Clases y Objetos

- Clases definidas por el usuario
  - galletas
  - películas
  - complejo, coche, fracción

POO

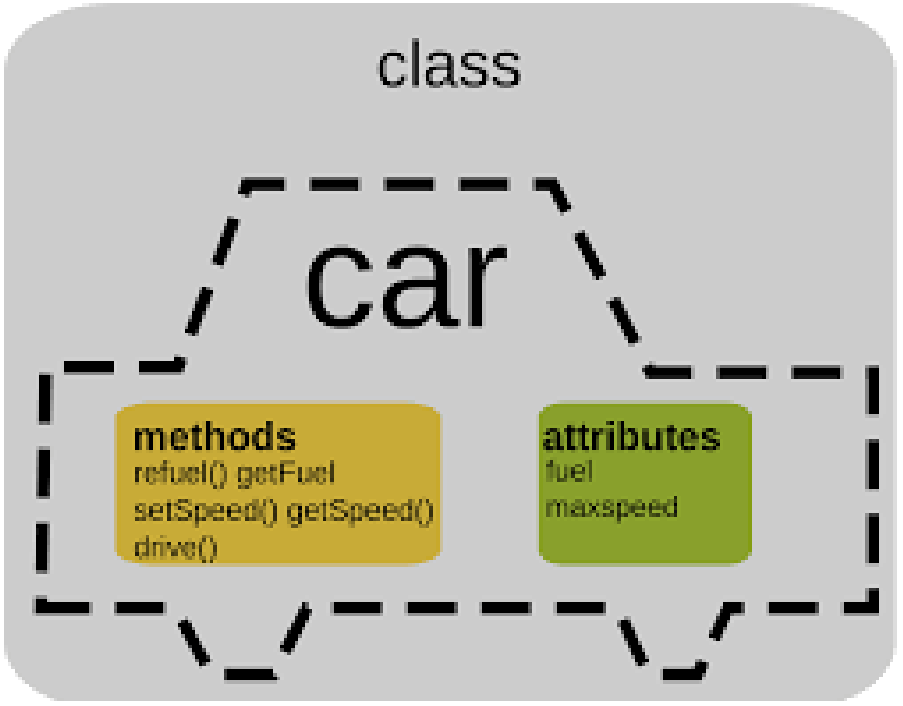
Paradigma de solución de problemas que identifica entidades de la realidad y las traslada a clases y objetos

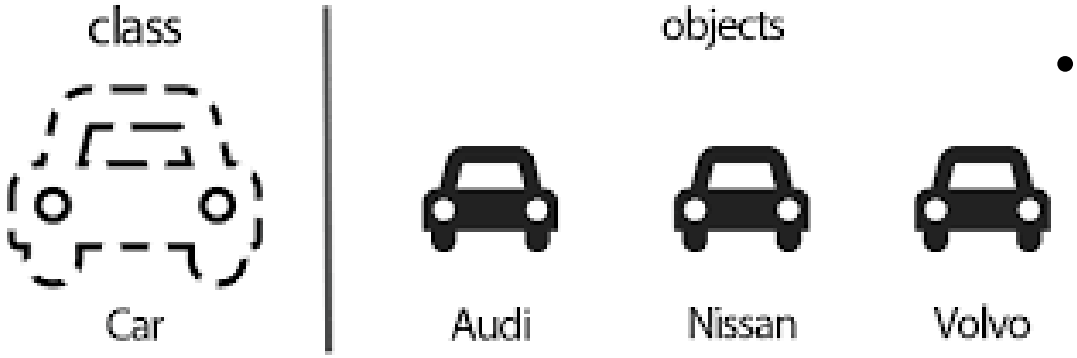
Respecto a la programación clásica:

- Más ágil
- Más intuitiva
- Más organizable
- Más escalable

Que es una clase ?

