



# Programación. Python

## Clases y objetos

Cristóbal Pareja Flores :

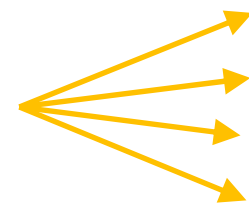
# Clases Y objetos



co\_cris = Coche()

```
class Coche(object)
```

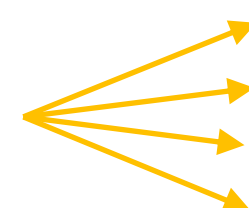
atributos



- Encendido/apagado
- Posición
- Orientación
- Velocidad

estado

métodos



- Encender/apagar
- Acelerar o frenar
- Girar
- Saber la velocidad

operaciones

# La clase Punto

```
▶ class Point(object):  
    def __init__(self):  
        self.x = 0.0  
        self.y = 0.0
```

```
▶ p0 = Point()  
p1 = Point()  
p1.x, p1.y = 4., 5.  
print (p0.x, p0.y)  
print (p1.x, p1.y)  
print(type(p0))
```

0.0 0.0

4.0 5.0

<class '\_\_main\_\_.Point'>

```
▶ from math import sqrt, pi  
  
class Point(object):  
    def __init__(self):  
        self.x = 0.0  
        self.y = 0.0  
    def dist_origen(self):  
        return sqrt(self.x**2 + self.y**2)  
  
p = Point()  
p.x, p.y = 12.0, 5.0  
print(p.dist_origen())
```

13.0

# La clase Punto

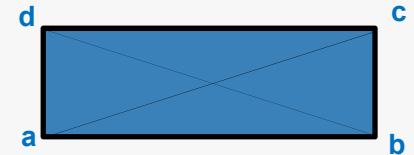
```
▶ class Point(object):  
    def __init__(self):  
        self.x = 0.0  
        self.y = 0.0
```

```
▶ p0 = Point()  
p1 = Point()  
p1.x, p1.y = 4., 5.  
print (p0.x, p0.y)  
print (p1.x, p1.y)  
print(type(p0))
```

```
0.0 0.0  
4.0 5.0  
<class '__main__.Point'>
```

```
from math import sqrt, pi  
  
def distancia(p0, p1):  
    return sqrt((p0.x - p1.x)**2 + (p0.y - p1.y)**2)
```

```
def es_rectangulo(a, b, c, d):  
    dab = distancia(a, b)  
    dac = distancia(a, c)  
    dad = distancia(a, d)  
    dbc = distancia(b, c)  
    dbd = distancia(b, d)  
    dcd = distancia(c, d)  
    return dab == dcd and dac == dbd and dad == dbc
```



```
p0, p1, p2, p3 = Point(), Point(), Point(), Point()
```

```
p0.x, p0.y = 0, 0  
p1.x, p1.y = 1, 1  
p2.x, p2.y = 0, 1  
p3.x, p3.y = 1, 0
```

```
es_rectangulo(p0, p1, p2, p3)
```

```
True
```

## Métodos especiales

```
class Point(object):
    """
    clase Point. Representa puntos en 2D

    Attributes
    -----
    x, y: float
    """
    def __init__(self, px, py):
        """
        Constructor

        Parameters
        -----
        x: float
        y: float
        """
        self.x = px
        self.y = py

    def __str__(self):
        """
        Este metodo devuelve el str que representa un Point
        """
        return '({0:.2f}, {1:.2f})'.format(self.x, self.y)
```

```
p0 = Point(3.0, 4.0)
print(p0)
p0
```

```
(3.00, 4.00)
```

```
<__main__.Point at 0x19bbbedee9e8>
```

```
p1 = Point(6.0, 0.0)
distancia(p0, p1)
```

```
5.0
```

```
➤ class Point(object):
    def __init__(self, px, py):
        self.x = px
        self.y = py

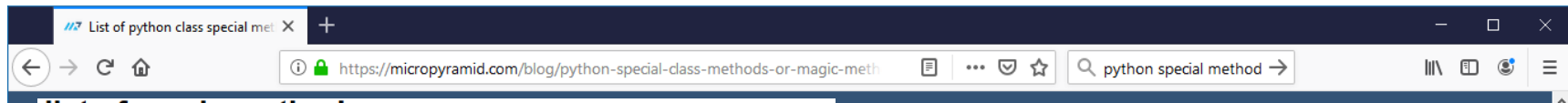
    def __str__(self):
        return 'Point(' + str(self.x) + ', ' + str(self.y) + ')'

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

    def move(self, t_x, t_y):
        self.x = self.x + t_x
        self.y = self.y + t_y
```

```
➤ p0 = Point(1.0, 2.0)
p1 = Point(7.0, 3.5)
print(p0)
print(p1)
print(p0.distance(p1))
p0.move(2.0, 4.0)
print(p0)
```

```
Point(1.0, 2.0)
Point(7.0, 3.5)
6.18465843842649
Point(3.0, 6.0)
```



