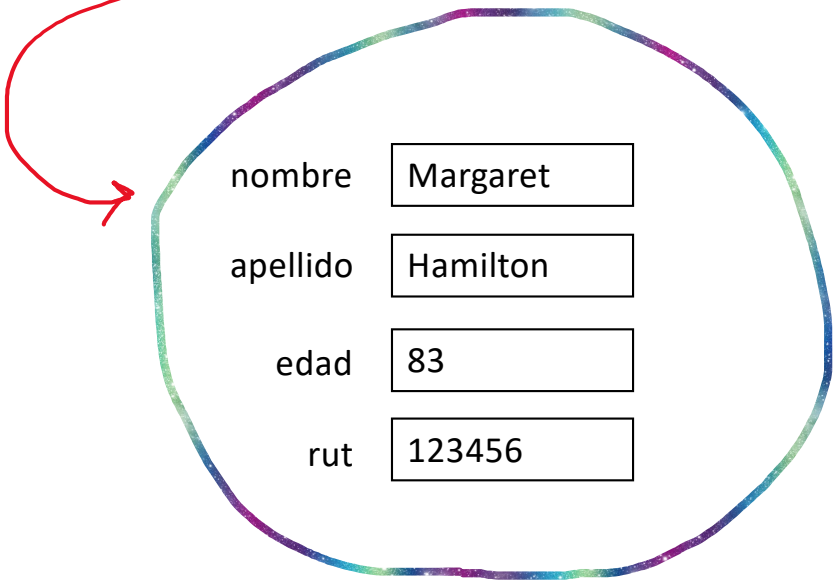
Pero haremos orientación a objetos!



Por ejemplo



Clase Persona



Instancia de Persona
ó
Objeto Persona

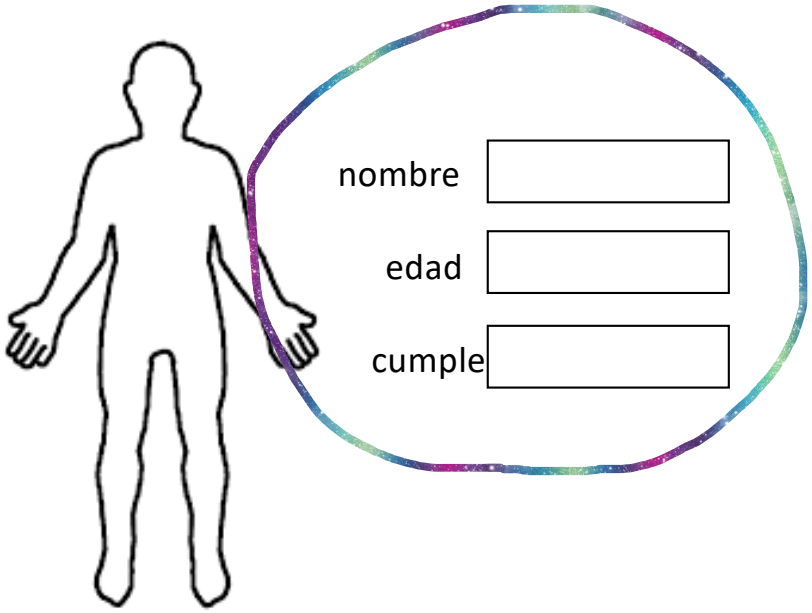

Objetos Galleta
Clase Galleta

Programación Orientada a Objetos (OOP)

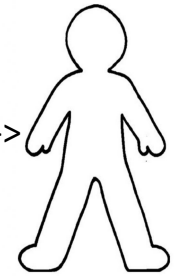
- Definiremos objetos, con:
  - atributos (características)
  - métodos (maneras de interactuar con objetos)

Programación Orientada a Objetos (OOP)

