# Python 2 vs Python 3

¿Qué ha cambiado?

**Profesor: Jérémy Barbay**
Auxiliares: Felipe Lizama, F. Giovanni Sanguineti

Para el curso de *Introducción a la Programación*, trabajamos utilizando Python 2. Esta versión del lenguaje de programación, que lleva sin actualizarse desde 2018, dejará de tener soporte a contar del 1 de enero de 2020. La nueva versión de Python, la 3, introdujo una serie de cambios y mejoras que han hecho cambiar ligeramente la forma en la que escribimos código. En este documento señalaremos las más relevantes, para tenerlas en consideración en nuestro trabajo durante el semestre. Si bien hay varios cambios más de los que se mencionan en este documento, el resto exceden la utilidad de este curso, por lo que no los abordaremos.

## Cambios en la división de enteros

En Python 2, cuando dividíamos dos números enteros entre sí, el resultado siempre era un entero. En el siguiente ejemplo vemos qué pasa si dividimos 10 entre 3:

```
>>> 10 / 3
3
```

Si queríamos obtener el resultado como un decimal, teníamos que agregar un .0 al número o *castearlo* a float.

```
>>> a = 3
>>> float(a) / 2
1.5
>>> 10.0 / 3
3.33333333
```

En Python 3 se ha superado esto: ahora la división entre enteros siempre retornará el resultado correcto como un decimal.

```
>>> 3 / 2
1.5
>>> 10 / 3
3.33333333
```

Recuerda que existe la *división entera*, que nos garantiza obtener siempre un entero truncado de la división realizada.

```
>>> 6 // 7
```

```
0
```

Esta diferencia en la división entera es una de las que produce mayores *incompatibilidades* al portar aplicaciones escritas en Python 2 a Python 3.

# Unicode por defecto

¿Recuerdas que en Python 2 no podías escribir tildes, la letra 'ñ' y otros caracteres especiales porque el código no compilaba? Esto se debía a que el tipo de codificación que utilizaba esta versión de Python, ASCII, no admitía estos caracteres especiales. Ahora, la versión 3 de Python utiliza la codificación Unicode, que nos permite escribir todas las tildes, eñes, números romanos y *emojis* que queramos. Si te interesa revisar la infinidad de caracteres que posee la codificación Unicode, puedes revisar este enlace.

# La función print()

En Python 2, para imprimir, lo hacíamos utilizando **print** como una *declaración* o *statement*:

```
>>> print "La respuesta es", 2*2
La respuesta es 4
```

En Python 3, esto ha cambiado: ahora print es una función:

```
>>> print("La respuesta es", 2*2)
La respuesta es 4
```

Puede parecer sencillo, pero ojo, en algunos casos puede ser un poco más complejo. Veamos el siguiente caso en Python 2:

```
>>> x = 2
>>> y = 3
>>> print(x,y)
(2,3)
>>> print x,y
2 3
```

En cambio, en Python 3 tenemos lo siguiente:

```
>>> print(x,y)
2 3
>>> print((x,y))
(2,3)
```

Finalmente, podemos introducir separadores de la siguiente forma:

```
>>> print("somos", 55, "personas", sep=" ")
somos 55 personas
>>> print("somos", 55, "personas", sep="_ ")
somos_55_personas
```

# input()

¿Recuerdas cómo recibir una entrada desde el usuario de tu programa? Utilizábamos la función `input()`, que recibía de forma literal la entrada y la evaluaba en función de su tipo. Esto hacía que si un usuario tenía que escribir un texto, necesariamente tenía que hacerlo utilizando comillas, o se producía un error.

```
>>> a = input()
hola
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<string>", line 1, in <module>
NameError: name 'hola' is not defined
>>> a = input()
"hola"
>>> print a
hola
```

Para evitar estos problemas y otros, utilizábamos otra función: `raw_input()`, la que trabajaba toda entrada como texto.

```
>>> a = raw_input()
hola
>>> print a
hola
```

Por suerte para nosotros, en Python 3 se ha eliminado `raw_input()`, pero se le ha dado su funcionalidad al `input()` normal: ahora `input()` recibe toda entrada como texto. Si queremos extraer un número, debemos *castearlo*.

```
>>> a = input()
hola
>>> print(a)
hola
>>> b = input()
1234
>>> type(b)
<class 'str'>
>>> b + 5
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    b + 5
TypeError: can only concatenate str (not "int") to str
>>> b = int(b) + 5
>>> print(b)
1239
```