04 Funciones definidas por el usuario

Una función, es la forma de agrupar expresiones y sentencias (algoritmos) que realicen determinadas acciones, pero que éstas, solo se ejecuten cuando son llamadas. Es decir, que al colocar un algoritmo dentro de una función, al correr el archivo, el algoritmo no será ejecutado si no se ha hecho una referencia a la función que lo contiene.

Definiendo funciones

En Python, la definición de funciones se realiza mediante la instrucción `def` más un nombre de función descriptivo -para el cuál, aplican las mismas reglas que para el nombre de las variables- seguido de paréntesis de apertura y cierre. Como toda estructura de control en Python, la definición de la función finaliza con dos puntos (:) y el algoritmo que la compone, irá identado con 4 espacios:

```
def mi_funcion():
    # aquí el algoritmo
```

Una función, no es ejecutada hasta tanto no sea invocada. Para invocar una función, simplemente se la llama por su nombre:

```
def mi_funcion():
    print "Hola Mundo"

funcion()
```

Cuando una función, haga un **retorno de datos**, éstos, pueden ser asignados a una variable:

```
def funcion():
    return "Hola Mundo"

frase = funcion()
print frase
```

Sobre los parámetros

Un parámetro es un valor que la función espera recibir cuando sea llamada (invocada), a fin de ejecutar acciones en base al mismo. Una función puede esperar uno o más parámetros (que irán separados por una coma) o ninguno.

```
def mi_funcion(nombre, apellido):
    # algoritmo
```

> *Los parámetros, **se indican** entre los paréntesis, **a modo de variables**, a fin de poder utilizarlos como tales, dentro de la misma función.*

Los parámetros que una función espera, serán utilizados por ésta, dentro de su algoritmo, a modo de **variables de ámbito local**. Es decir, que los parámetros serán variables locales, a las cuáles solo la función podrá acceder:

```
def mi_funcion(nombre, apellido):
    nombre_completo = nombre, apellido
    print nombre_completo
```

Si quisiéramos acceder a esas variables locales, fuera de la función, obtendríamos un error:

```
def mi_funcion(nombre, apellido):
    nombre_completo = nombre, apellido
    print nombre_completo

print nombre  # Retornará el error: NameError: name 'nombre' is not defined
```

> *Al llamar a una función, **siempre se le deben pasar sus argumentos en el mismo orden en el que los espera**. Pero esto **puede evitarse**, haciendo uso del paso de argumentos como keywords (ver más abajo: "**Keywords como parámetros**").*

Parámetros por omisión

En Python, también es posible, asignar valores por defecto a los parámetros de las funciones. Esto significa, que la función podrá ser llamada con menos argumentos de los que espera:

```
def saludar(nombre, mensaje='Hola'):
    print mensaje, nombre

saludar('Pepe Grillo')  # Imprime: Hola Pepe Grillo
```

> **PEP 8: Funciones**
> A la definición de una función la deben anteceder dos líneas en blanco.
> Al asignar parámetros por omisión, no debe dejarse espacios en blanco ni antes ni después del signo =.

Keywords como parámetros

En Python, también es posible llamar a una función, pasándole los argumentos esperados, como pares de `claves=valor`:

```
def saludar(nombre, mensaje='Hola'):
    print mensaje, nombre
```

```
saludar(mensaje="Buen día", nombre="Juancho")
```

Parámetros arbitrarios

Al igual que en otros lenguajes de alto nivel, es posible que una función, espere recibir un número arbitrario -desconocido- de argumentos. Estos argumentos, llegarán a la función en forma de tupla.

Para definir argumentos arbitrarios en una función, se antecede al parámetro un asterisco (*):

```
def recorrer_parametros_arbitrarios(parametro_fijo, *arbitrarios):
    print parametro_fijo

    # Los parámetros arbitrarios se corren como tuplas
    for argumento in arbitrarios:
        print argumento

recorrer_parametros_arbitrarios('Fixed', 'arbitrario 1', 'arbitrario 2',
                                'arbitrario 3')
```

> *Si una función espera recibir parámetros fijos y arbitrarios,* ***los arbitrarios siempre deben suceder a los fijos.***

