

Programación Orientada a Objetos

Nivelatorio para trabajar con data



Profesor José Tomás Marquinez V.

Introducción

- Conocemos los tipos de datos básicos de Python
 - int, str, list, float, bool, ...

```
>>> type("w")  
<class 'str'>  
>>> type(2)  
<class 'int'>  
>>> type(True)  
<class 'bool'>  
>>> type([2,4,5])  
<class 'list'>
```

- Ahora queremos definir nuestros propios tipos de datos
 - Perro, Carta, Auto, Estudiante, Sensor, Semáforo, Comuna, ...
- Que además permitan hacer cosas útiles
 - Ladrar, cambiar_pinta, estudiar, cambiar_luz, ...

TIPOS PROPIOS DE DATOS

Objetos

Introducción

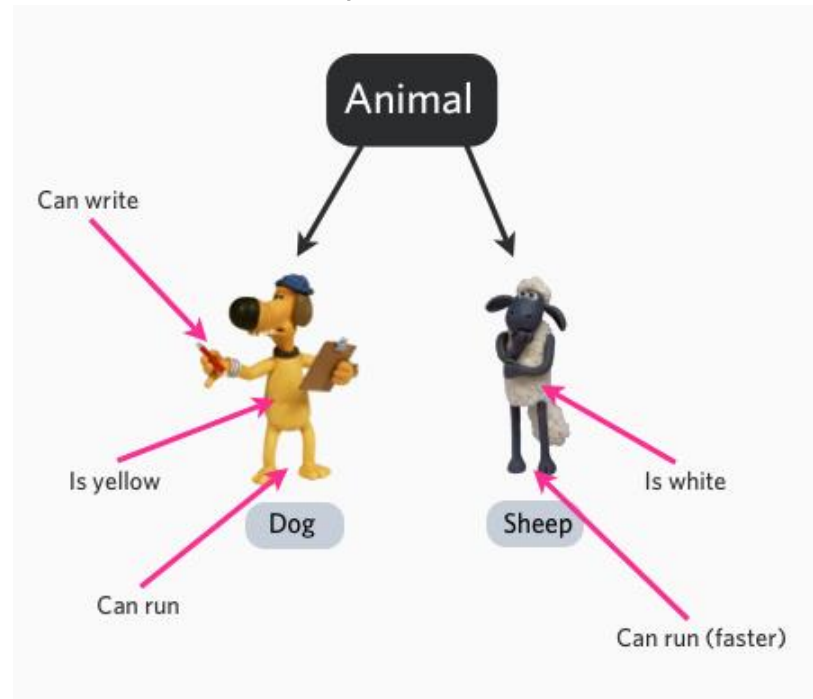
- En la vida, todo es un objeto:



¿Qué es un objeto?

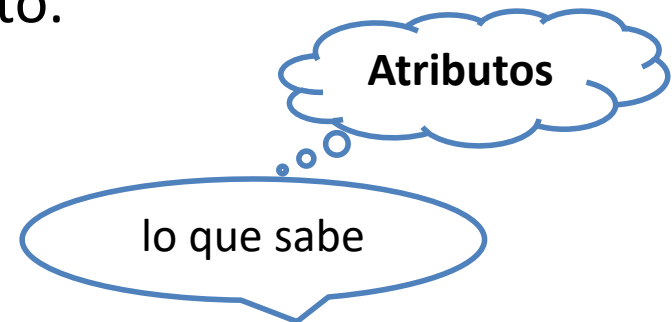


- **Definición:** La *Programación Orientada a Objetos* (POO; o OOP por sus siglas en inglés) es un paradigma de programación, en que los programas son constituidos por definición de objetos y métodos.
 - Permite que la computación se pueda hacer como operaciones sobre objetos.
- **Definición:** Un *objeto* es un conjunto de datos y **tiene** un conjunto de comportamientos.
 - Cada objeto podría, por ejemplo, corresponder a un objeto del mundo real, y permiten interactuar con ellos de cierta forma.



¿Qué es un objeto?

- **Definición:** Los *atributos* de un objeto es el conjunto de datos que hace un objeto.
 - Es lo que sabe, sus características, su estado.
- **Definición:** Los *métodos* de un objeto es el conjunto de comportamientos que tiene el objeto.
 - Es lo que puede hacer

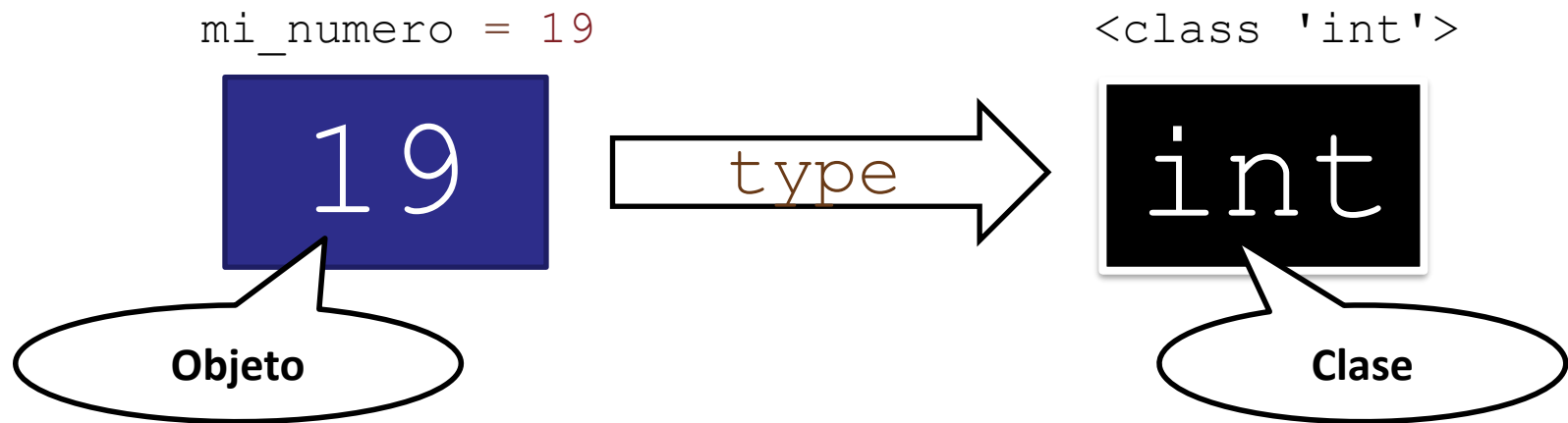


- **Definición** (de nuevo): Un *objeto* es un conjunto de datos y **tiene** un conjunto de comportamientos.



