Después de mucho tiempo sin escribir, dejo un post en donde comparo algunos de los cambios introducidos en Python 3.x con respecto a Python 2.x con sus respectivos ejemplos para explicarlos.

En primer lugar, al inicio de todos los ejemplos vamos a poner lo siguiente al inicio de cada ejemplo para que al ejecutarlos, nos muestre la versión del intérprete de python que estamos usando:

from platform import python\_version  
print ("Python " + python\_version())

# Print es una función en python 3

En **python 2**, para imprimir/mostrar un mensaje bastaba con un print “texto”, por ejemplo:

from platform import python\_version  
print ("Python " + python\_version())  
print "hola mundo"

Da como resultado:

Python 2.7.11  
hola mundo  
En cambio si ejecutamos lo mismo en **python 3**, obtenemos un error:

File "test.py", line 4  
 print "hola mundo"  
 ^  
SyntaxError: Missing parentheses in call to 'print'

Esto se debe a que en python 3 print es una función y por lo tanto debe llevar los paréntesis:

from platform import python\_version  
print ("Python " + python\_version())  
print ("hola mundo")

Ahora funciona y da como resultado:

Python 3.5.1  
hola mundo  
Que print sea una función tiene algunas ventajas, como que por ejemplo usando parámetros opcionales, uno puede controlar que no se produzca el salto de línea (\n) reemplazandolo por otra cosa.

from platform import python\_version  
print ("Python " + python\_version())  
print("Hola mundo,", end="")

print(" agregamos otro texto")

Da como resultado:

Python 3.5.1  
Hola mundo, agregamos otro texto

# División de números enteros

En **python 2** al dividir enteros, siempre el resultado era un entero (aproximando dependiendo del resultado).

La forma de que esto no ocurriera era escribiendo uno de los números como flotante (ejemplo 2.0) o señalando directamente (con un float(2) por ejemplo). Así tenemos:

from platform import python\_version  
print ("Python " + python\_version())  
print (3 / 2)  
print (3 // 2)  
print (3 / 2.0)  
print (3 // 2.0)

En python 2 da como resultado:

Python 2.7.11  
1  
1  
1.5  
1.0  
En cambio en **python 3**, el primer resultado sí da como decimal:

Python 3.5.1  
1.5  
1  
1.5  
1.0

Este es uno de los cambios más peligrosos a la hora de portar código de una versión a otra.

*Nota: la // es una división entera, o sea el resultado siempre va a ser un numero entero tanto en python 2.X como en python 3.X (por ejemplo 12.5//2 es 6.0)*

# Las “cadenas” (“strings”) son Unicode de forma predeterminada en python 3

En **python 2** la codificación por defecto de las cadenas de texto es en formato ASCII (7 bits), lo cual no te da problemas si tu idioma es el inglés, pero si vas a escribir palabras con acentos o letras que no están en ASCII como la ñ, tenías que especificar un codificación que soportara caracteres [Unicode](https://es.wikipedia.org/wiki/Unicode) en tu script, como por ejemplo UTF-8 (y guardar el archivo con esa codificación), de caso contrario se producían errores al ejecutarlo.

Así por ejemplo al ejecutar esto en **Python 2:**

from platform import python\_version  
print ("Python " + python\_version())  
print("más ñandú")

Obteníamos como resultado un error (en Python 2):

File "test.py", line 4  
SyntaxError: Non-ASCII character '\xc3' in file test.py on line 4, but no encoding declared; see http://python.org/dev/peps/pep-0263/ for details

Así para escribir una frase como "más ñandú" y que funcionaran había que hacer algo como esto en python 2.X (declarar una codificación, en este caso utf-8, y comprobar además que el editor de texto en donde estabas escribiendo el script guardara el archivo en esa codificación utf-8):

# -\*- coding: utf-8 -\*-  
from platform import python\_version  
print ("Python " + python\_version())  
print("más ñandú")  
para obtener:

Python 2.7.11  
más ñandú

Ahora en **python 3** las cadenas por defecto son Unicode (utf-8), así por ejemplo esto funciona inmediatamente:

from platform import python\_version  
print ("Python " + python\_version())  
print("más ñandú")  
Obtenemos:

Python 3.5.1  
más ñandú

# Input es una cadena de texto en python 3

En **python 2** habían dos funciones para ingresar datos por un teclado raw\_input()en que lo ingresado se trataba como una cadena de texto e input() en lo que se ingresaba se evaluaba y se trataba por su tipo (por ejemplo 123 se considera un entero). Así en un intérprete interactivo de python 2 tenemos:

Python 2.7.11 (v2.7.11:6d1b6a68f775, Dec 5 2015, 20:40:30) [MSC v.1500 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license" for more information.  
  