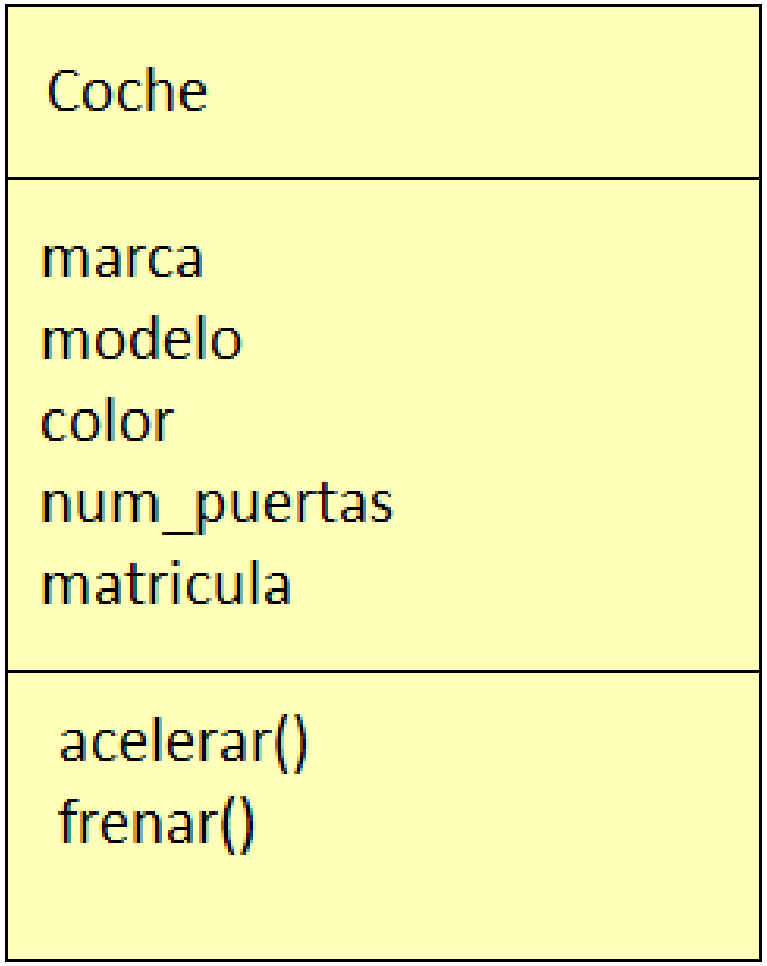
Un tipo de datos complejo, que puede incluir atributos y métodos.





- Nombre de la clase : Coche
- Atributos son características
  - Marca
  - Modelo
  - Color
  - Número de puertas
  - Matrícula
- Métodos son acciones
  - acelerar()
  - frenar()

Representación gráfica que aporta la persona que define el modelo el analista del sistema.



- Nombre de la clase : Coche
- Atributos o características
  - Marca
  - Modelo
  - Color
  - Número de puertas
  - Matrícula
- Métodos o acciones
  - acelerar()
  - frenar()

https://www.w3schools.com/python/python_classes.asp

Definir una clase en Python

```
#------------ definiciones

class Galleta :

    def __init__(self) :
        self.chocolate = False
        print ("Se ha creado una galleta")


    def mostrar_tipo(self) :
        if self.chocolate ==  False :
            print ("es una galleta sin chocolate")
        else:
            print ("es una galleta con chocolate")

#------------ inicio del Proceso
g = Galleta()
g.mostrar_tipo()
```

CLASE

OBJETO

```
Se ha creado una galleta
es una galleta sin chocolate
```

Clientes

Nombre
Apellidos
Sexo
Teléfono
Dirección
Ciudad
Estado

```
class Cliente:
    nombre = ""
    apellidos = ""
    sexo = ""
    telefono = ""
    direccion = ""
    ciudad = ""
    provincia = ""
```

Productos

Nombre
Descripción
Precio Unitario
Existencias

```
class Producto:
    nombre = ""
    descripcion = ""
    precio_unitario = ""
    existencias= ""
```

CLASE

```
#-------------------- PROCESO PRINCIPAL

cliente1 = Cliente()

cliente1.nombre = "Juan"
print (cliente1.nombre)
```

OBJETO

Definición de clases paso a paso

Crear una clase

Para crear una clase, use la palabra clave `class`:

Ejemplo

Cree una clase denominada MyClass, con una propiedad denominada x:

```
class MyClass:
  x = 5
```

Crear objeto

Ahora podemos usar la clase llamada MyClass para crear objetos:

Ejemplo

Cree un objeto llamado p1 e imprima el valor de x:

```
p1 = MyClass()
print(p1.x)
```

La función __init __ ()

Los ejemplos anteriores son clases y objetos en su forma más simple y no son realmente útiles en aplicaciones de la vida real.

