5. Usando python3 y python2

Sunday, August 30, 2020 6:43 PM

Diferencias entre versiones

Imprimir (sentencia print)

Esta sea quizás la diferencia más conocida de todas. En Python 3 la sentencia *print* es una función y por tanto hay que encerrar entre paréntesis lo que se quiere imprimir, mientras que con Python 2 los paréntesis no son necesarios.

```
# Python 2
>>> print 'Programa en Python'
Programa en Python
# Python 3
>>> print('Programa en Python')
```

División de números enteros

En Python 2 la división entre números enteros es otro número entero, y para obtener un resultado con decimales el numerador o el denominador tiene que tener también al menos un decimal.

```
# Python 2
>>> 1/2
0
>>> 1.0/2
0.5
# Python 3
>>> 1/2
0.5
>>> 1.0/2
0.5
```

El mismo comportamiento truncado puede obtenerse en Python 3 usando //.

```
# Python 3
>>> 1//2
0
```

Módulo __future__

Este módulo proporciona retrocompatibilidad de comandos no compatibles entre distintas versiones. Lo podemos usar para adaptar Python 2 a la sintaxis de Python 3. Por ejemplo, en relación con los puntos anteriores nos permite usar print como una función en Python 2.

```
# Python 2
>>> from __future__ import print_function
>>> print('Programa en Python')
```

Otra posibilidad es tener el comportamiento de la división de Python 3 en la versión 2.

```
# Python 2
>>> from __future__ import division
>>> 1/2
0.5
```

En este enlace se describen todos los comandos disponibles con este módulo.

Iterar un diccionario

Para iterar los elementos clave-valor de un diccionario podemos utilizar el método *iteritems()* o *items()* en Python 2.

```
# Python 2
d = {'animal': 'perro', 'vehiculo': 'coche'}
for k,v in d.iteritems():
    print k,':',v
vehiculo : coche
animal : perro
```

En Python 3 esta operación sólo puede realizarse con el método *items()*, ya que al usar *iteritems()* tenemos una excepción del tipo *AttributeError*.

```
#Uso correcto en Python 3
for k,v in d.items():
    print(k,':',v)
# Python 3
d = {'animal': 'perro', 'vehiculo': 'coche'}
for k,v in d.iteritems():
```

```
    print(k,':',v)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
AttributeError: 'dict' object has no attribute 'iteritems'
```
Del mismo modo, los métodos *iterkeys()* e *itervalues()* para iterar las claves y los valores de un diccionario respectivamente no existen en Python 3. En su lugar tenemos que usar los métodos *keys()* y *values()*.

Función next() y método .next()

Recuperar el siguiente elemento de un iterador puede hacerse tanto con la función *next()* como con el método *.next()* en Python 2, mientras que en Python 3 *next* sólo puede utilizarse como función.

```
# Python 2
>>> iterador = (letra for letra in 'python')
>>> next(iterador)
'p'
>>> iterador.next()
'y'
```
Utilizar *next* como método en Python 3 nos da una excepción del tipo *AttributeError*
```
# Python 3
>>> iterador = (letra for letra in 'python')
>>> next(iterador)
'p'
>>> iterador.next()
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
AttributeError: 'generator' object has no attribute 'next'
```

Comparación entre tipos

Python 2 permite la comparación entre objetos de tipo distinto, sin embargo Python 3 es restrictivo en este aspecto dándonos una excepción del tipo *TypeError*.

```
# Python 2
>>> 1 < '1'
True
# Python 3
>>> 1 < '1'
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: '<' not supported between instances of 'int' and 'str'
```

Función input

La función *input* sirve para capturar datos procedentes del teclado. En Python 2 esta función trata los datos tal cual sin realizar ninguna conversión. Por ejemplo, si entramos un número entero, la entrada será de tipo *int*. Si queremos que la entrada se trate como una cadena, entonces tenemos que usar la función *raw_input* ya que ésta convierte los datos a tipo *str*.

```
# Python 2
>>> entrada = input('Introduce un número: ')
Introduce un número: 1
>>> type(entrada)
<class 'int'>
>>> entrada = raw_input('Introduce un número: ')
Introduce un número: 1
>>> type(entrada)
<class 'str'>
```
En Python 3 se ha suprimido la función *raw_input*, y su comportamiento lo toma la función *input*. Esto facilita el tratamiento de los datos entrados por teclado ya que sabemos de antemano que siempre serán *strings*.
```
# Python 3
>>> entrada = input('Introduce un número: ')
Introduce un número: 1
>>> type(entrada)
<class 'str'>
```

Cadenas de texto Unicode

En Python 2 las cadenas de texto (tipo *str*) están codificadas en formato ASCII con lo que cada carácter sólo requiere 7 bits de información. Por otro lado, las cadenas de texto codificadas en bytes requieren 8 bits de información. Por lo tanto, sólo es posible combinar ambos tipos cuando la representación en bytes sólo contiene caracteres ASCII.

En Python 3 todas las cadenas de texto son Unicode (8 bits). Éstas pueden almacenarse como texto (tipo *str*) o como datos binarios (tipo *bytes*), siendo imposible combinar ambos tipos ya que nos da una excepción *TypeError*.

```
# Python 3
>>> a = 'Hola'
>>> type(a)
<class 'str'>
>>> b = b'mundo'
>>> type(b)
<class 'bytes'>
>>> a + b
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "bytes") to str
```

Módulo six

No hay compatibilidad directa entre Python2 y Python3, pero la estructura básica del lenguaje no ha cambiado entre versiones, por lo que es posible tener un "puente" entre ambas versiones.
Esto nos permitirá contar con código intermedio entre ambas versiones

```
import six
mydict = {'clave1': 1, 'clave2': 2, 'clave3': 3}
for k, v in six.iteritems(mydict):
    print(k,v)
```

Además, six tiene el atributo PY3 para indicarnos si estamos o no usando Python3.

Referencias:

- From <https://www.programaenpython.com/miscelanea/diferencias-entre-python-2-y-3/>