## list of magic methods:

### Binary Operators

Operator	Method
+	object.__add__(self, other)
-	object.__sub__(self, other)
*	object.__mul__(self, other)
//	object.__floordiv__(self, other)
/	object.__div__(self, other)
%	object.__mod__(self, other)
**	object.__pow__(self, other[, modulo])
<<	object.__lshift__(self, other)
>>	object.__rshift__(self, other)
&	object.__and__(self, other)
^	object.__xor__(self, other)
	object.__or__(self, other)

### Assignment Operators:

Operator	Method
+=	object.__iadd__(self, other)
-=	object.__isub__(self, other)
*=	object.__imul__(self, other)
/=	object.__idiv__(self, other)
//=	object.__ifloordiv__(self, other)
%=	object.__imod__(self, other)
**=	object.__ipow__(self, other[, modulo])
<<=	object.__ilshift__(self, other)
>>=	object.__irshift__(self, other)
&=	object.__iand__(self, other)
^=	object.__ixor__(self, other)
=	object.__ior__(self, other)

### Unary Operators:

Operator	Method
-	object.__neg__(self)
+	object.__pos__(self)
abs()	object.__abs__(self)
~	object.__invert__(self)
complex()	object.__complex__(self)
int()	object.__int__(self)
long()	object.__long__(self)
float()	object.__float__(self)
oct()	object.__oct__(self)
hex()	object.__hex__(self)

### Comparison Operators

Operator	Method
<	object.__lt__(self, other)
<=	object.__le__(self, other)
==	object.__eq__(self, other)
!=	object.__ne__(self, other)
>=	object.__ge__(self, other)
>	object.__gt__(self, other)

```

class Persona(object):
    """
    Esta clase representa la ficua de una persona.
    No incluye el núm. de registro o el DNI,
    porque se asume que puede ser la clave de búsqueda
    en un diccionario.

    Attributes
    -----
    nombre: string
    edad: int
    estatura: float
    direccion: string
    """

    def __init__(self, nombre, edad, estatura, direccion):
        self.nombre = nombre
        self.edad = edad
        self.estatura = estatura
        self.direccion = direccion

    def __str__(self):
        #...
        return '<Nombre: ' + self.nombre + ', Edad: ' + str(self.edad) + ', ...>'

p = Persona("Blacky", 12, 0.30, "Carretera de Húmera, Pozuelo de Alarcón")
print(p)
p

```

<Nombre: Blacky, Edad: 12, ...>  
 Out[1]:  
 <\_\_main\_\_.Persona at 0x185b0ca69a0>



```
# Formamos ahora una agenda con un diccionario,  
# donde la clave es el número de registro:
```

```
mi_agenda = dict()  
mi_agenda["7023"] = p  
  
def mostrar_agenda(agenda):  
    for n in agenda:  
        print(n, agenda[n])  
  
mostrar_agenda(mi_agenda)
```

<

7023 <Nombre: Blacky, Edad: 12, ...>

*# Leemos el contenido de un archivo en la agenda*

```
def crear_agenda(nombre_archivo):
    la_agenda = dict()
    archivo = open(nombre_archivo, "r")
    for linea in archivo:
        # print(linea) # just for testing
        lin_limpia = linea.rstrip('\n')
        reg, nom, edad, estat, direcc = lin_limpia.split(" # ")
        edad = int(edad)
        estat = float(estat)
        # print(reg, nom, edad, estat, direcc) # just for testing
        la_agenda[reg] = Persona(nom, edad, estat, direcc)
    return la_agenda
```

```
ag = crear_agenda("agenda.txt")
mostrar_agenda(ag)
print(ag["023491"].direccion)
```

```
023491 <Nombre: Fernando, Edad: 22, ...>
324098 <Nombre: Elena    , Edad: 56, ...>
Pozuelo de Alarcón
```

*# Definimos una clase extendiendo la anterior: una clase derivada:*

```
class Familiar(Persona):

    def __init__(self, nombre, edad, estatura, direccion, par):
        Persona.__init__(self, nombre, edad, estatura, direccion)
        self.parentesco = par

    def __str__(self):
        return '<Nombre: ' + self.nombre + ', Edad: ' + str(self.edad) \
            + ', Parentesco: ' + str(self.parentesco) + '>'

    def es_humano(self):
        if self.parentesco == "Mascota":
            return False
        else:
            return True

f = Familiar("Blacky", 12, 0.30, \
            "Carretera de Húmera, Pozuelo de Alarcón", "Mascota")

print(f)

print(f.es_humano)
```

---

<Nombre: Blacky, Edad: 12, "Parentesco: Mascota>