- Definiremos objetos, con:
  - atributos (características)
  - métodos (maneras de interactuar



objeto
de tipo persona -->

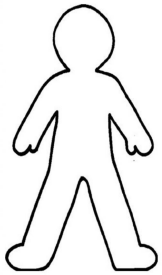


atributos

Juan	21	17/10
nombre	edad	cumple

atributos

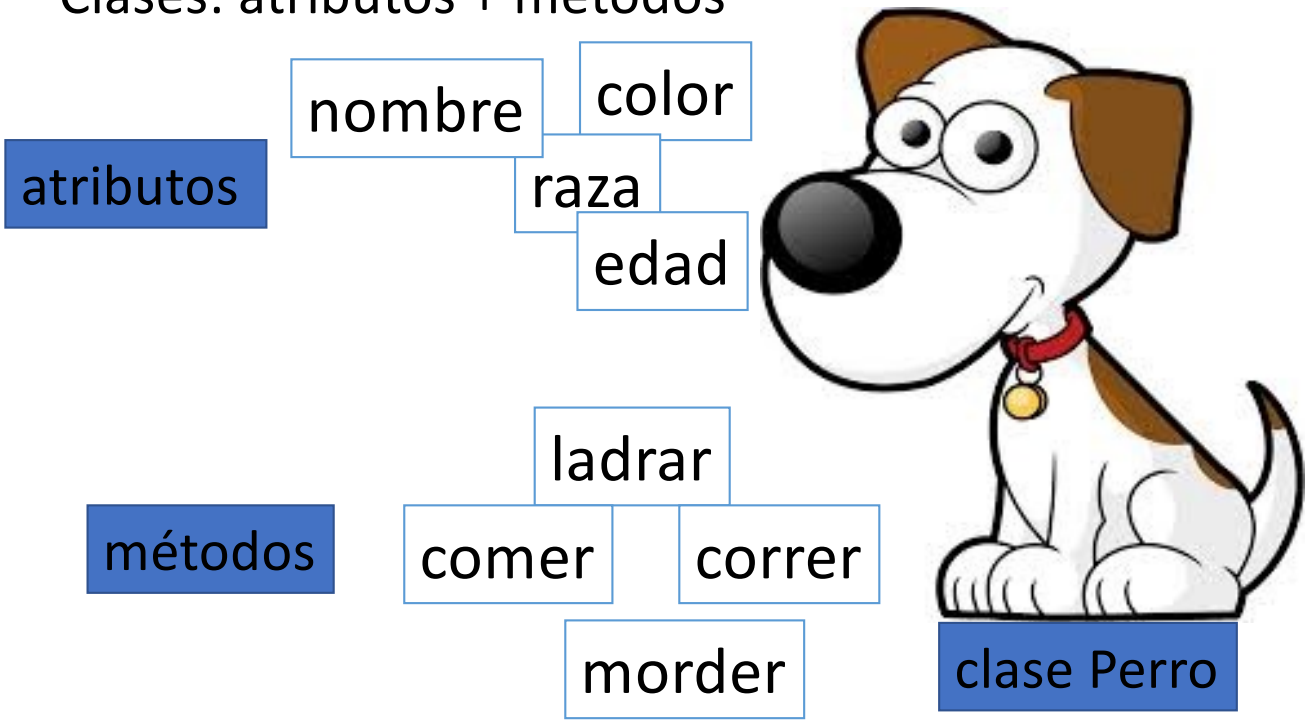
Elena	19	22/08
nombre	edad	cumple