Para comprender el significado de las clases, debemos comprender la función incorporada __init __ ().

Todas las clases tienen una función llamada __init __ (), que siempre se ejecuta cuando se inicia la clase.

Utilice la función __init __ () para asignar valores a las propiedades del objeto u otras operaciones que sean necesarias cuando se crea el objeto:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age


#------------------------ inicio
p1 = Person("John", 36)

print (p1.name)
print (p1.age)
```

Práctica P00 **Ejercicio 1**

- Crea una clase llamada Persona, usa la función __init __ () para asignar valores para el id, nombre, apellidos, correo y teléfono.

- Crea dos objetos de esa clase con datos diferentes.
- Imprímelos.

El parámetro self

El parámetro **self** es una referencia a la instancia actual de la clase y se usa para acceder a las variables que pertenecen a la clase.

No tiene porque llamarse **self**, puedes llamarlo como quieras, pero tiene que ser el primer parámetro de cualquier función en la clase:

```
class Person:
  def __init__(selfie, name, age):
    selfie.name = name
    selfie.age = age

#------------------------ inicio
p1 = Person("John", 36)

print (p1.name)
print (p1.age)
```

La declaración pass

Las definiciones no pueden estar vacías pero, si transitoriamente tienes una definición de clase sin contenido, coloca la palabra pass para evitar errores.

Ejemplo

```
class Person:
  pass
```

Práctica P00 **Ejercicio 2**

Completa la clase Persona realizada en el Ejercicio 1 con dos métodos:

- **enviar_correo (self, asunto, cuerpo)** que realiza
  - un print de la cadena “mailto: “ seguida del correo de la persona
  - y en otra línea “subject=“ seguido del asunto del mensaje
  - y en otra línea “body=“ seguido del cuerpo del mensaje

- **marcar_telf (self)** y dejaremos sin programar su contenido.

Un vuelo de pájaro del Tutorial de Hektor Profe

Apartado : atributos y Métodos

https://docs.hektorprofe.net/python/programacion-orientada-a-objetos/atributos-y-metodos/

Clases

Definición de una clase Galleta paso a paso

```
#------------- definiciones

class Galleta :
    pass

#------------- inicio del Proceso
g = Galleta()
print(type(g))
```

La palabra pass se usa para dejar la clase o sus funciones vacías de momento

Creación de la clase

Se crea una instancia de galleta o una galleta concreta

```
<class '__main__.Galleta'>
```

g es un objeto y una instancia de **Galleta**

```
#------------ definiciones

class Galleta :
    chocolate = False

#------------ inicio del Proceso
g = Galleta()
print(g.chocolate)
```

```
False
```

Constructor de la clase

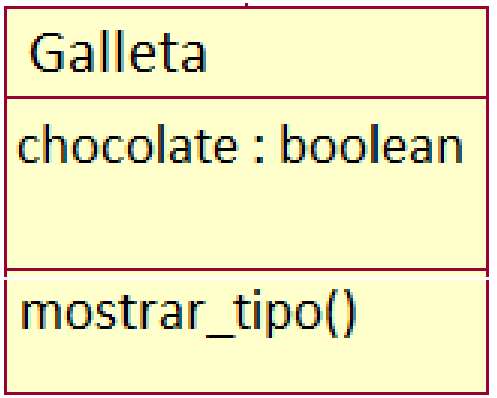
```
#------------ definiciones

class Galleta :

    def __init__(self) :
        self.chocolate = False
        print ("Se ha creado una galleta")

#------------ inicio del Proceso
g = Galleta()
```

```
Se ha creado una galleta
```

```python
#------------ definiciones

class Galleta :

    def __init__(self) :
        self.chocolate = False
        print ("Se ha creado una galleta")


    def mostrar_tipo(self) :
        if self.chocolate ==  False :
            print ("es una galleta sin chocolate")
        else:
            print ("es una galleta con chocolate")