Es posible también, obtener parámetros arbitrarios como pares de clave=valor. En estos casos, al nombre del parámetro deben precederlo dos astericos (**):

```
def recorrer_parametros_arbitrarios(parametro_fijo, *arbitrarios, **kwords):
    print parametro_fijo
    for argumento in arbitrarios:
        print argumento

    # Los argumentos arbitrarios tipo clave, se recorren como los diccionarios
    for clave in kwords:
        print "El valor de", clave, "es", kwords[clave]

recorrer_parametros_arbitrarios("Fixed", "arbitrario 1", "arbitrario 2",
                                "arbitrario 3", clave1="valor uno",
                                clave2="valor dos")
```

Desempaquetado de parámetros

Puede ocurrir además, una situación inversa a la anterior. Es decir, que la función espere una lista fija de parámetros, pero que éstos, en vez de estar disponibles de forma separada, se encuentren contenidos en una lista o tupla. En este caso, el signo asterisco (*) deberá preceder al nombre de la lista o tupla que es pasada como parámetro durante la llamada a la función:

```
def calcular(importe, descuento):
    return importe - (importe * descuento / 100)

datos = [1500, 10]
print calcular(*datos)
```

El mismo caso puede darse cuando los valores a ser pasados como parámetros a una función, se encuentren disponibles en un diccionario. Aquí, deberán pasarse a la función, precedidos de dos asteriscos (**):

```
def calcular(importe, descuento):
    return importe - (importe * descuento / 100)

datos = {"descuento": 10, "importe": 1500}
print calcular(**datos)
```

Llamadas de retorno

En Python, es posible (al igual que en la gran mayoría de los lenguajes de programación), llamar a una función dentro de otra, de forma fija y de la misma manera que se la llamaría, desde fuera de dicha función:

```
def funcion():
    return "Hola Mundo"

def saludar(nombre, mensaje='Hola'):
    print mensaje, nombre
    print mi_funcion()
```

Sin embargo, es posible que se desee **realizar dicha llamada, de manera dinámica**, es decir, **desconociendo el nombre de la función** a la que se deseará llamar. A este tipo de acciones, se las denomina **llamadas de retorno**.

Para conseguir llamar a una función de manera dinámica, Python dispone de dos funciones nativas: **`locals()`** y **`globals()`**

Ambas funciones, retornan un diccionario. En el caso de `locals()`, éste diccionario se compone -justamente- de todos los elementos de ámbito local, mientras que el de `globals()`, retorna lo propio pero a nivel global.

```
def funcion():
    return "Hola Mundo"


def llamada_de_retorno(func=""):
    """Llamada de retorno a nivel global"""
    return globals()[func]()
```

```
print llamada_de_retorno("funcion")

# Llamada de retorno a nivel local
nombre_de_la_funcion = "funcion"
print locals()[nombre_de_la_funcion]()
```

Si se tienen que pasar argumentos en una llamada retorno, se lo puede hacer normalmente:

```
def funcion(nombre):
    return "Hola " + nombre


def llamada_de_retorno(func=""):
    """Llamada de retorno a nivel global"""
    return globals()[func]("Laura")

print llamada_de_retorno("funcion")

# Llamada de retorno a nivel local
nombre_de_la_funcion = "funcion"
print locals()[nombre_de_la_funcion]("Facundo")
```

Saber si una función existe y puede ser llamada

Durante una llamada de retorno, el nombre de la función, puede no ser el indicado. Entonces, siempre que se deba realizar una llamada de retorno, es necesario comprobar que ésta exista y pueda ser llamada.

```
if nombre_de_la_funcion in locals():
    if callable(locals()[nombre_de_la_funcion]):
        print locals()[nombre_de_la_funcion]("Emilse")
```

El operador `in`, nos permitirá conocer si un elemento se encuentra dentro de una colección, mientras que la función `callable()` nos dejará saber si esa función puede ser llamada.

```
def funcion(nombre):
    return "Hola " + nombre


def llamada_de_retorno(func=""):
    if func in globals():
        if callable(globals()[func]):
            return globals()[func]("Laura")
    else:
        return "Función no encontrada"

print llamada_de_retorno("funcion")


nombre_de_la_funcion = "funcion"
```

```
if nombre_de_la_funcion in locals():
    if callable(locals()[nombre_de_la_funcion]):
        print locals()[nombre_de_la_funcion]("Facundo")
else:
    print "Función no encontrada"
```

Llamadas recursivas

Se denomina llamada recursiva (o recursividad), a aquellas funciones que en su algoritmo, hacen referencia sí misma.

