Programación orientada a objetos

Es un paradigma de la programacion que permite tener codigos flexibles que pueden crecer indefinidamente y reutilizar los codigos desarrollados.
Desde el punto de vista del debugging localiza la propagacion de los errores.
Objetos son conjuntos de variables y de metodos que operan sobre esas variables.

Elementos de una clase

```
class Auto:
   __init__(self,color='',posx=0,posy=0):

      # Posicion initial
      self.posx=posx
      self.posy=posy

   mover(self,dx=0,dy=0):
      self.posx=self.posx+dx
      self.posy=self.posy+dy
```

Instanciacion de una clase

Se crean instancias de la clase o se definen objetos de la clase al llamar a esta:

```
citroen=Auto(color='amarillo',posx=14.0,posy=2.0)
```

Cada objeto es llamado una instancia de la clase. Tambien se dice que se instancia cuando se crea la clase.

Cada clase se puede instanciar cuantas veces se quiera, entre estas son totalmente independientes:

```
clio=Auto(color='gris',posx=20.0,posy=5.0)
```

Creacion del objeto

Cuando se hace referencia al objeto Python va a crear la instancia y llama a

```
Auto.__init__
```

Este es el "inicializador" de la clase. Alli se ponen los parametros que se quieren definir por default.

self

```
__init__(self,color='',posx=0,posy=0):
```

Notar la presencia de self. Esto es obligatorio en todos los metodos de una clase y es una forma de dejar explicito que los atributos metodo estan compartiendo entre si las variables y los metodos de la clase.
Se podria utilizar otro nombre en lugar de 'self' pero es conveniente usarlo de esa manera ya que es el default.
Desde cualquier lugar de la clase podemos acceder a las variables de la clase y/o los metodos de la clase:
self.posx me dice la posicion del auto. Es decir es como que tenemos variables "modulares" son compartidas por todas las funciones de la clase. Estas son distintas de las variables locales de cada funcion y de las variables globales.

Referencias a los atributos de una clase

Con el nombre de la instancia punto y el atributo se puede acceder a cualquier atributo de la clase.

```
print 'El citroen es de color: ', citroen.color
```

En este caso citroen.color es un atributo tipo dato de la clase.
También se puede acceder a las funciones de la clase que es un atributo método

```
clio.mover(10,5)
print 'La posicion actual del clio es:',clio.posx,clio.po
```

Herencia

Los atributos de una clase se pueden heredar.

Supongamos que tenemos la clase los polinomios de grado n.

```
class polinomios:

    __init__
```