#------------ inicio del Proceso
g = Galleta()
g.mostrar_tipo()
```

```
Se ha creado una galleta
es una galleta sin chocolate
```

Representación gráfica

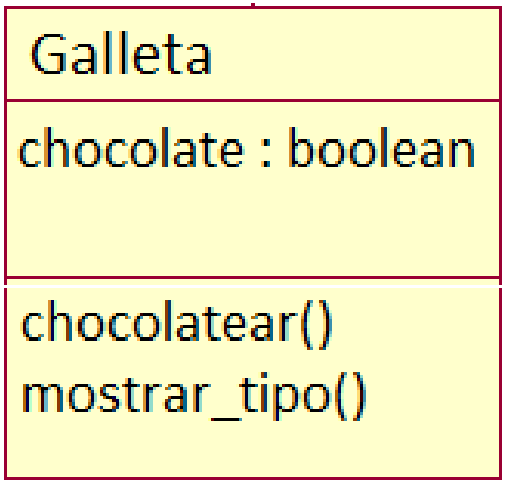


Método público

```python
#------------ definiciones

class Galleta :

    def __init__(self) :
        self.chocolate = False
        print ("Se ha creado una galleta")

    def chocolatear(self):
         self.chocolate = True

    def mostrar_tipo(self) :
        if self.chocolate ==  False :
            print ("es una galleta sin chocolate")
        else:
            print ("es una galleta con chocolate")

#------------ inicio del Proceso
g = Galleta()
g.mostrar_tipo()
g.chocolatear()
g.mostrar_tipo()
```

```
Se ha creado una galleta
es una galleta sin chocolate
es una galleta con chocolate
```

Representación gráfica



```
#------------ definiciones

class Galleta :

    def __init__(self,marca,forma ) :
        self.chocolate = False
        self.marca = marca
        self.forma = forma
        print ("Se ha creado una galleta {} {}".format(marca, forma))

    def chocolatear(self):
         self.chocolate = True

    def mostrar_tipo(self) :
        if self.chocolate ==  False :
            print ("es una galleta sin chocolate")
        else:
            print ("es una galleta con chocolate")

#------------ inicio del Proceso
g = Galleta()
g.mostrar_tipo()
g.chocolatear()
g.mostrar_tipo()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call
<ipython-input-37-aeffc4d40e07> in <module>
```

Añadir más atributos y parámetros en la función constructora

__init__()

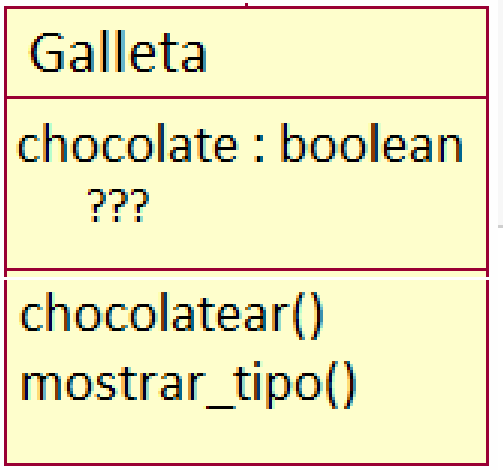
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-34-f64490986935> in <module>
     19
     20 #------------ inicio del Proceso
---> 21 g = Galleta()
     22 g.mostrar_tipo()
     23 g.chocolatear()

TypeError: __init__() missing 2 required positional arguments: 'marca' and 'forma'
```

```
class Galleta :

    def __init__(self,marca="lamia",forma="redona" ) :
        self.chocolate = False
        self.marca = marca
        self.forma = forma
        print ("Se ha creado una galleta")

#------------ inicio del Proceso
g = Galleta()
```

```
Se ha creado una galleta lamia redona
```

```python
#------------ definiciones

class Galleta :

    def __init__(self,marca,forma ) :
        self.chocolate = False
        self.marca = marca
        self.forma = forma
        print ("Se ha creado una galleta {} {}".format(self.marca, self.forma))

    def chocolatear(self):
         self.chocolate = True

    def mostrar_tipo(self) :
        if self.chocolate ==  False :
            print ("Es una galleta sin chocolate {} {}".format(self.marca, self.forma))
        else:
            print ("Es una galleta con chocolate {} {}".format(self.marca, self.forma))

#------------ inicio del Proceso
g = Galleta("Oreo", "redonda")
g.mostrar_tipo()
g.chocolatear()
g.mostrar_tipo()
```

```
Se ha creado una galleta Oreo redonda
Es una galleta sin chocolate Oreo redonda
Es una galleta con chocolate Oreo redonda
```