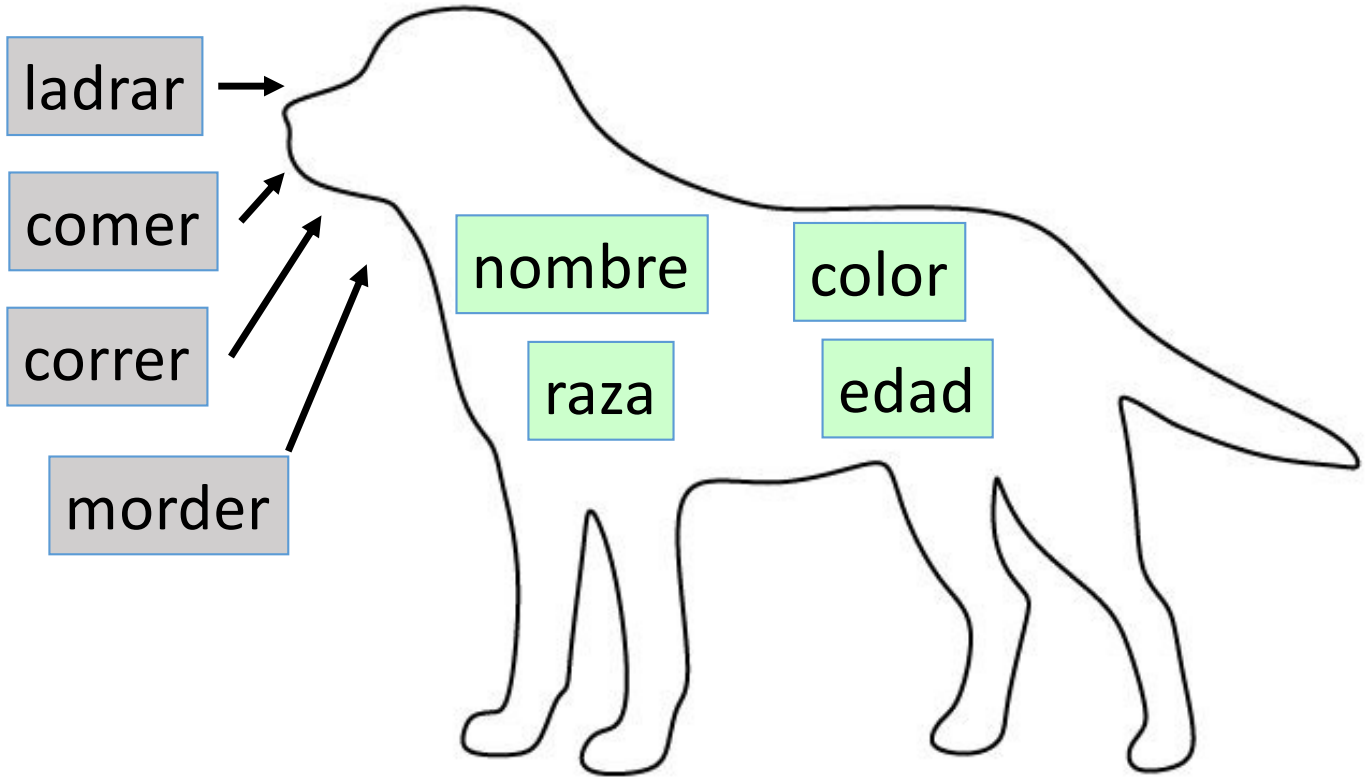


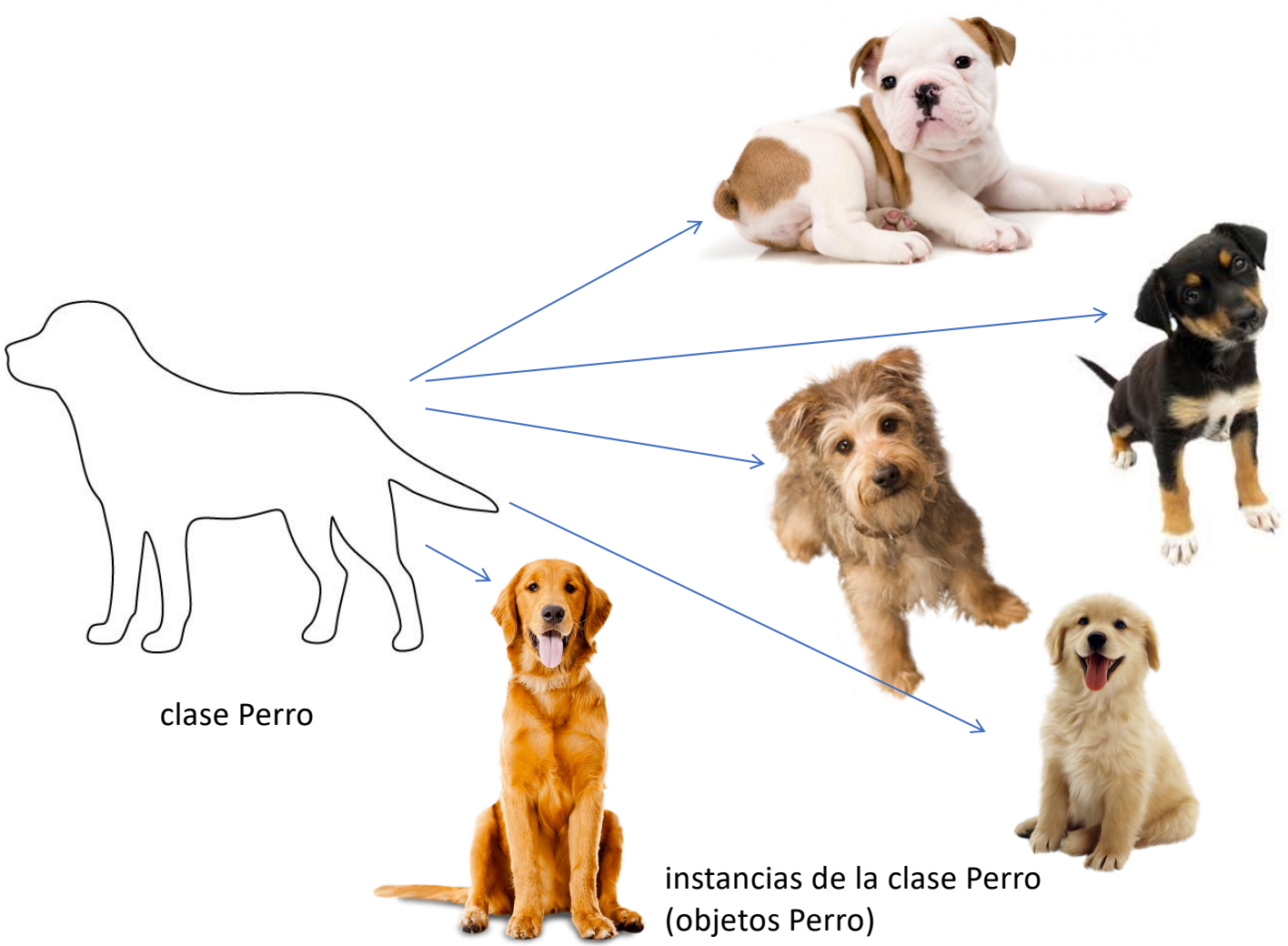
<-- objeto
de tipo persona

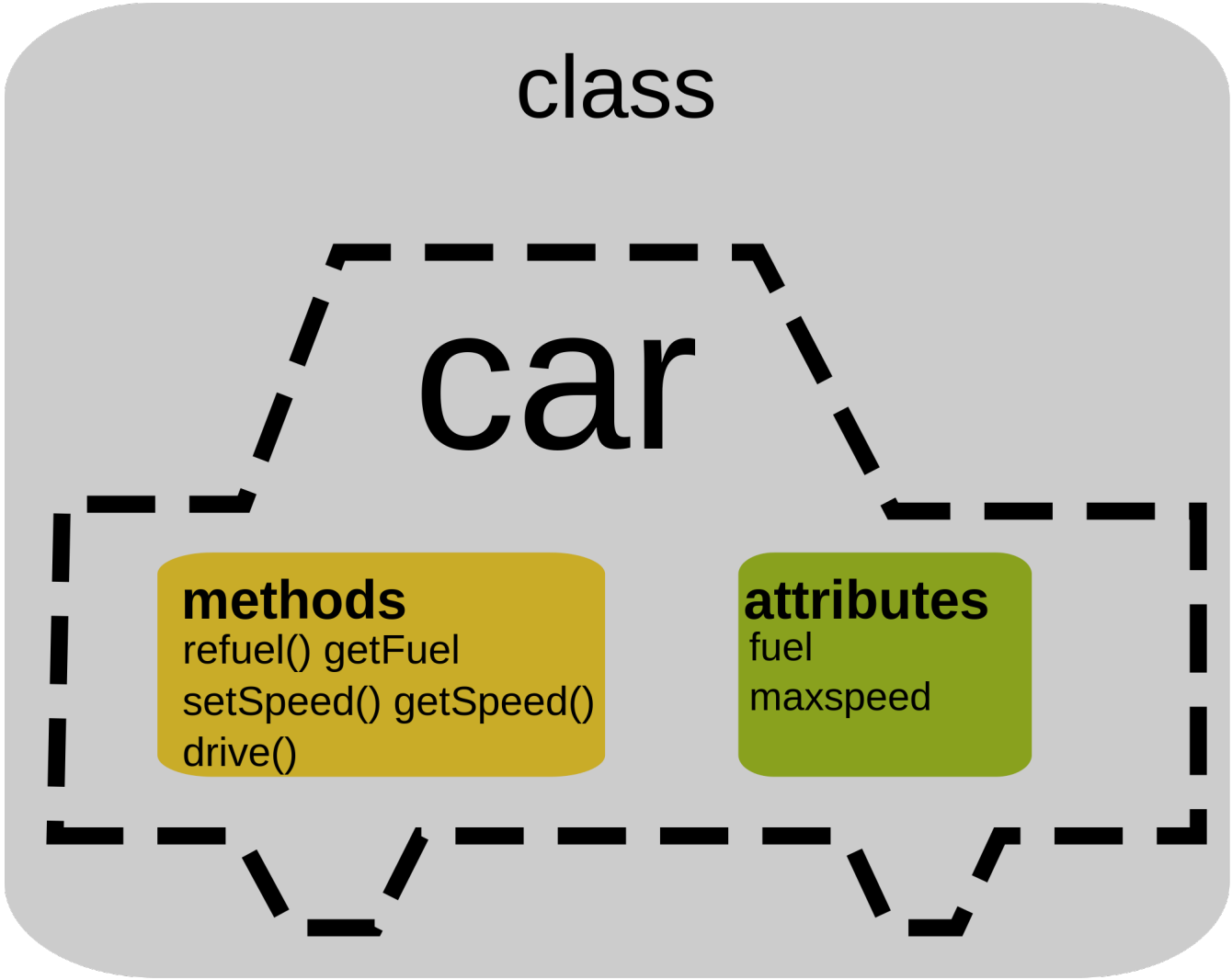
Más que funciones… **métodos**