Las llamadas recursivas suelen ser muy útiles en casos muy puntuales, pero debido a su gran factibilidad de caer en iteraciones infinitas, deben extremarse las medidas preventivas adecuadas y, solo utilizarse cuando sea estrictamente necesario y no exista una forma alternativa viable, que resuelva el problema evitando la recursividad.

Python admite las llamadas recursivas, permitiendo a una función, llamarse a sí misma, de igual forma que lo hace cuando llama a otra función.

```
def jugar(intento=1):
    respuesta = raw_input("¿De qué color es una naranja? ")
    if respuesta != "naranja":
        if intento < 3:
            print "\nFallaste! Inténtalo de nuevo"
            intento += 1
            jugar(intento)  # Llamada recursiva
        else:
            print "\nPerdiste!"
    else:
        print "\nGanaste!"

jugar()
```

Sobre la finalidad de las funciones

Una función, puede tener cualquier tipo de algoritmo y cualquier cantidad de ellos y, utilizar cualquiera de las características vistas hasta ahora. No obstante ello, **una buena práctica, indica que la finalidad de una función, debe ser realizar una única acción, reutilizable y por lo tanto, tan genérica como sea posible**.

05 Introducción a la Orientación a Objetos

En Python todo es un “objeto” y debe ser manipulado -y entendido- como tal. Pero ¿Qué es un objeto? ¿De qué hablamos cuando nos referimos a “orientación a objetos? En este capítulo, haremos una introducción que responderá a estas -y muchas otras- preguntas.

Nos enfocaremos primero, en cuestiones de conceptos básicos, para luego, ir introduciéndonos de a poco, en principios teóricos elementalmente necesarios, para implementar la orientación a objetos en la práctica.

Pensar en objetos

Pensar en objetos, puede resultar -al inicio- una tarea difícil. Sin embargo, difícil no significa complejo. Por el contrario, pensar en objetos representa la mayor simplicidad que uno podría esperar del mundo de la programación. **Pensar en objetos, es simple**... aunque lo simple, no necesariamente signifique sencillo.

Y ¿qué es un objeto?

Pues, como dije antes, es "simple". Olvidemos los formalismos, la informática y todo lo que nos rodea. Simplemente, olvida todo y concéntrate en lo que sigue. Lo explicaré de manera "simple":

Un objeto es "una cosa". Y, si una cosa es un sustantivo, entonces **un objeto es un sustantivo**.

Mira a tu alrededor y encontrarás decenas, cientos de objetos. Tu ordenador, es un objeto. Tú, eres un objeto. Tu llave es un objeto. El cenicero (ese que tienes frente a ti cargado de colillas de cigarrillo), es otro objeto. Tu mascota también es un objeto.

> *Cuando pensamos en "objetos", **todos los sustantivos son objetos.***

Sencillo ¿cierto? Entonces, de ahora en más, solo concéntrate en pensar la vida en objetos (al menos, hasta terminar de leer este documento).

Ahora ¿qué me dices si describimos las cualidades de un objeto?

Describir un objeto, es simplemente mencionar sus cualidades. **Las cualidades son adjetivos**. Si no sabes que es un adjetivo, estamos jodidos (y mucho). Pero, podemos decir que **un adjetivo es una cualidad del sustantivo**.

Entonces, para describir "la manera de ser" de un objeto, debemos preguntarnos **¿cómo es el objeto?** Toda respuesta que comience por "el objeto es", seguida de un adjetivo, será una cualidad del objeto.

Algunos ejemplos:

- **El objeto es** verde
- **El objeto es** grande

- **El objeto es** feo

Ahora, imagina que te encuentras frente a un niño de 2 años (niño: objeto que pregunta cosas que tú das por entendidas de forma implícita). Y cada vez que le dices las cualidades de un objeto al ~~molesto~~ niño-objeto, éste te pregunta: -"¿Qué es...?", seguido del adjetivo con el cuál finalizaste tu frase. Entonces, tu le respondes diciendo "es un/una" seguido de un sustantivo. Te lo muestro con un ejemplo:

- El objeto es verde. **¿Qué es** verde? Un color.
- El objeto es grande. **¿Qué es** grande? Un tamaño.
- El objeto es feo. **¿Qué es** feo? Un aspecto.

Estos sustantivos que responden a la pregunta del niño, pueden pasar a formar parte de una **locución adjetiva** que especifique con mayor precisión, las descripciones anteriores:

- El objeto es **de *color*** verde.
- El objeto es **de *tamaño*** grande.
- El objeto es **de *aspecto*** feo.

