Primeros pasos

Recordando Python:

Tipos de datos en Python

- Enteros (int): todos los números, flotantes enteros y *long*.
- Booleanos (bool): falso o verdadero.
- Cadenas (str): cadenas de palabras.
- Listas: grupo de datos [1,2,3,"hola",[1,2,3]].
- Diccionarios: un grupo de datos que se acceden a partir de una clave {"clave":"dato"}.
- Tuplas: es un grupo de datos que se diferencia de la lista en que esta después de creada no se puede modificar.

Funciones

Las funciones las defines con **def** junto a un nombre y unos paréntesis que reciben los parámetros a usar. Terminas con dos puntos.

Después por indentación colocas los datos que se ejecutarán desde la función:

```
>>> def my_first_function():
...     return "Hello World!"
>>> my_first_function()
```

Hello World!

Variables

Las variables, a diferencia de los demás lenguajes de programación, no debes definirlas, ni tampoco su tipo de dato, ya que al momento de iterarlas se identificará su tipo. **Recuerda que en Python todo es un objeto**.

```
A=3
B=A
```

Listas

Las listas las declaras con corchetes. Estas pueden tener una lista dentro o cualquier tipo de dato.

```
>>> L=[22,True,"una lista",[1,2]]
>>> L[0]
22
```

Tuplas

Las tuplas tienen el mismo formato de las listas, sin embargo no puedes editar los datos de una tupla después de haberla creado.

```
>>> T=(22,True,”una tupla”,(1,2))
>>> T[0]
22
```

Diccionarios

En los diccionarios tienes un grupo de datos con un formato: la primera cadena o número será la clave para acceder al segundo dato, el segundo dato será el dato al cual accederás con la llave.

```
>>> D={“Kill Bill”:”Tarantino”,”DEN”:”Roger Korman”}
>>> D[“Kill Bill”]
"Tarantino"
```

Conversiones

Para convertir diferentes tipos de datos:

De flotante a entero:
```
>>> int(4.3)
4
```

De entero a flotante:
```
>>> float(4)
4.0
```

De entero a string:
```
>>>str(4.3)
“4.3”
```

De tupla a lista:
```
>>> list((4,5,2))
[4,5,2]
```

Operadores Comunes

Longitud de una cadena, lista, tupla, etc.:
```
>>> len(“key”)
3
```

Tipo de dato:
```
>>> type(4)
< type int >
```

Aplicar una conversión a un conjunto como una lista:

```
>>> map(str,[1,2,3,4])
['1','2','3','4']
```

Redondear un flotante con x número de decimales:

```
>>> round(6.3243,1)
6.3
```

Generar un rango en una lista **(todos los números entre 0 y el número definido)**:

```
>>> range(5)
[0,1,2,3,4]
```

Sumar un conjunto:

```
>>>sum([1,2,4])
7
```

Organizar un conjunto:

```
>>> sorted([5,2,1])
[1,2,5]
```

Mostrar funciones objeto x tipo de datos:

```
>>> dir([5,2,1])
(Aparecerá todas las funciones que tiene una lista)
```

Información sobre una función o librería:

```
>>> help(sorted)
(Aparecerá la documentación de la función sorted)
```

Clases

Clases es uno de los conceptos con más definiciones en la programación, pero en resumen **sólo son la representación de un objeto**. Para definir la clase usas *class* y el nombre. En caso de tener parámetros los pones entre paréntesis.

Para crear un constructor haces una función dentro de la clase con el nombre **__int__** y de parámetros **self** (significa su clase misma), **nombre_r** y **edad_r**:

```
>>> class Estudiante(object):
...     def __init__(self,nombre_r,edad_r):
...         self.nombre = nombre_r
...         self.edad = edad_r
...
...     def hola(self):
...         return "Mi nombre es %s y tengo %i" % (self.nombre,
self.edad)
...
>>> e = Estudiante("Iñigo Montoya", 22)
>>> print e.hola()
Mi nombre es Iñigo Montoya y tengo 22
```

Lo que hicimos en las dos últimas líneas fue:

1. En la variable e llamamos la clase Estudiante y le pasamos la cadena "Iñigo Montoya" y el entero 33.
2. Imprimimos la función hola() dentro de la variable e (a la que anteriormente habíamos pasado la clase).

Y por eso se imprime la cadena "Mi nombre es Iñigo Montoya y tengo 22"

Métodos especiales

__cmp__(self,otro)
Método llamado cuando utilizas los operadores de comparación para comprobar si tu objeto es menor, mayor o igual al objeto pasado como parámetro.

__len__(self)
Método llamado para comprobar la longitud del objeto. Lo usas, por ejemplo, cuando llamas la función **len(obj)** sobre nuestro código. Como es de suponer el método te debe devolver la longitud del objeto.

__init__(self,otro)
Es un constructor de nuestra clase, es decir, es un "método especial" que es llamas automáticamente cuando creas un objeto.

Condicionales IF

Los condicionales tienen la siguiente estructura. Ten en cuenta que lo que contiene los paréntesis es la comparación que debe cumplir para que los elementos se cumplan.

```
if ( a > b):
            elementos
elif( a == b ):
            elementos
else:
            elementos
```

Bucle FOR

El bucle de **for** lo puedes usar de la siguiente forma: recorres una cadena o lista a la cual va a tomar el elemento en cuestión con la siguiente estructura:

```
for i in ____:
            elementos
```

Ejemplo:

```
for i in range(10):
            print i
```

En este caso recorrerá una lista de diez elementos, es decir el *print i* de ejecutar diez veces. Ahora i va a tomar cada valor de la lista, entonces este **for** imprimirá los números del 0 al 9 (recordar que en un *range* vas hasta el número puesto -1).

Bucle WHILE

En este caso **while** tiene una condición que determina hasta cuándo se ejecutará. O sea que dejará de ejecutarse en el momento en que la condición deje de ser cierta. **La estructura de un while es la siguiente**:

```
while (condición):
            elementos
```

Ejemplo:

```
>>> x = 0
>>> while x<10:
...       print x
...       x+=1
```

En este ejemplo preguntará si es menor que diez. Dado que es menor imprimirá x y luego sumará una unidad a x. Luego x es 1 y como sigue siendo menor a diez se seguirá ejecutando, y así sucesivamente hasta que x llegue a ser mayor o igual a 10.

Control de excepciones

En este caso usarás un **try** que intentará resolver el contenido que tienes, pero en caso de existir un error pasará a un **except** en el que ejecutará su contenido. Puedes especificar los errores para dar diferentes mensajes según el error.

Para consumir una página web usamos la librería **urllib2** que debes importar de la siguiente manera:

```
>>> import urllib2
>>> try:
...       f = urllib2.urlopen("http://google.com")
...       print f.read()
...       f.close()
... except:
...       print "Error"
```

Este código te dirá: primero prueba realizar el código donde en f guardará el html de la página de google y luego se imprimirá. En caso de que haya un error se imprimirá **error**.

Crear un entorno virtual

Para crear un entorno virtual basta con determinar la carpeta en la que se ubicará y luego ejecutar:

```
$ virtualenv "name"
```

Siendo **"name"** el nombre del entorno que deseas crear. Para usarlo deberás colocar:

```
$ cd name
$ source bin/activate
```

Listo.

* Por último, dentro del entorno virtual para instalar Django en cualquier sistema operativo basta con ejecutar:

```
$ pip install django -U
```

* "U" lo colocamos para obtener la última versión.

* Ahora todo se instalará y ejecutará dentro del entorno virtual. Si deseas colocar algo de manera global la instalación deberá ser fuera de este.