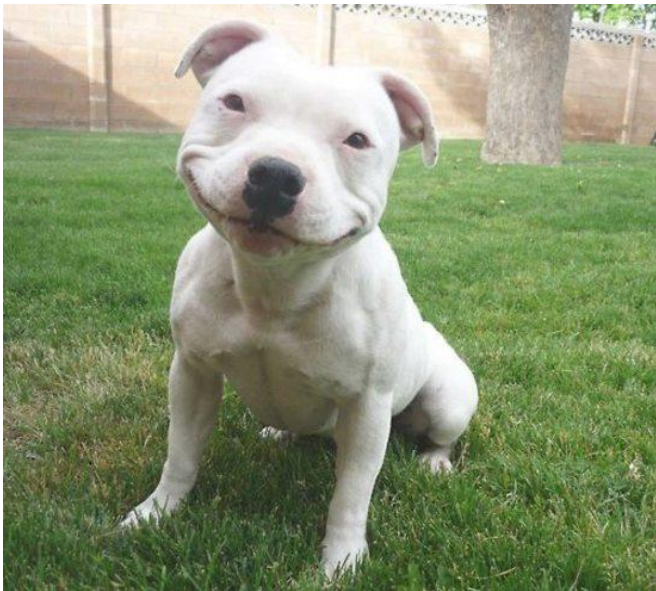
¿Qué es un objeto?

- **Definición:** Una *clase* es una plantilla para la creación de objetos de datos según un modelo predefinido.
 - Es básicamente, la definición de un conjunto de atributos y métodos para un objeto en particular.
 - Es decir, **es el molde**.
- **Función:** La función `type()` permite averiguar de qué clase es el objeto que le entregamos como parámetro.



¿Qué es un objeto?

- Así, una clase son atributos y métodos.
 - Hay una relación explícita entre clase y método.



Clase Perro

Atributos

Nombre
Raza
Edad
Color
Está sonriendo

Métodos

Ladrar
Comer
Correr
Morder al cartero
Deja de sonreír

¿Qué es un objeto?

- Cada clase tiene sus atributos y sus métodos propios, pues son distintas.