Podemos decir entonces -y todo esto, gracias al ~~molesto~~ niño-objeto-, que una cualidad, es un atributo (derivado de "cualidad *atribuible* a un objeto") y que entonces, **un objeto es un sustantivo que posee atributos, cuyas cualidades lo describen.**

Veámoslo más gráficamente:

OBJETO (sustantivo)	ATRIBUTO (locución adjetiva)	CUALIDAD DEL ATRIBUTO (adjetivo)
(el) Objeto	(es de) color	Verde
	(es de) tamaño	Grande
	(es de) aspecto	Feo

Pero algunos objetos, también se componen de otros objetos...

Además de cualidades (locución adjetiva seguida de un adjetivo), **los objetos "tienen otras cosas"**. Estas "otras cosas", son aquellas "pseudo-cualidades" que en vez de responder a ¿cómo es el objeto? responden a **"¿cómo está compuesto el objeto?"** o incluso, aún más simple **"¿Qué tiene el objeto?".**

La respuesta a esta pregunta, estará dada por la frase "el objeto tiene...", seguida de un adverbio de cantidad (uno, varios, muchos, algunos, unas cuantas) y un sustantivo.

Algunos ejemplos:

- El objeto **tiene *algunas*** antenas
- El objeto **tiene *un*** ojo
- El objeto **tiene *unos cuantos*** pelos

Los componentes de un objeto, también integran los atributos de ese objeto. Solo que **estos atributos**, son algo particulares: **son otros objetos que poseen sus propias cualidades**. Es decir, que estos "atributos-objeto" también responderán a la pregunta "¿Cómo es/son ese/esos/esas?" seguido del atributo-objeto (sustantivo).

Ampliemos el ejemplo para que se entienda mejor:

- El objeto tiene algunas antenas. ¿***Cómo son esas*** antenas?
  - Las antenas son de color violeta
  - Las antenas son de longitud extensa
- El objeto tiene un ojo. ¿***Cómo es ese*** ojo?
  - El ojo es de forma oval
  - El ojo es de color azul
  - El ojo es de tamaño grande
- El objeto tiene unos cuantos pelos. ¿***Cómo son esos*** pelos?
  - Los pelos son de color fucsia
  - Los pelos son de textura rugosa

Pongámoslo más gráfico:

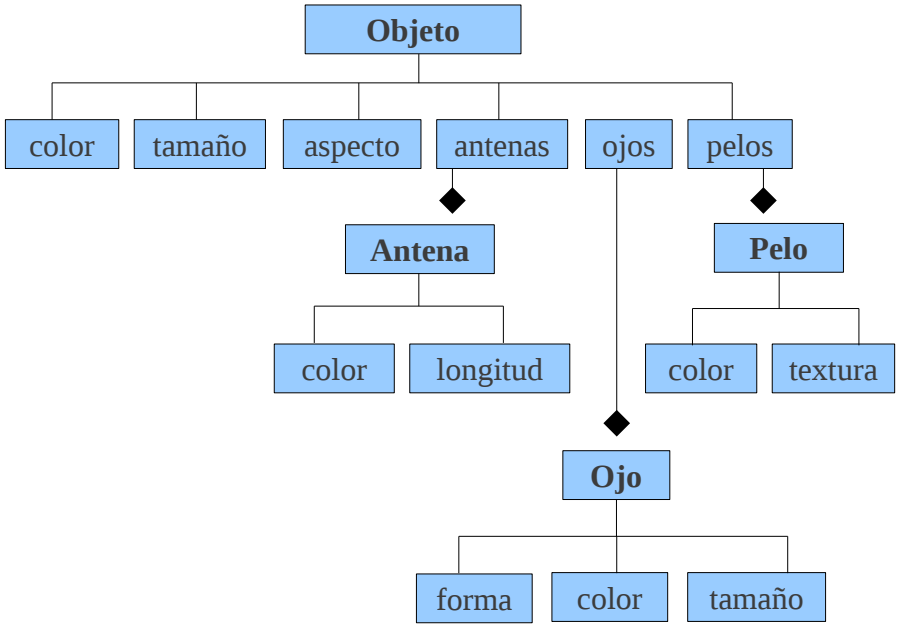
OBJETO (sustantivo)	**ATRIBUTO-OBJETO (sustantivo)**	**ATRIBUTOS (locución adjetiva)**	**CUALIDADES DE LOS ATRIBUTOS (adjetivo)**
(el) Objeto	(tiene algunas) antenas	(de) color (de) longitud	Violeta extensa
	(tiene un) ojo	(de) forma (de) color (de) tamaño	Oval azul grande
	(tiene unos cuantos) pelos	(de) color (de) textura	Fucsia rugosa