>>> mi\_input = input("ingresar un numero: ")  
ingresar un numero: 123  
>>> type(mi\_input)  
<type 'int'>  
>>> mi\_input = raw\_input("ingresar un numero: ")  
ingresar un numero: 123  
>>>type(mi\_input)  
<type 'str'>

Ojo que esto podía ser peligroso, ya que si no se tenía cuidado se podía ingresar código peligroso por el input() generando una brecha se seguridad en el sistema.

En **python 3**, se eliminó el input() de python 2 quedando el raw\_input() como el nuevo input(). O sea el input() de python 3 siempre devuelve una cadena de texto.

Python 3.5.1 (v3.5.1:37a07cee5969, Dec 6 2015, 01:54:25) [MSC v.1900 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license" for more information.  
>>> mi\_input = input("ingresar un numero: ")  
ingresar un numero: 123  
>>> type(mi\_input)  
<class 'str'>

# La función next() y el método .next()

En **python 2** se puede usar next tanto como una función next(algo) o como un método algo.next(), sin embargo en **Python 3** solo se puede usar como función:

Asi para **python 2** es valido:

Python 2.7.11 (v2.7.11:6d1b6a68f775, Dec 5 2015, 20:40:30) [MSC v.1500 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license" for more information.  
>>> mi\_generator = (letra for letra in "abcdef")  
>>> next(mi\_generator)  
'a'  
>>> mi\_generator.next()  
'b'  
>>>  
En cambio en **python 3**, el metodo lanza un error:

Python 3.5.1 (v3.5.1:37a07cee5969, Dec 6 2015, 01:54:25) [MSC v.1900 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license" for more information.  
  
>>> mi\_generator = (letra for letra in "abcdef")  
>>> next(mi\_generator)  
'a'  
>>> mi\_generator.next()  
Traceback (most recent call last):  
 File "<stdin>", line 1, in <module>  
AttributeError: 'generator' object has no attribute 'next'  
>>>  
Comparando tipos

Otra diferencia es que en Python 3 nos indica un error cuando intentamos comparar tipos de datos diferentes, ejemplo:

from platform import python\_version  
print ("Python " + python\_version())  
print("Comparando [1, 2] > 'foo' = "), [1, 2] > 'foo'  
print("Comparando (1, 2) > 'foo' = "), (1, 2) > 'foo'  
print("Comparando [1, 2] > (1, 2) = "), [1, 2] > (1, 2)

En **python 2**:

Python 2.7.11  
Comparando [1, 2] > 'foo' = False  
Comparando (1, 2) > 'foo' = True  
Comparando [1, 2] > (1, 2) = False

En **python 3** nos arroja un error porque son tipos de datos diferentes (lista vs cadena de texto):

Python 3.5.1  
Comparando [1, 2] > 'foo' =  
Traceback (most recent call last):  
Traceback (most recent call last):  
 File "D:\test.py", line 4, in <module>  
 print("Comparando [1, 2] > 'foo' = "), [1, 2] > 'foo'  
TypeError: unorderable types: list() > str()

# Manejo de Exepciones

En **python 2** se acepta las dos maneras de escribir una excepción (sin paréntesis o con paréntesis como si fuera una función), en cambio en python 3 solo se acepta de la segunda forma.

from platform import python\_version  
print ("Python " + python\_version())  
raise IOError, "error de archivo"

Resultado en python 2:

Python 2.7.11  
Traceback (most recent call last):  
 File "test.py", line 4, in <module>  
 raise IOError, "error de archivo"  
IOError: error de archivo

En **python 3** se produce un error por la sintaxis no valida:

File "test.py", line 4  
 raise IOError, "error de archivo"  
 ^  
SyntaxError: invalid syntax

**En cambio de esta forma es válida tanto en python 2 como en 3**:

from platform import python\_version  
print ("Python " + python\_version())  
raise IOError("error de archivo")

dando como resultados:

Python 2.7.11  
Traceback (most recent call last):  
 File "test.py", line 4, in <module>  
 raise IOError("error de archivo")  
IOError: error de archivo

Python 3.5.1  
Traceback (most recent call last):  
 File "test.py", line 4, in <module>  
 raise IOError("error de archivo")  
OSError: error de archivo