Clase Perro

Atributos

Nombre

Raza

Color

Edad

Está sonriendo

Métodos

Ladrar

Correr

Comer

Morder al cartero

Sonreír



Clase Oveja

Peso

Edad

Color

Cantidad de lana

Comer

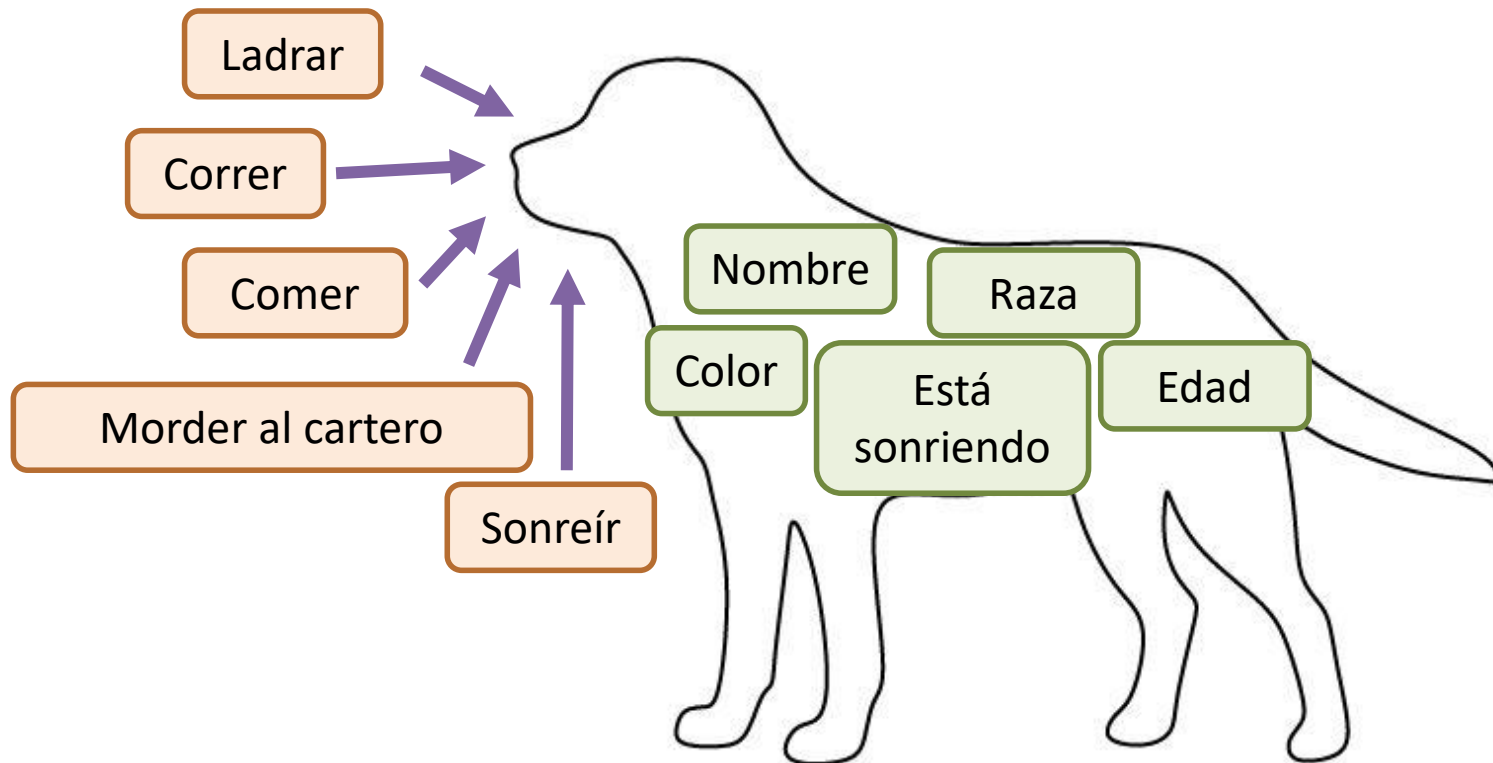
Correr

Balar

Ser trasquilada

¿Qué es un objeto?

- Así, estamos definiendo que todo perro seguirá este molde:



¿Qué es un objeto?

- Finalmente, los métodos se aplican sobre objetos.

```
bobby = Perro("Bobby", "quiltro",  
3, "blanco")
```

```
juan = Cartero("Juan Perez")
```

```
bobby.ladRAR()
```

```
bobby.ladRAR()
```