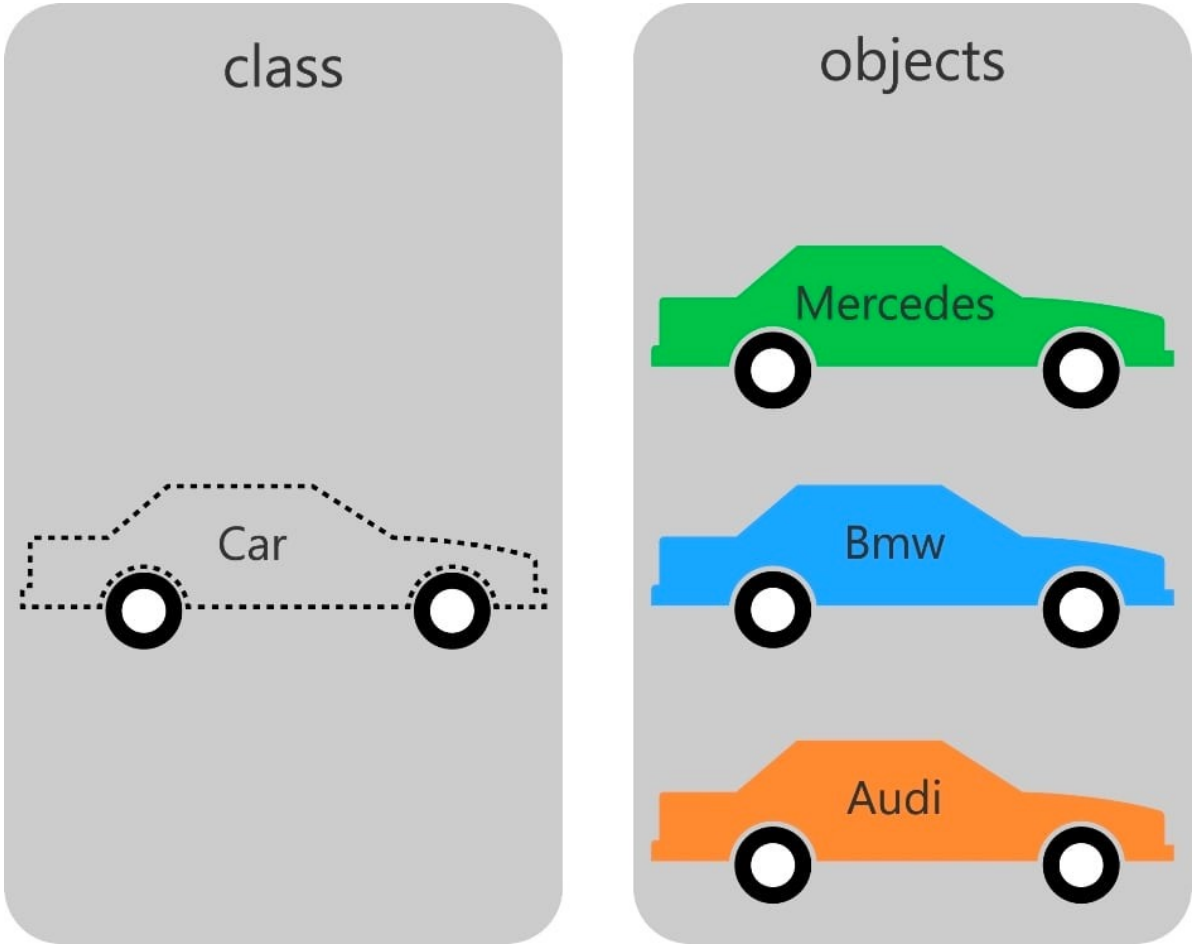
- Métodos se definen dentro de una clase: relación explícita entre clase y método
- Clases: atributos + métodos