Práctica P01

Ejercicio 1

Pasa a tu cuaderno los distintos pasos de la creación de galleta que encontrarás en este enlace, verifica que comprendes casa paso ejecutándolo por separado.

https://docs.hektorprofe.net/python/programacion-orientada-a-objetos/atributos-y-metodos/

Ejercicio 2 : Avanzado

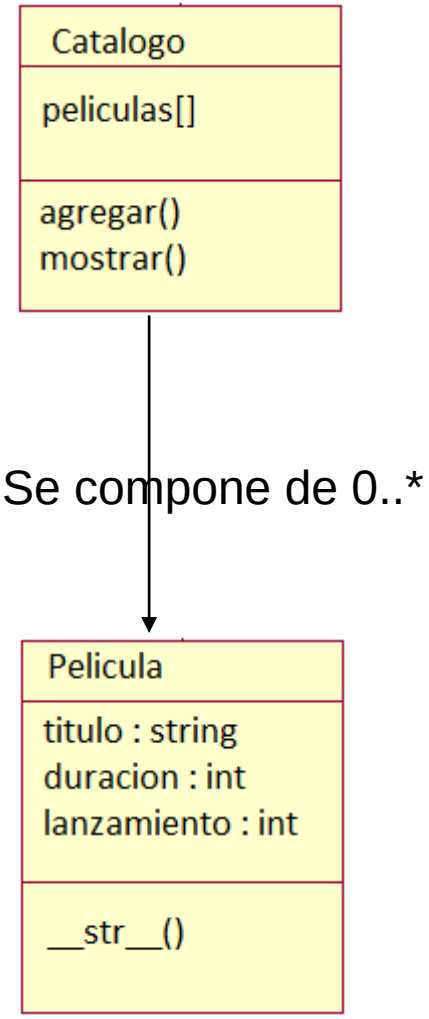
- Crea esta clase en Jupyter ,

```
class Complejo :

    def __init__(self, realpart, imagpart):
        self.r = realpart
        self.i = imagpart

    def imprimeReal(self):
        print ("parte real: ", self.r)
```

- Añade un método para mostrar la parte imaginaria
- Crea un objeto de tipo Complejo con los valores realpart=2 imagpart=3
- Después de crear el objeto usa su método para mostrar la parte real del objeto.

Objetos dentro de Objetos

Películas

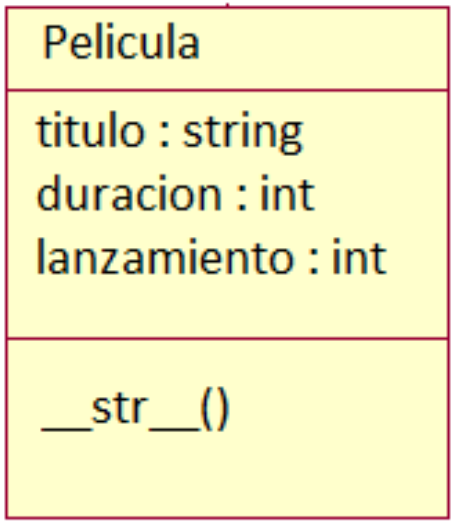


Vamos a crear un catálogo de películas

https://docs.hektorprofe.net/python/programacion-orientada-a-objetos/objetos-dentro-de-objetos/

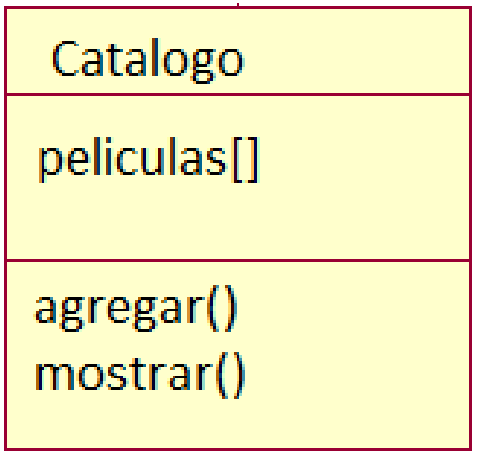
En el siguiente ejemplo de HektorProfe, se crea una clase llamada Película

- Tiene 3 atributos
- Y se define 1 método __str__() que sirve para modificar el comportamiento del print cuando se imprima este objeto.