Entonces, podemos deducir que **un objeto puede tener dos tipos de atributos**:

1) Los que responden a la pregunta ***"¿Cómo es el objeto?"*** con la frase ***"El objeto es..."*** + adjetivo (atributos definidos por cualidades)
2) Los que responden a la pregunta ***"¿Qué tiene el objeto?"*** con la frase ***"El objeto***

tiene...” + sustantivo (atributos definidos por las cualidades de otro objeto)

Veámoslo aún, más gráficamente:



Viendo el gráfico anterior, tenemos lo siguiente: *Un **objeto** (sustantivo) al cual hemos descrito con tres **atributos** (adjetivos) y otros tres **atributos-objeto** (sustantivos) los cuáles son a la vez, otros tres objetos (sustantivos) con sus atributos (adjetivos) correspondientes. ¿Simple, no? Ahora, compliquemos todo un poco.*

Y también hay objetos que comparten características con otros objetos

*Resulta ser, que nuestro Objeto, es prácticamente igual a un nuevo objeto. Es decir, que el nuevo objeto que estamos viendo, tiene absolutamente todas las características que nuestro primer objeto, es decir, tiene los mismos atributos. Pero también, tiene algunas más. Por ejemplo, este **nuevo objeto**, además de los atributos de nuestro primer objeto, **tiene un pie**. Es decir, que las características de nuestro nuevo objeto, serán todas las del objeto original, más una nueva: pie.*

Repasemos las características de nuestro nuevo objeto:

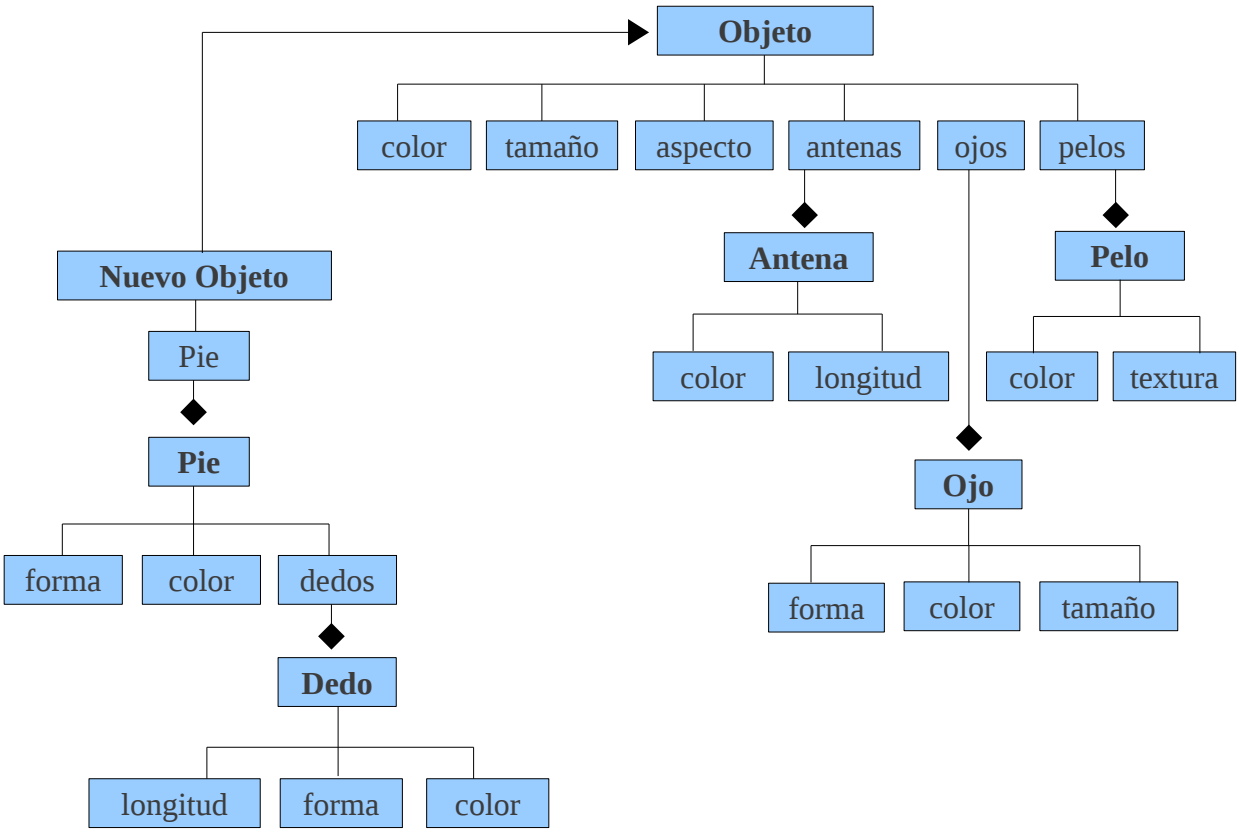
- El nuevo objeto es de color verde.
- El nuevo objeto es de tamaño grande.
- El nuevo objeto es de aspecto feo.