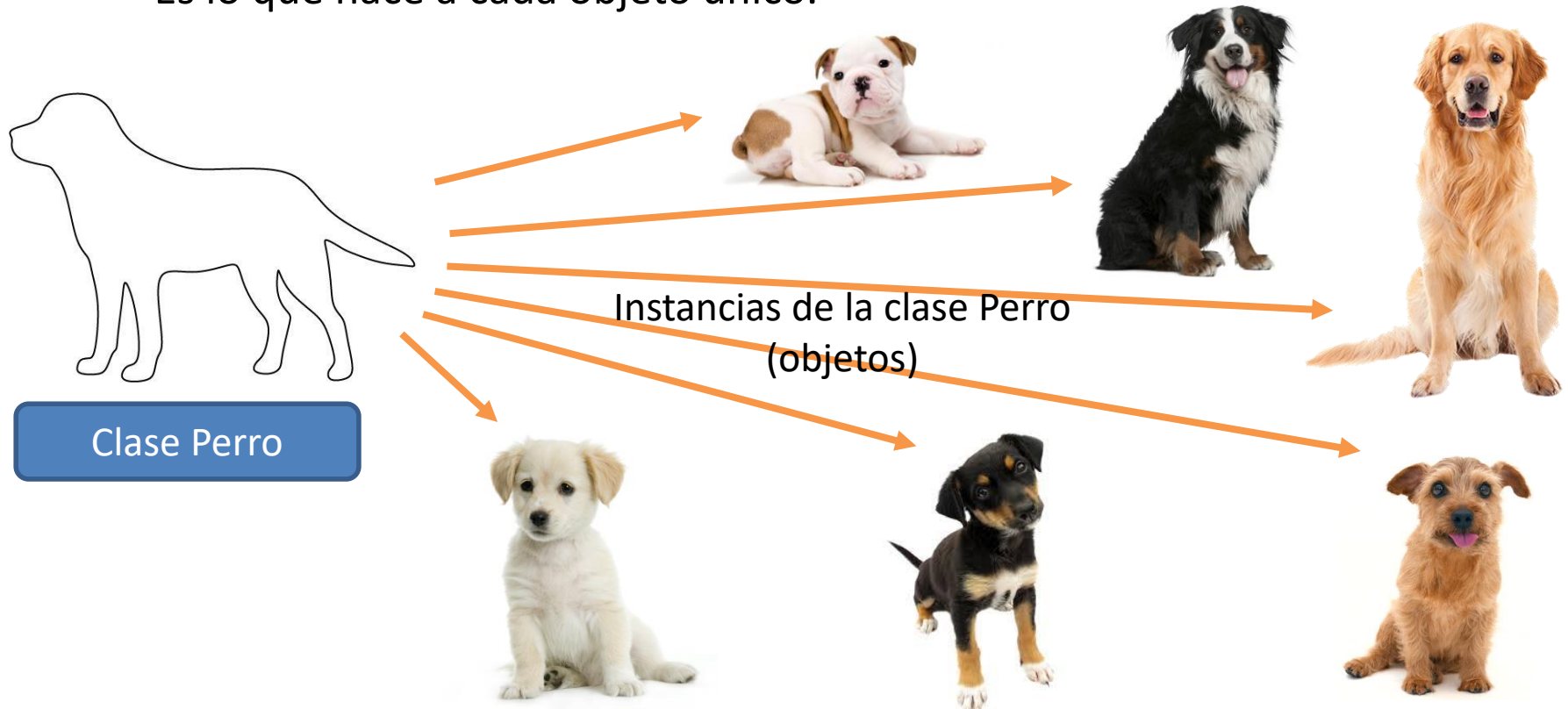
```
bobby.morder(juan)
```



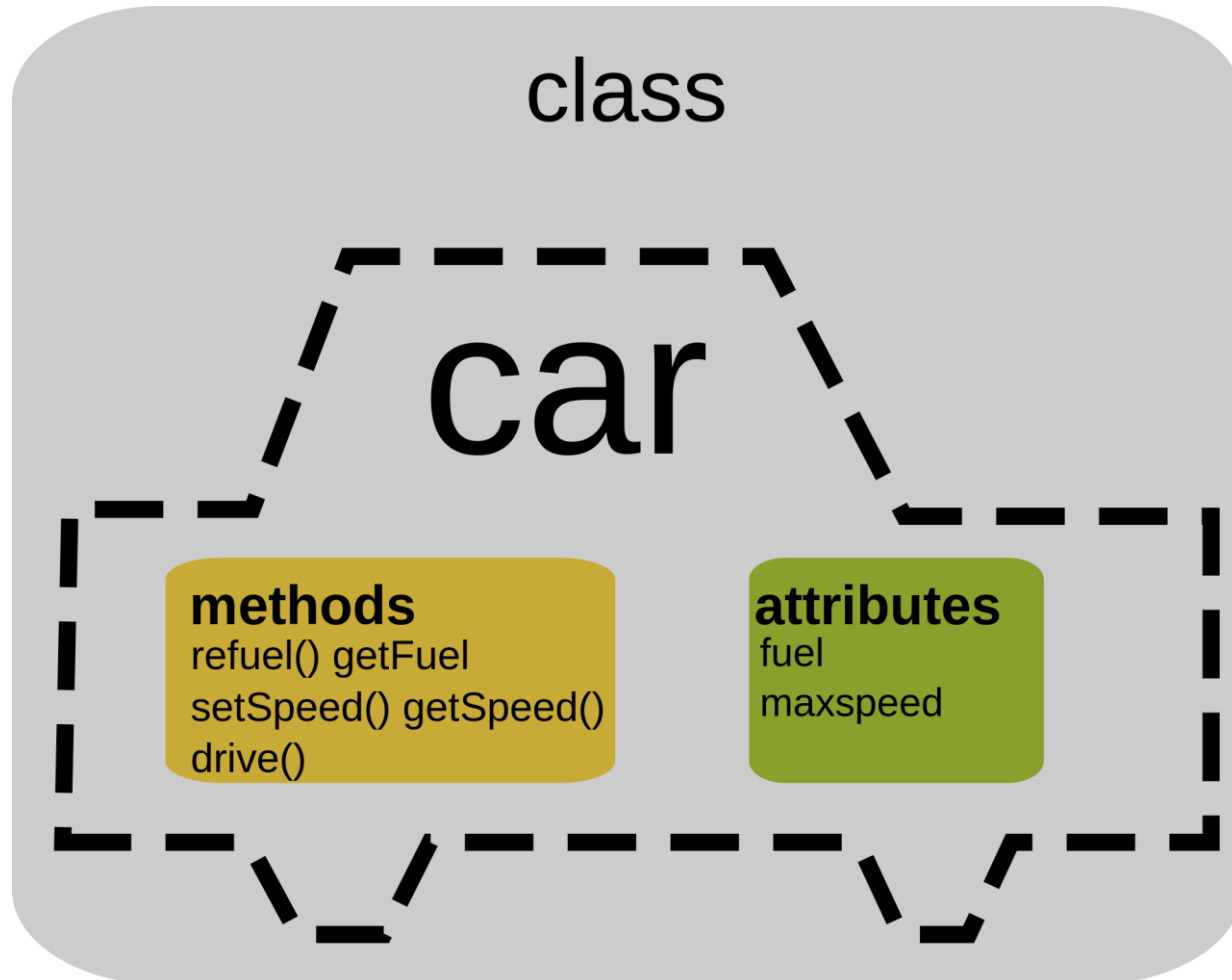
INSTANCIA

¿Qué es un objeto?

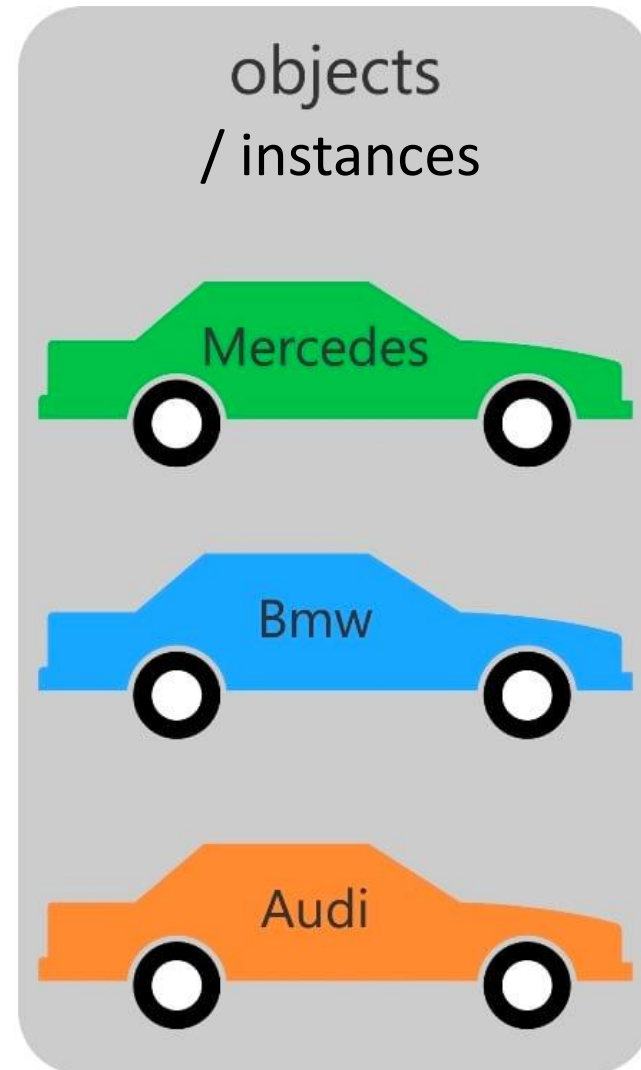
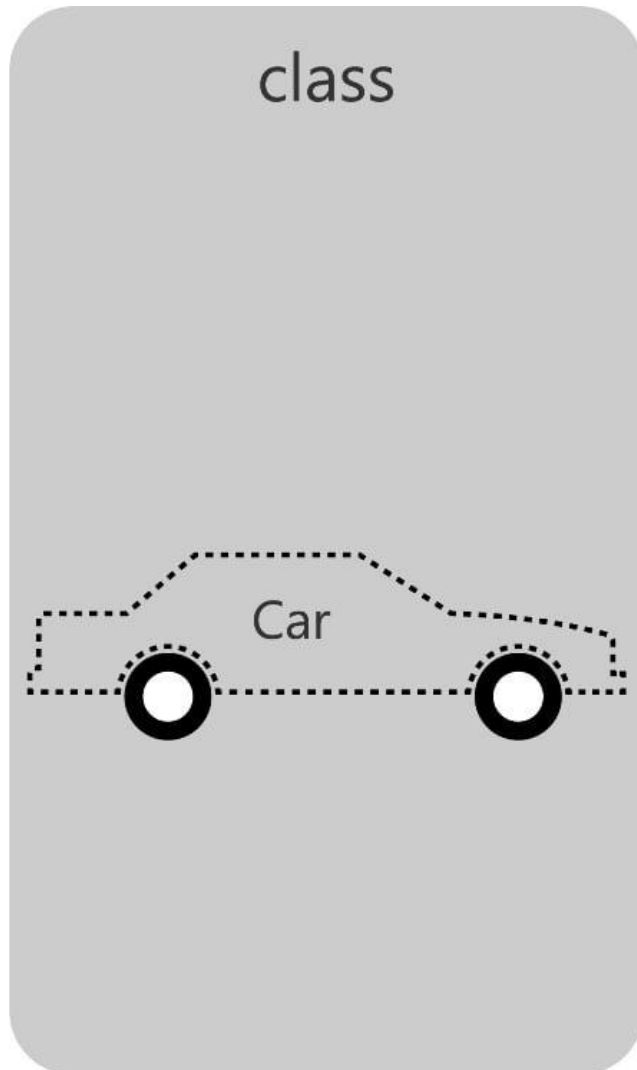
- **Definición:** Cada objeto creado a partir de una clase se denomina *instancia* de esa clase.
 - Es la particularización o realización específica de una clase.
 - Es lo que hace a cada objeto único.