En un paso posterior se crea la clase Catalogo que contiene varias películas

Se define como un array en el que se Insertan objetos de tipo Pelicula



Métodos especiales de clase
constructor

```
class Pelicula :
    # Constructor de clase
  def __init__(self, titulo, duracion, lanzamiento):
      self.titulo = titulo
      self.duracion = duracion
      self.lanzamiento = lanzamiento
      print("Se ha creado la película", self.titulo)
```

```
p = Pelicula("ET",115,1982)
```

```
Se ha creado la película ET
```

```
p = Pelicula("ET",115, 1982)
del(p)
```

destructor

```
Se ha creado la película ET
Se esta borrando la película ET
```

HOY no hace falta crear el **destructor**

```
p = Pelicula("ET",115, 1982)
```

```
Se ha creado la película ET
Se esta borrando la película ET
```

```
p = Pelicula("ET",115, 1982)
p.titulo
del(p)
```

```
Se ha creado la película ET
Se esta borrando la película ET
Se esta borrando la película ET
```

En jupyter el destructor de la clase, se dispara en cada ejecución, no es igual a ejecutar este programa desde otro entorno.

```
str(p)
```

```
'<__main__.Pelicula object at 0x000020BB8318FD0>'
```

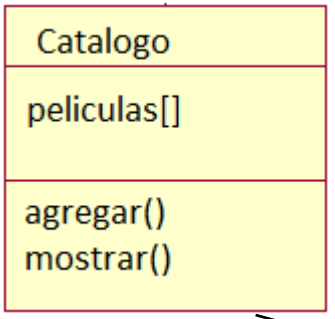
Es una referencia a una instancia del tipo película que está almacenada en esta dirección de memoria.

```
class Pelicula:
    # Constructor de clase
    def __init__(self,titulo,duracion,lanzamiento):
        self.titulo = titulo
        self.duracion = duracion
        self.lanzamiento = lanzamiento
        print("Se ha creado la película",self.titulo)

    # Redefinimos el método string
    def __str__(self):
        return "{} lanzada en {} con una duración de {} minutos".format(
                                        self.titulo,self.lanzamiento,self.duracion)
p = Pelicula("El Padrino",175,1972)
```

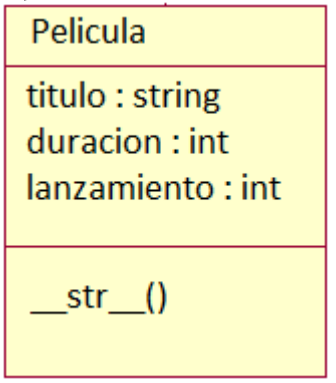
```
Se ha creado la película El Padrino
```

```
    # Redefinimos el método length
    def __len__(self):
        return self.duracion

p = Pelicula("El Padrino",175,1972)
len(p)
```

```
Se ha creado la película El Padrino
Se está borrando la película El Padrino

: 175
```



Se compone de 0..*



Representación de una composición de clases

Objetos dentro de Objetos

```
class Pelicula:

    # Constructor de clase
    def __init__(self, titulo, duracion, lanzamiento):
        self.titulo = titulo
        self.duracion = duracion
        self.lanzamiento = lanzamiento
        print('Se ha creado la película:', self.titulo)

    def __str__(self):
        return '{} ({})'.format(self.titulo, self.lanzamiento)
```

```
class Catalogo:

    peliculas = []  # Esta lista contendrá objetos de la clase Pelicula

def __init__(self,peliculas=[]):
    self.peliculas = peliculas

def agregar(self,p):
    self.peliculas.append(p)
```

```python
def mostrar(self):
    for p in self.peliculas:
        print(p)  # Print toma por defecto str(p)
```

con el método string y este print tomará por defecto el estreno de la película.

```python
p = Pelicula("El Padrino",175,1972)
c = Catalogo([p])
```

```
Se ha creado la película: El Padrino
```

```python
c.mostrar()
```

```
El Padrino (1972)
```