- El nuevo objeto tiene algunas antenas. ¿Cómo son esas antenas?
  - Las antenas son de color violeta
  - Las antenas son de longitud extensa
- El nuevo objeto tiene un ojo. ¿Cómo es ese ojo?
  - El ojo es de forma oval
  - El ojo es de color azul
  - El ojo es de tamaño grande
- El nuevo objeto tiene unos cuantos pelos. ¿Cómo son esos pelos?
  - Los pelos son de color fucsia
  - Los pelos son de textura rugosa

(nuevas características)

- El nuevo objeto tiene un pie. ¿Cómo es ese pie?
  - El pie es de forma rectangular
  - El pie es de color amarillo
  - El pie tiene 3 dedos. ¿Cómo son esos dedos?
    - Los dedos son de longitud mediana
    - Los dedos son de forma alargada
    - Los dedos son de color amarillo

Veamos todas las características de este nuevo, en un gráfico como lo hicimos antes.



Con mucha facilidad, podemos observar como nuestro nuevo objeto es una especie de "objeto original ampliado". Es decir que el nuevo objeto, es exactamente igual al objeto original (comparte todos sus atributos) pero posee nuevas características.

Está claro además, que el objeto original y el nuevo objeto, son dos objetos diferentes ¿cierto? No obstante, **el nuevo objeto es un sub-tipo del objeto original**.

Ahora sí, a complicarnos aún más.

Los objetos, también tienen la capacidad de "hacer cosas"

Ya describimos las cualidades de nuestros objetos. Pero de lo que no hemos hablado, es de aquellas cosas que los objetos "pueden hacer", es decir, "cuáles son sus capacidades".