ladrar
comer
correr
morder
nombre
raza
color
edad

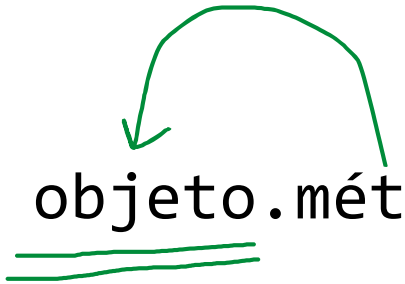

clase Perro
instancias de la clase Perro
(objetos Perro)


class
car
methods
refuel() getFuel
setSpeed() getSpeed()
drive()
attributes
fuel
maxspeed


class
Car
objects
Mercedes
Bmw
Audi

Los **métodos** se aplican sobre **objetos**

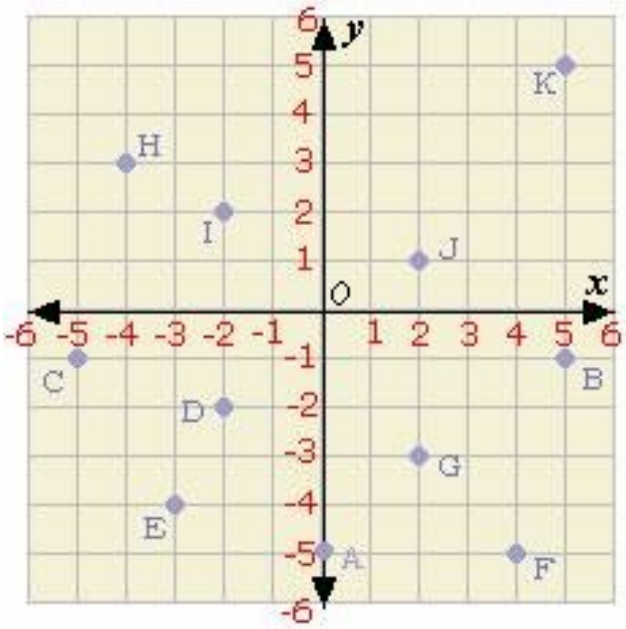
- `bobby = Perro(“Bobby”,”quiltro”,3,”blanco”)`
- `bobby.ladrar()`
- `bobby.comer(“comida de perros”)`
- `bobby.correr(100)`