¿Qué es un objeto?



¿Qué es un objeto?



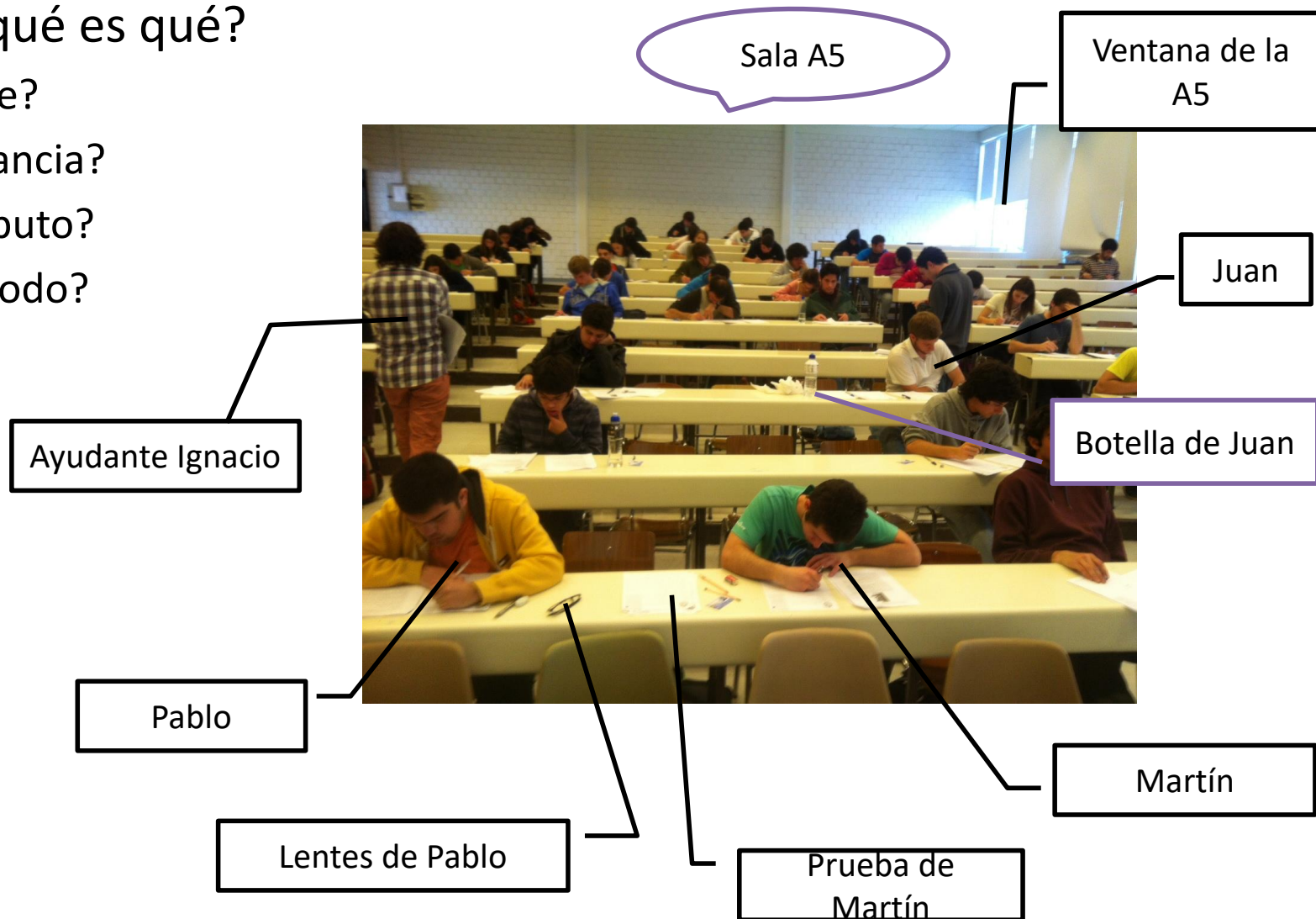
RECONOCIENDO...

Clases, instancias (objetos), atributos, métodos

¿Qué es un objeto?

- Acá, ¿qué es qué?

- ¿Clase?
- ¿Instancia?
- ¿Atributo?
- ¿Método?



