01_Pensando_como_pythonista_1

January 20, 2021

1 Python de cero a experto

Autor: Luis Miguel de la Cruz Salas

Objetivos. Revisar los conceptos de objetos, etiquetas (nombres, variables), identidad, tipado dinámico, funciones de la biblioteca estándar, estado interno y comportamiento.

Instrucciones.

1. Para ejecutar una celda teclear: [Shift + Enter]
2. Cuando se modifica el código de cualquier celda, se debe re-ejecutar la celda.
3. Las celdas que dependan de una celda modificada, se deben re-ejecutar.
4. Cuando tenemos el nombre de una función de biblioteca, si ubicamos el cursor sobre el nombre y tecleamos [Shift + Tab] obtendremos ayuda sobre dicha función.
5. Lo anterior aplica a funciones definidas por el usuario, siempre y cuando se haya documentado con *docstring*.
6. Cuando tecleamos el nombre de un objeto, previamente definido, y le agregamos un '.' podemos en ese momento teclear el [Tab] para obtener ayuda sobre su comportamiento.

1.1 Objetos, Etiquetas e Identidad.

En Python, todo lo que se crea es un **objeto**.

En la celda que sigue, teclear lo siguiente:

`a = 1`

[2]:
```
a = 1
```

Acabamos de crear uno objeto! - El objeto es `1` - La etiqueta con que identificamos ese objeto es la letra `a` (nombre del objeto)

Cuando se pone el nombre del objeto `a` en una celda y se ejecuta, generalmente se obtiene su contenido o su tipo, veamos:

[3]:
```
a
```

[3]:
```
1
```

Para saber de qué tipo es el objeto, usamos la función `type(a)`

[4]:
```
type(a)
```

[4]:
```
int
```

Observamos que el objeto cuyo nombre es `a` es de tipo `int` (entero)

La función `id()` devuelve la identidad del objeto `a` en la memoria de la computadora

[5]:
```
id(a)
```

[5]:
```
93943995190592
```

Creamos ahora el objeto `2` y lo etiquetamos con `a` y luego preguntamos el tipo y la identidad del objeto:

[6]:
```
a = 2
```

[7]:
```
type(a)
```

[7]:
```
int
```

[8]:
```
id(a)
```

[8]:
```
93943995190624
```

¿Qué pasó con el objeto `1`?

Podemos darle otro nombre al objeto etiquetado previamente con el nombre `a`

[9]:
```
b = a
```

[10]:
```
id(b)
```

[10]:
```
93943995190624
```

Obsérvese que `b` y `a` tienen el mismo identificador, lo que significa que el objeto al que "apuntan" es el mismo. ¿A qué objeto hacen referencia `a` y `b`? Para saberlo, usamos las funciones `type()` y `print()`:

[11]:
```
type(a)
```

[11]:
```
int
```

[12]:
```
type(b)
```

[12]:
```
int
```

[13]:
```
print(a,b)
```

```
2 2
```

Observamos que el objeto al que hacen referencia es el entero 2.

¿Quá pasa si hacemos `a = 5`?

[14]:
```
a = 5
```

[15]:
```
print(a,b)
```

```
5 2
```

[16]:
```
id(a)
```

[16]:
```
93943995190720
```

[17]:
```
id(b)
```

[17]:
```
93943995190624
```

Ahora hagamos `c=b` y consultemos el tipo e identificador de `a`, `b` y `c`:

[18]:
```
c = b
c
```

[18]:
```
2
```

[19]:
```
print(type(a), type(b), type(c))
```

```
<class 'int'> <class 'int'> <class 'int'>
```

[20]:
```
print(id(a), id(b), id(c))
```

```
93943995190720 93943995190624 93943995190624
```

Es posible eliminar etiquetas usando la función `del()`

[21]:
```
del(c)
```

[22]:
```
c
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-22-2b66fd261ee5> in <module>
----> 1 c

NameError: name 'c' is not defined
```

[23]:
```
print(a, b)
```

```
5 2
```

[24]:
```
print(type(a), type(b))
```

```
<class 'int'> <class 'int'>
```

[25]:
```
print(id(a), id(b))
```

```
93943995190720 93943995190624
```

1.2 Tipado dinámico

Podemos crear objetos de muchos tipos y el tipo del objeto se determina en el momento de su creación. Por ejemplo:

[26]:
```
o = 100
print(type(o), id(o), o)
```

```
<class 'int'> 93943995193760 100
```

[27]:
```
o = 3.141592
print(type(o), id(o), o)
```

```
<class 'float'> 140347993951856 3.141592
```

[28]:
```
o = 'Hola Mundo Pythonico!'
print(type(o), id(o), len(o), o)
```

```
<class 'str'> 140347993295328 21 Hola Mundo Pythonico!
```

En los tres ejemplos anteriores observamos que `o` hace referencia a objetos de tipos diferentes, y ese tipo se conoce cuando se crea el objeto. Entonces, el tipo de un objeto se determina en el momento de su creación. A esto se le conoce como *tipado dinámico*.

1.3 Funciones de la biblioteca estándar

Las funciones que hemos estado usando hasta ahora `type()`, `id()`, `print()`, `len()` son funciones de la biblioteca estándar (*Built-in-functions*) que se pueden usar directamente. Puede encontrar una lista de estas funciones en Built-in Functions (puede ver la diferencia entre versiones de Python).

Una función que proporciona ayuda es `help()`, por ejemplo:

[29]:
```
help(print)
```

```
Help on built-in function print in module builtins:

print(…)
    print(value, …, sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
```

```
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

También se puede obtener ayuda si ponemos el cursor en la función y tecleamos [Shift+Tab], por ejemplo:

[30]:
```
print
```

[30]:
```
<function print>
```

Las funciones de la biblioteca estándar reciben como parámetros objetos de distintos tipos. Algunas de ellas no funcionan con ciertos tipos de objetos, por ejemplo:

[31]:
```
print(o, len(o))
```

```
Hola Mundo Pythonico! 21
```

[32]:
```
p = 3.1416
```

[33]:
```
print(p, len(p))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-33-7853bb315784> in <module>
----> 1 print(p, len(p))

TypeError: object of type 'float' has no len()
```

[34]:
```
help(len)
```

```
Help on built-in function len in module builtins:

len(obj, /)
    Return the number of items in a container.
```

Cualquier función en Python, también es un objeto:

[35]:
```
print
```

[35]:
```
<function print>
```

[36]:
```
print, len, type, id
```

[36]:
```
(<function print>, <function len(obj, /)>, type, <function id(obj, /)>)
```

[37]:
```
print(print, type)
```

```
<built-in function print> <class 'type'>
```

1.4 Estado interno y comportamiento de los objetos

Los objetos tienen dos características importantes: 1. estado interno y 2. comportamiento.

[38]:
```
cadena = 'Hola mundo pythonico'
```

[39]:
```
print(type(cadena))
```

```
<class 'str'>
```

Si tecleamos `cadena.` e inmediatamente después [Tab] obtendremos una lista de funciones y atributos que podemos usar sobre los objetos de tipo `str`:

[40]:
```
cadena.split()
```

[40]:
```
['Hola', 'mundo', 'pythonico']
```

En el ejemplo anterior, la función `split()` es parte del comportamiento del objeto. Pruebe otras funciones.

[41]:
```
cadena.center(30,'*')
```

[41]:
```
'*****Hola mundo pythonico*****'
```

El estado interno de un objeto, es básicamente el valor que contienen sus atributos. En algunos casos podemos conocer ese estado interno usando la función `print`:

[42]:
```
print(cadena)
```

```
Hola mundo pythonico
```

[43]:
```
print(cadena[6])
```

```
u
```

[44]:
```
p = 5
```

[45]:
```
p.numerator
```

[45]:
```
5
```

[46]:
```
p.denominator
```

[46]:
```
1
```

[47]:
```
p = 3.1416
```

[48]:
```
p.real
```

[48]:
```
3.1416
```

[49]:
```
p.imag
```

[49]:
```
0.0
```

1.5 Reglas para los nombres de los objetos.

Las siguientes son algunas reglas para poner nombre a los objetos:

1. Los nombres no pueden iniciar con un número.
2. No puede haber espacios en los nombres; se recomienda usar guión bajo '_' para separar nombres:

```
fuerza_de_gravedad = 9.8
```

3. No se puede usar ninguno de los siguientes símbolos en los nombres de las etiquetas:

```
' " , < > / ? | \ ( ) ! @ # $ % ^ & * ~ - +
```

4. Se considera una buena práctica usar minúsculas en los nombres.

Para más información véase: PEP8

[ ]:
```

```