`objeto.método(parámetros)`

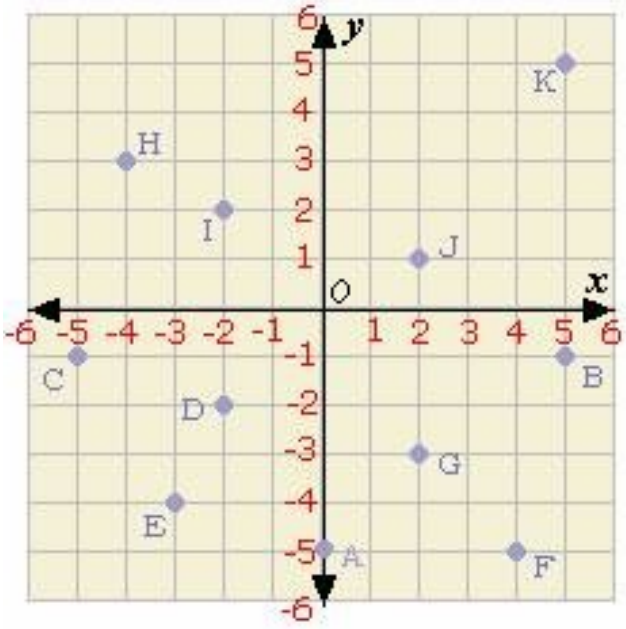
Problema #1

- Calcular la distancia de dos puntos de un plano cartesiano, e.g. A y K



Solución

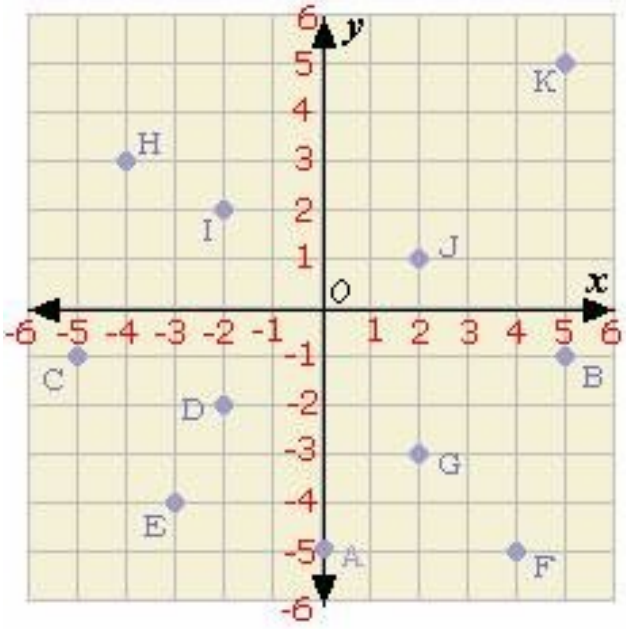
- Calcular la distancia de dos puntos de un plano cartesiano



- `import math`
- `a_x = 0`
- `a_y = -5`
- `k_x = 5`
- `k_y = 5`
- `dist = math.sqrt((a_x-k_x)**2 + (a_y-k_y)**2)`
- `print(“La distancia es “+str(dist))`

Solución #2 – con funciones

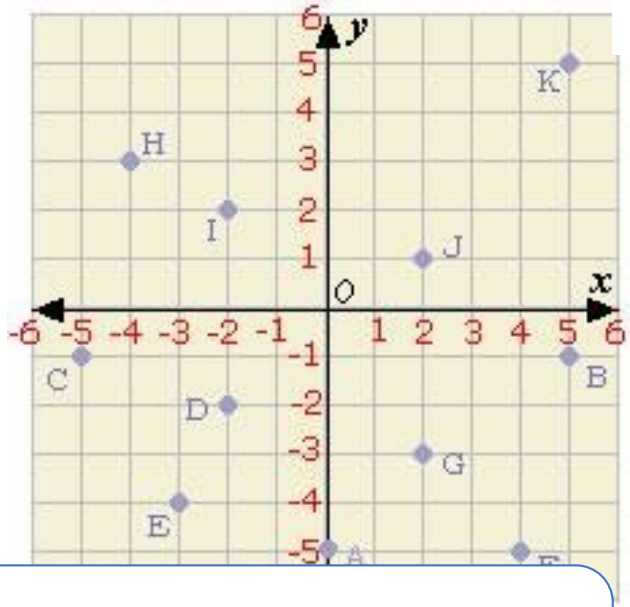
- Calcular la distancia de dos puntos de un plano cartesiano




```
import math
def distancia(x1,y1,x2,y2):
    return math.sqrt((x1-x2)**2+(y1-y2)**2)
a_x = 0
a_y = -5
k_x = 5
k_y = 5
dist = distancia(a_x,a_y,k_x,k_y)
print("La distancia es "+str(dist))
```

Solución usando OOP

- `a = Punto2D(0,-5)`
- `k = Punto2D(5,5)`
- `dist = a.distancia(k)`
- `print("La distancia es "+str(dist))`