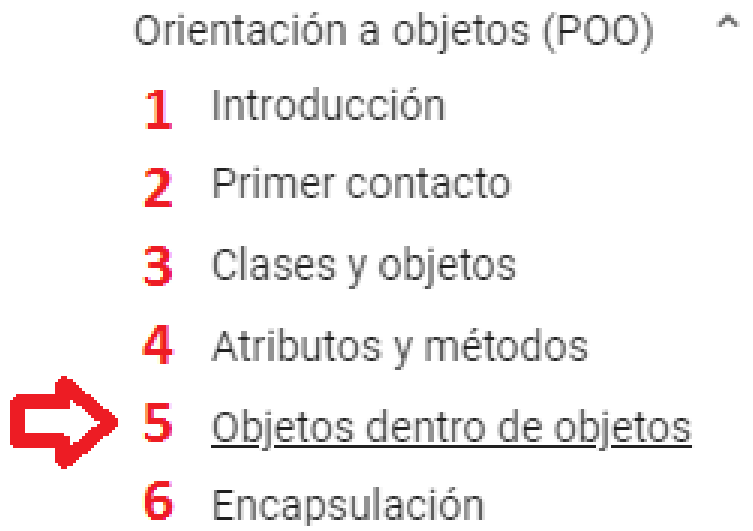
```python
c.agregar(Pelicula("El Padrino: Parte 2",202,1974))
```

```
Se ha creado la película: El Padrino: Parte 2
```

Práctica P02 **Ejercicio 1**

Repasa este capítulo Objetos dentro de Objetos del tutorial de Hektor Profe y verifica que entiendes los pasos (excepto el destructor de la clase, que mejor no usarlo en Júpiter Notebook).

https://docs.hektorprofe.net/python/programacion-orientada-a-objetos/objetos-dentro-de-objetos/



Practica P02 **Ejercicio 2**

Estudia este fragmento de código

[19]:
```
class Car:
      def __init__(self, marca, modelo, color, matricula):
          self.marca = marca
          self.modelo = modelo
          self.color = color
          self.matricula = matricula


c1 = Car ("TOYOTA", "YARIS","ROJO","6789-CYR")
c2 = Car ("TOYOTA", "YARIS","VERDE","333-CYR")

print (c1)
print (c1.marca)
```

```
<__main__.Car object at 0x000000000513B3C8>
TOYOTA
```

Práctica P02

Ejercicio 3 : Avanzado

Crea una clase que se llame Cars, que contenga 1 lista de coches, y dos funciones o métodos:

1. **listar_todos (self)**: Muestra toda la lista de coches con todos los datos
2. **busca_matricula (self, matricula)** : Muestra los datos de un solo coche Buscando por matrícula.

Ayuda para ->

Copiar-Pegar

```
class Car:
    def __init__(self, marca, modelo, color, matricula):
        self.marca = marca
        self.modelo = modelo
        self.color = color
        self.matricula = matricula


c1 = Car ("TOYOTA", "YARIS","ROJO","6789-CYR")
c2 = Car ("TOYOTA", "YARIS","VERDE","333-CYR")

print (c1)
print (c1.marca)
```

Práctica P03

Crea la siguiente clase :

Una vez creada la clase con sus atríbutos y métodos,

1. Crea un objeto de clase Tarjeta con los datos “123456-JK”, 1200
2. **Muestra el saldo**
3. **Realiza un pago** de 500 €, y otro de 9,25
4. Vuelve a **Mostrar el saldo**

https://aprendeconalf.es/docencia/python/manual/objetos/

Práctica P04- La clase Fracción

Define una clase para implementar el tipo abstracto de datos Fraccion.

Python proporciona una serie de clases numéricas para nuestro uso. Hay ocasiones en las que, sin embargo, sería más apropiado ser capaz de crear objetos de datos que “aparezcan” como fracciones.

Una fracción como 3 /5 consta de dos partes.
El valor de arriba, conocido como el numerador, puede ser cualquier entero.
El valor de abajo, llamado el denominador, puede ser cualquier entero mayor que 0 (las fracciones negativas tienen un numerador negativo).

Aunque es posible crear una aproximación de punto flotante para cualquier fracción, en este caso nos gustaría representar la fracción como un valor expresado con enteros : implementa el método __str__

Implementa la suma y resta de fraciones. Repasa esta web

https://uniwebsidad.com/libros/algoritmos-python/capitulo-14/metodos-especiales