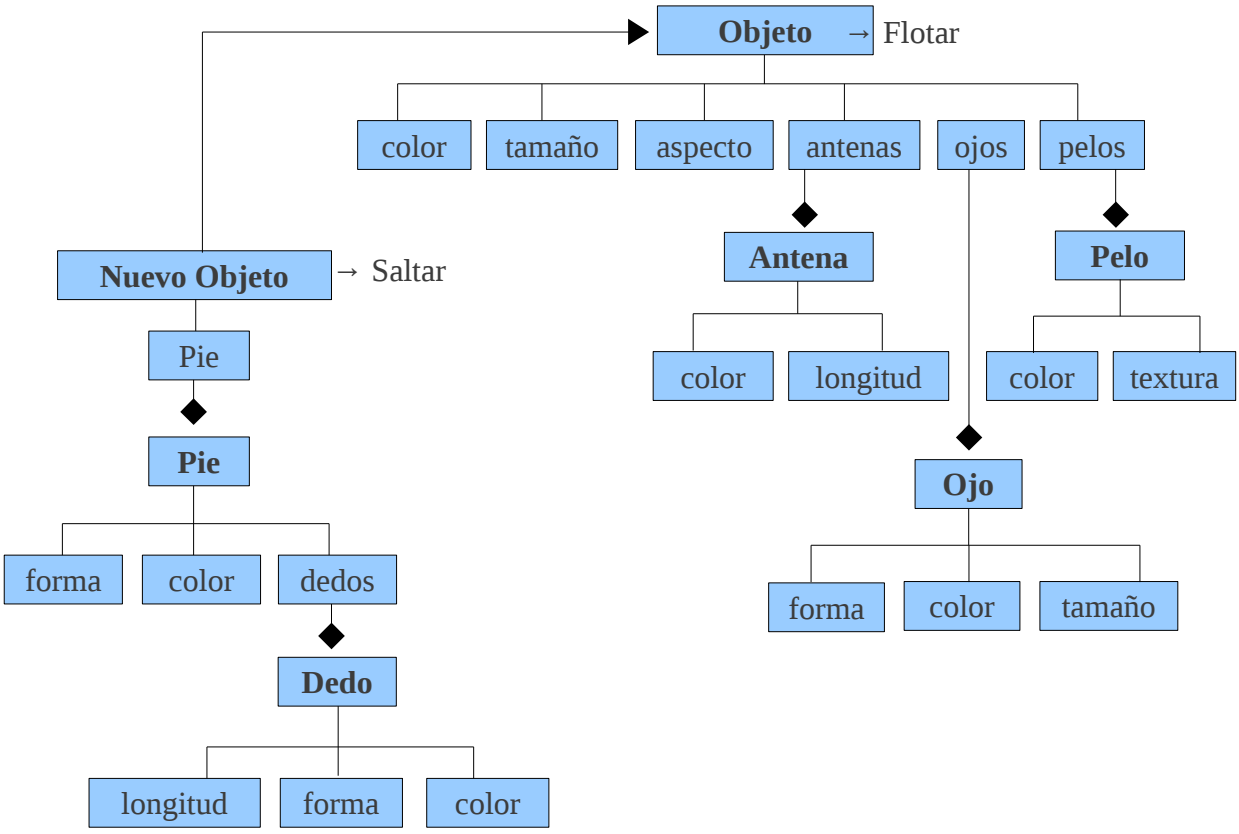
Los objetos tiene la capacidad de realizar acciones. Las acciones, son verbos. Es decir, que para conocer las capacidades de un objeto, debes preguntarte **"¿Qué puede hacer el objeto?"** y la respuesta a esta pregunta, estará dada por todas aquellas que comiencen por la frase "el objeto puede" seguida de un verbo en infinitivo.

Algunos ejemplos:

– El objeto original **puede** flotar

– El nuevo objeto (además) **puede** saltar

Si completamos el gráfico anterior con las acciones, obtendremos lo siguiente:



Si observas el gráfico anterior, notarás que el nuevo objeto, no solo tiene los mismos atributos que el objeto original, sino que además, también puede realizar las mismas acciones que éste. Sencillo, cierto?

Ahora sí, compliquémonos del todo :)

Objetos y más objetos: la parte difícil

Si entendiste todo lo anterior, ahora viene la parte difícil. ¿Viste que esto de “pensando en objetos” viene a colación de la programación orientada a objetos? Bueno, la parte difícil es que en la programación, todo lo que acabamos de ver, se denomina de una forma particular. Pero, la explicación es la misma que te di antes.

Al pan, pan. Y al vino, vino. Las cosas por su nombre

Cuando en el documento...	En la programación se denomina...	Y con respecto a la programación orientada a objetos es...
Hablamos de “objeto”	**Objeto**	Un elemento
Hablamos de “atributos” (o cualidades)	**Propiedades**	Un elemento
Hablamos de “acciones” que puede realizar el objeto	**Métodos**	Un elemento
Hablamos de “atributos-objeto”	**Composición**	Una técnica
Vemos que los objetos relacionados entre sí, tienen nombres de atributos iguales (por ejemplo: color y tamaño) y sin embargo, pueden tener valores diferentes	**Polimorfismo**	Una característica
Hablamos de objetos que son sub-tipos (o ampliación) de otros	**Herencia**	Una característica

Ahora, pasemos a un marco un poco más “académico”.

Programación Orientada a Objetos

La Programación Orientada a Objetos (POO u OOP por sus siglas en inglés), es un **paradigma de programación**.

> ***Paradigma*: teoría cuyo núcleo central** *[...]* **suministra la base y modelo para resolver problemas** *[...] Definición de la Real Academia Española, vigésimo tercera edición*

Cómo tal, nos enseña un método -probado y estudiado- el cual se basa en las interacciones de objetos (todo lo descrito en el título anterior, “Pensar en objetos”) para resolver las necesidades de un sistema informático.

Básicamente, este paradigma se compone de 6 elementos y 7 características que veremos a continuación.