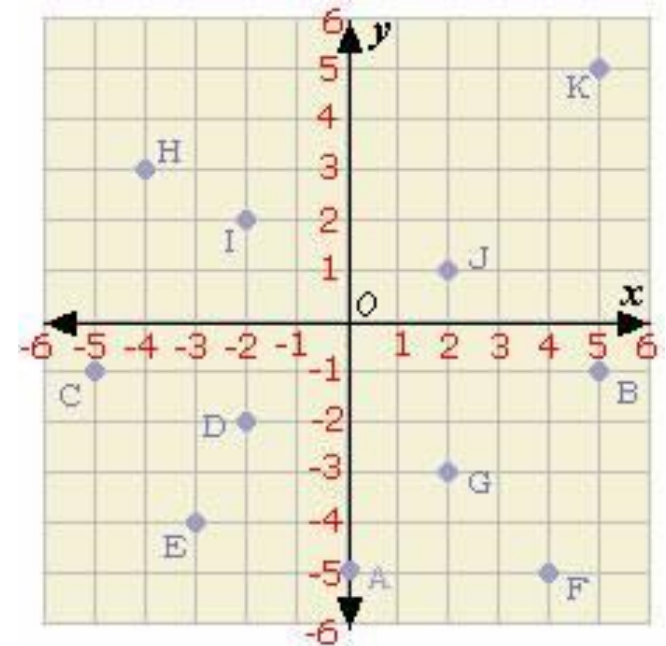
DISEÑAR MÁS PARA PROGRAMAR MEJOR

Ventajas de programar OO

Basemos la mayoría de la clase en un ejemplo. Ejemplo 1:

- Calcular la distancia entre dos puntos de un plano cartesiano. Por ejemplo, entre A y K.

```
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))
```



ejemplo1-1.py

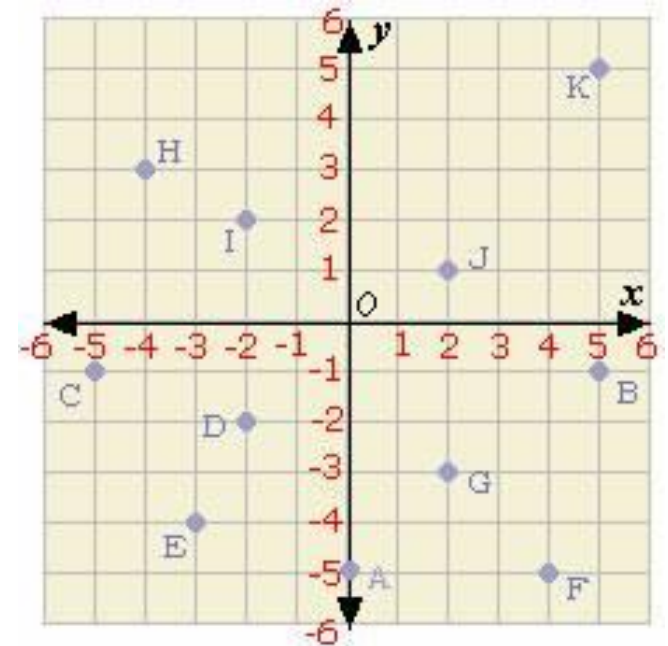
Ventajas de programar OO

Ejemplo 1:

- Otra forma sería utilizando funciones:

```
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))
```



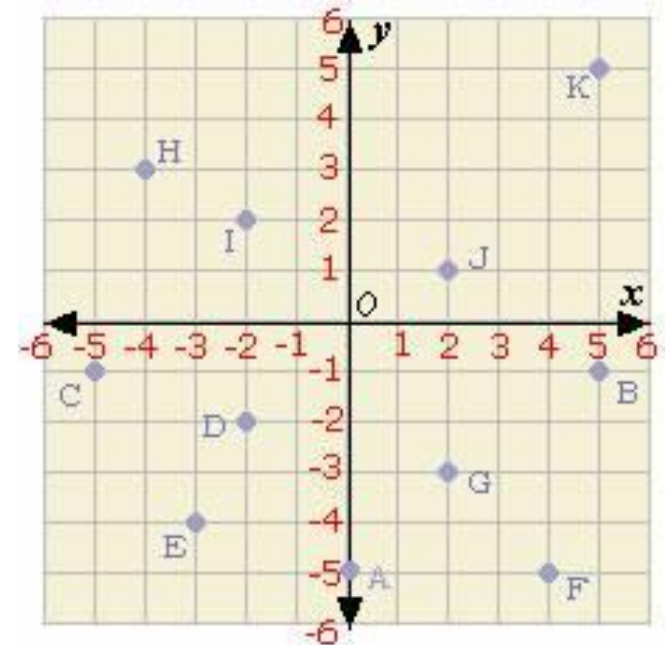
ejemplo1-2.py

Ventajas de programar OO

- Sin embargo, con el paradigma POO queda mucho más directo.

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

- No sólo es más directo...



Ventajas de programar OO

- Es más simple.
- Es más legible.
- Es más cercano a modelar el problema real.
- Es más fácil de escribir.
- Es más fácil de entender.
- Es más fácil de mantener.
- Permite definir nuestros tipos de dato con el comportamiento que nosotros queramos.

```
turno = 0
continuar = True
while continuar:
    if turno == 0:
        print("Turno de", m1.name())
        spell = input("Ingresar hechizo: ")
        s = m1.attack(spell, m2)
        if s:
            print(m1.name(), "atacó exitosamente a", m2.name(), "con", spell, "y lo deja con", m2.life(), "de vida")
        else:
            print(m1.name(), "falló")
        if m2.life() <= 0:
            continuar = False
    else:
        print("Turno de", m2.name())
        spell = input("Ingresar hechizo: ")
        s = m2.attack(spell, m1)
        if s:
            print(m2.name(), "atacó exitosamente a", m1.name(), "con", spell, "y lo deja con", m1.life(), "de vida")
        else:
            print(m2.name(), "falló")
        if m1.life() <= 0:
            continuar = False
    turno = (turno+1)%2
```



CÓMO DEFINIMOS NUESTRO PROPIO TIPO DE DATO

¿Cómo definimos un objeto?