Más simple y legible (entendible)
Más cercano a modelar el problema real

clase Punto2D: atributos + metodos

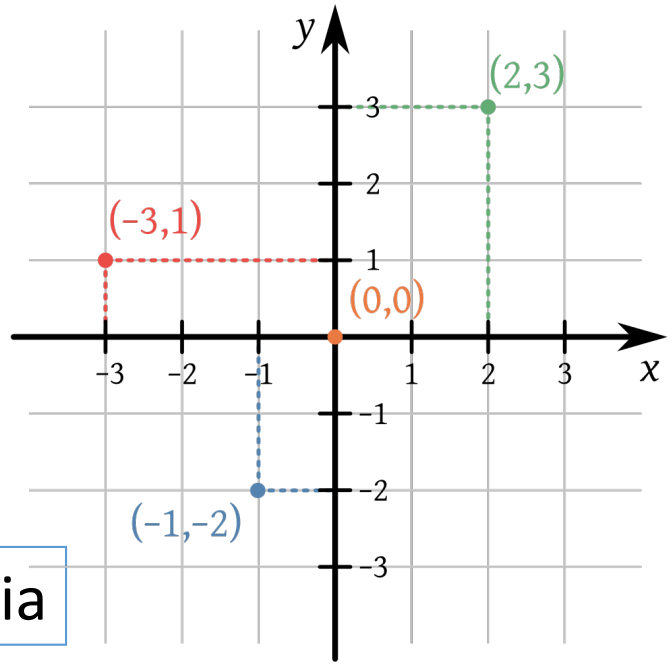
atributos

x

y

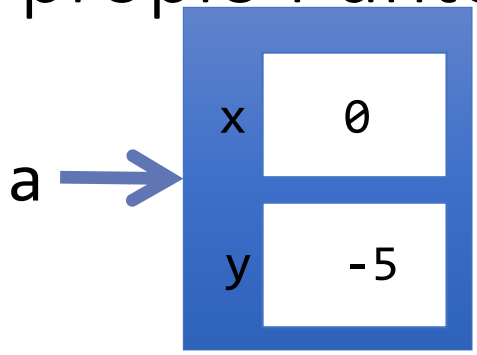
métodos

distancia



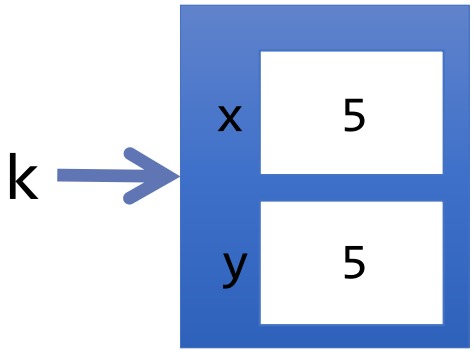
Definiendo nuestro propio Punto2D

- a = Punto2D(0,-5)



Cada punto tiene dos variables internas o atributos, llamados x e y

- k = Punto2D(5,5)



Pero: ¿Cómo hago para crear un Punto2D?: método constructor

- `class Punto2D:`

- **`def __init__(self, a, b):`**
- **`self.x = a`**
- **`self.y = b`**

- **`punto = Punto2D(5,5)`**

self: el primer parámetro
se refiere al objeto sobre el cual se aplica el método

distancia

- `def distancia(self, other):`
- `        return math.sqrt((self.x-other.x)**2+(self.y-other.y)**2)`

Solución completa

```
import math

class Punto2D:
    def __init__(self, a, b):
        self.x = a
        self.y = b

    def distancia(self, other):
        return math.sqrt((self.x-other.x)**2+(self.y-other.y)**2)

a = Punto2D(0,0)
k = Punto2D(2,2)
dist = a.distancia(k)
```

atributos

constructor

método

Código principal

¿Qué pasa?

- `p = Punto2D(4,4)`
- `print(p)`

Solución completa, para poder imprimir Punto2d

```
import math

class Punto2D:
    def __init__(self, a, b):
        self.x = a
        self.y = b

    def distancia(self, other):
        return math.sqrt((self.x-other.x)**2+(self.y-other.y)**2)

    def __str__(self):
        return str(self.x)+", "+str(self.y)

a = Punto2D(0,0)
k = Punto2D(2,2)
dist = a.distancia(k)
```

constructor

atributos

método

redefinición de método existente

Código principal

Menti

- `c1 = Carta(13,”C”)`
- `c2 = Carta(5,”D”)`
- `d = c1.diferencia(c2) #d es 8`
- `print(c1) #imprime KC`

Resumen de hoy

```
class <Nombre de Clase>:
    def __init__(self, parámetros):
        self.atributo1 = ...

    def método(self,...):
        ...

    def __str__(self):
        ...
```