Elementos y Características de la POO

Los **elementos** de la POO, pueden entenderse como los ***“materiales”*** que necesitamos para diseñar y programar un sistema, mientras que las **características**, podrían asumirse como las ***“herramientas”*** de las cuáles disponemos para construir el sistema con esos materiales.

Entre los **elementos principales** de la POO, podremos encontrar a:

Clases

Las clases son los modelos sobre los cuáles se construirán nuestros objetos. Podemos tomar como ejemplo de clases, el gráfico que hicimos en la página 8 de este documento.

En Python, una clase se define con la instrucción `class` seguida de un nombre genérico para el objeto.

```
class Objeto:
    pass


class Antena:
    pass


class Pelo:
```

```
    pass


class Ojo:
    pass
```

> **PEP 8: clases**
> El nombre de las clases se define en singular, utilizando CamelCase.

Propiedades

Las propiedades, como hemos visto antes, son las características intrínsecas del objeto. Éstas, se representan a modo de variables, solo que técnicamente, pasan a denominarse "propiedades":

```
class Antena():
    color = ""
    longitud = ""


class Pelo():
    color = ""
    textura = ""


class Ojo():
    forma = ""
    color = ""
    tamanio = ""


class Objeto():
    color = ""
    tamanio = ""
    aspecto = ""
    antenas = Antena()  # propiedad compuesta por el objeto objeto Antena
    ojos = Ojo()        # propiedad compuesta por el objeto objeto Ojo
    pelos = Pelo()      # propiedad compuesta por el objeto objeto Pelo
```

> **PEP 8: propiedades**
> Las propiedades se definen de la misma forma que las variables (aplican las mismas reglas de estilo).

Métodos

Los métodos son "funciones" (como las que vimos en el capítulo anterior), solo que

técnicamente se denominan métodos, y representan acciones propias que puede realizar el objeto (y no otro):

```
class Objeto():
    color = "verde"
    tamanio = "grande"
    aspecto = "feo"
    antenas = Antena()
    ojos = Ojo()
    pelos = Pelo()


    def flotar(self):
        pass
```

> *Notar que el primer parámetro de un método, siempre debe ser* `self`.

Objeto

Las clases por sí mismas, no son más que modelos que nos servirán para crear objetos en concreto. Podemos decir que una clase, es el razonamiento abstracto de un objeto, mientras que el objeto, es su materialización. A la acción de crear objetos, se la denomina "instanciar una clase" y dicha instancia, consiste en asignar la clase, como valor a una variable:

```
class Objeto():
    color = "verde"
    tamanio = "grande"
    aspecto = "feo"
    antenas = Antena()
    ojos = Ojo()
    pelos = Pelo()

    def flotar(self):
        print 12


et = Objeto()
print et.color
print et.tamanio
print et.aspecto
et.color = "rosa"
print et.color
```

Herencia: característica principal de la POO

Como comentamos en el título anterior, algunos objetos comparten las mismas

propiedades y métodos que otro objeto, y además agregan nuevas propiedades y métodos. A esto se lo denomina herencia: una clase que hereda de otra. Vale aclarar, que en Python, **cuando una clase no hereda de ninguna otra, debe hacerse heredar de `object`**, que es la clase principal de Python, que define un objeto.

```
class Antena(object):
    color = ""
    longitud = ""


class Pelo(object):
    color = ""
    textura = ""


class Ojo(object):
    forma = ""
    color = ""
    tamanio = ""


class Objeto(object):
    color = ""
    tamanio = ""
    aspecto = ""
    antenas = Antena()
    ojos = Ojo()
    pelos = Pelo()

    def flotar(self):
        pass


class Dedo(object):
    longitud = ""
    forma = ""
    color = ""


class Pie(object):
    forma = ""
    color = ""
    dedos = Dedo()


# NuevoObjeto sí hereda de otra clase: Objeto
class NuevoObjeto(Objeto):
    pie = Pie()

    def saltar(self):
        pass
```

Accediendo a los métodos y propiedades de un objeto

Una vez creado un objeto, es decir, una vez hecha la instancia de clase, es posible

acceder a su métodos y propiedades. Para ello, Python utiliza una sintaxis muy simple: el nombre del objeto, seguido de punto y la propiedad o método al cuál se desea acceder:

```
objeto = MiClase()
print objeto.propiedad
objeto.otra_propiedad = "Nuevo valor"
variable = objeto.metodo()
print variable
print objeto.otro_metodo()
```