- Entonces... ¡Manos a la obra!
- Primero lo primero: Para definir un tipo nuevo (objeto), es necesario definir su clase:

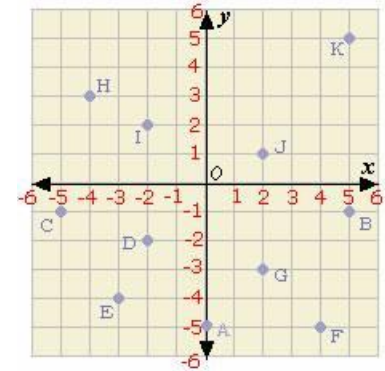
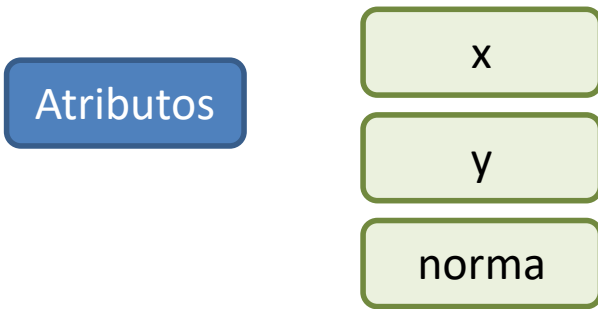
Sintaxis:

```
class NombreClase:  
    # ...  
    #código de la clase  
    # ...
```

- Las clases suelen escribirse en “CamelCase”
 - Mayúscula cada palabra, y todas juntas.
 - Ejemplo: Perro, Oveja, Estudiante, Auto, CartaPoker, ProfesorTitular, ...

¿Cómo creamos un objeto?

- Ahora, ¿qué atributos podría tener la clase Punto2D?



- Por lo mismo, cuando queramos crear un objeto nuevo, uno buscará entregar la información básica necesaria (los atributos básicos).

Sintaxis:

`nombre_instancia = NombreClase (<atributos,parametros>)`

Atributos

- Punto 2D:
 - Creación: `a = Punto2D(5,5)`
- Carta (naipe)
 - Atributos: Número (int), Pinta (str)
 - Creación: `mi_carta = Carta(2, "corazones")`
- Auto
 - Atributos: Modelo (str), Patente (str)
 - Creación: `auto1 = Auto("Corsa", "ABCD11")`
- Estudiante
 - Atributos: Nombre (str), Rut (str), Tareas (float), Interrogaciones (float), Examen (float), Participación (float)
 - Creación: `juan = Estudiante("Juan", "22.222.222-2", 6.7, 5.0, 6.0, 7.0)`
`ricardo = Estudiante("Ricardo", "22.333.333-3", 5.0, 5.5, 6.5, 7.0)`

Es buena práctica escribir las instancias/variables en 'snake_case' (como todo en python).
Es decir, minúscula cada palabra, separadas por _

Constructor

- Ahora, ¿cómo sabe Python qué tiene que hacer cuando crea un objeto?
- Cada clase tiene definido un método especial llamado `__init__()`, que le permite controlar cómo se *inicializan* los atributos de los objetos de una clase.
 - Notar que hay dos `'_'` antes y después.

Sintaxis:

```
class NombreClase:  
    def __init__(self<, parámetros>):  
        #Acá se inicializa el valor de los ATRIBUTOS  
        #Los atributos se definen como self.atributo
```



- **Definición:** El método anterior se conoce como *constructor* de la clase.
- El primer parámetro, **self**, se refiere al objeto sobre el cual se aplica el método.
 - En este caso, estamos creando “este” Punto2D (y no otro).

PUNTO 2D

Constructor y atributos

Constructor

- Ejemplo 1:

```
class Punto2D:
    def __init__(self, a, b):
        self.x = a
        self.y = b
        self.norma = (a**2 + b**2)**(1/2)
```

```
punto_a = Punto2D(5,5)
```

ejemplo1.py

- Notar:

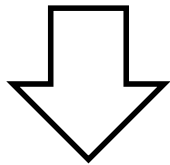
- Al llamar al constructor, sólo se pasan los parámetros que NO son el self.

- ¿Qué hace esto?

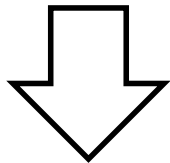
Constructor

- ¿Qué hace esto?

```
punto_a = Punto2D(0,5)
```



```
Punto2D.__init__(punto_a, 0, 5)
```



```
def __init__(self, a, b):  
    self.x = a  
    self.y = b  
    self.norma = (a**2 + b**2)**(1/2)
```

Atributos

- ¿Cómo podemos acceder a los atributos?
 - Es similar a los métodos (y ya veremos por qué)...

```
class Punto2D:
    def __init__(self, a, b):
        self.x = a
        self.y = b
        self.norma = (a**2 + b**2)**(1/2)

punto1 = Punto2D(6,5)
punto2 = Punto2D(0,5)

print("El primer punto ingresado fue el")
print("(" + str(punto1.x) + "," + str(punto1.y) + ")")

print("La norma del segundo punto ingresado es:")
print(punto2.norma)
```

ejemplo1.py

MÉTODOS

Definiendo comportamiento

Métodos

- Así como los objetos tienen estados, también tienen comportamientos → ¡Métodos!
 - Buscamos que los tipos de datos que definimos sean útiles.

Sintaxis:

- Para describir el comportamiento posible de un objeto de una clase, creamos una función dentro de una clase que tiene como primer parámetro a **self**.
 - Una función dentro de una clase es lo que se conoce como método.

```
class NombreClase:
```

```
    def nombre_método(self<, parámetros>):
```

```
        #Código del método
```

```
        <return valor_retorno>
```

ejemplo1.py

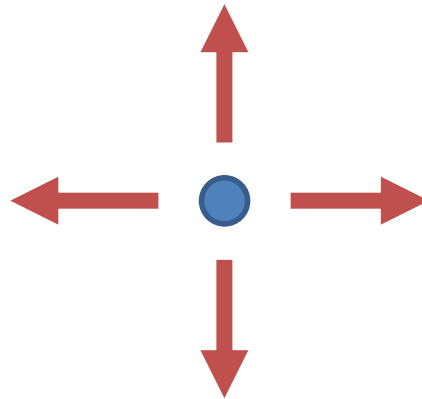
PUNTO 2D

Métodos

Métodos

Ejemplo 1:

- ¿Cómo serían los siguientes métodos que desplazan al punto?
 - **mover_x(size)**: Mueve la componente x del punto en **size** (puede ser negativo)
 - **mover_y(size)**: Mueve la componente y del punto en **size** (puede ser negativo).



[ejemplo1.py](#)

Métodos

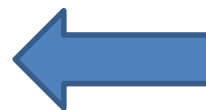
- Que, dentro de la clase, completaría el ejemplo:

```
class Punto2D:
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.norma = (x**2+y**2)**(1/2)

    def mover_x(self, size):
        self.x += size
        self.norma = (self.x**2 + self.y**2)**0.5

    def mover_y(self, size):
        self.y += size
        self.norma = (self.x**2 + self.y**2)**0.5

punto_a = Punto2D(5,5)
punto_a.mover_x(4)
print(punto_a.x)
```



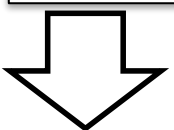
¡Ya sabíamos cómo
invocar métodos!
¿Qué otros métodos
habíamos usado? (¿De
qué clases?)

[ejemplo1.py](#)

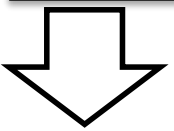
Métodos

- ¿Qué hace Python cuando invoca un método?
 - Cuando se invoca un método de una instancia, Python se encarga de que el **primer argumento** sea la instancia del objeto que invoca.
 - Este hecho explica por qué **self** es tan importante y por eso mismo debe ser el primer argumento de cada método que se escribe.

```
punto1 = Punto2D(5,5)  
punto1.mover_x(4)
```



```
Punto2D.mover_x(punto1, 4)
```



```
def mover_x(self, size):  
    self.x += size  
    self.norma = (self.x**2 + self.y**2) ** (1/2)
```

Métodos

Ejemplo 2:

- ¿Cómo sería la función `distancia()`, que reciba otro `Punto2D`?

```
def distancia(self, otro):  
    return ((self.x - otro.x)**2 +  
            (self.y - otro.y)**2)**0.5
```

- ¿Cómo usaríamos este método?
 - Notar que debe recibir una instancia de `Punto2D` para que tenga sentido calcular la distancia entre “este” (`self`) punto y el “otro” punto recibido.

[ejemplo1.py](#)

COPIA POR REFERENCIA

Copia por referencia

- Un objeto es una instancia de una clase, y por lo tanto “vive” en la memoria del computador.
- ¿Qué pasa si hacemos **print (instancia)**?

```
punto1 = Punto2D(4,3)
print(punto1)

<__main__.Punto2D object at 0x100691050>
```

La memoria de nuestro PC

0x100691048	pi
0x100691049	3,1416
0x100691050	
	punto1
0x100691060	1
0x100691065	...
0x100691070	12

[ejemplo2.py](#)

Copia por referencia

- En el siguiente fragmento de código, ¿por qué cambia el valor de `punto1.x` ?

```
punto1 = Punto2D(3, 6)
punto2 = punto1
punto2.mover_x(2)
punto2.mover_y(-5)
print(punto1.x, punto1.y)
print(punto2.x, punto2.y)
print(punto1)
print(punto2)
```

```
5 1
5 1
<__main__.Punto2D object at 0x0205EE30>
<__main__.Punto2D object at 0x0205EE30>
```

- Ambas variables apuntan al mismo objeto de la posición **0x0205EE30**

[ejemplo3.py](#)

Copia por referencia

- En el siguiente fragmento de código, ¿por qué **NO** cambia el valor de `punto1.x` ?

```
punto1 = Punto2D(5,3)
punto2 = Punto2D(5,3)
punto1.mover_y(3)
punto2.mover_x(10)
print(punto1.x, punto1.y)
print(punto2.x, punto2.y)
print(punto1)
print(punto2)
```

```
5 6
15 3
<__main__.Punto2D object at 0x020AEE50>
<__main__.Punto2D object at 0x020E0A90>
```

- Las dos variables apuntan a distintos objetos en la posición **0x020AEE50** y en la posición **0x020E0A90**

ejemplo3.py

MÉTODO __STR__

Método `__str__`

- **Definición:** Toda clase tiene, por defecto, un método `__str__()` que permite generar un string que muestra el espacio en memoria. Es invocado por `print()` o por `str()` cuando recibe un objeto de dicha clase como parámetro.
 - Notar que también son dos `'_'` antes y después.
- Uno puede redefinir este método.

Sintaxis:

```
class NombreClase:  
    def __str__(self<, parámetros>):  
        #Código del método, si es necesario  
        return valor_retorno
```

Método `__str__`

Ejemplo 1:

- ¿Qué nos gustaría que muestre la consola si es que hacemos `print(punto1)`?

- Actualmente está mostrando:

```
<__main__.Punto2D object at 0x020AEE50>
```

- ¿Cómo se suelen mostrar los puntos cartesianos?
 - Podríamos querer seguir esa representación.

```
class Punto2D:
```

```
...
```

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

```
    return "(" + str(self.x) + ", " + str(self.y) + ")«
```

```
punto1 = Punto2D(3,4)
```

```
print(punto1)
```

```
>>> (3, 4)
```

ejemplo1.py

Programación Orientada a Objetos

Nivelatorio para trabajar con data



Profesor José Tomás Marquinez V.