2. Variables y operadores

Comenzamos abriendo el REPL "Read-Eval-Print-Loop“, más conocido como la consola del lenguaje:

```
$ python3
Python 3.4.3 (default, Nov 28 2017,
16:41:13)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or
"license" for more information.
>>>
>>>exit()
>>>quit()
$
```

Primer contacto con el REPL

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

```
>>> Venimos en son de paz, por favor
llevadnos ante vuestro lider
  File "<stdin>", line 1
    Venimos en son de paz, por favor llevadnos ante vuestro
lider
             ^
SyntaxError: invalid syntax
>>>
```

Variables y operadores

Vamos a realizar el acostumbrado saludo “Hola mundo”

Para ello vamos a usar el idioma que python entiende

```
>>> print ('Hola mundo')
Hola mundo
>>>
```

'Hola mundo' Es una cadena de caracteres por lo que va encerrada entre ‘’

Print(…, …, …) Es una función por lo que lleva () y dentro de los paréntesis uno o más argumentos, separados por comas

```
>>> print ('Me alegro mucho de conocerte')
Me alegro mucho de conocerte
>>> print (3)
3
```

Variables y operadores

Vamos a utilizar la consola de python como una calculadora:

```
>>> 3
3
>>> 3+2
5
>>> 7 % 2
1
>>> (1 + 3) * 4 +2
18
>>> 1000 ** 2
1000000
>>> 2 ** 64
18446744073709551616
>>>
```

+ - Operadores de suma o resta

% Operador Módulo

*** Operador Producto**

**** Operador Exponenciación**

Variables y operadores

Una mejora más intuitiva de python 3 es con respecto al operador de división:

```
>>> 4/3
1.3333333333333333
>>> 4//3
1
>>> 4.68 / 2
2.34
```

// es la división entera

- Reglas de precedencia de operadores:

1. **Paréntesis ()**
2. **Exponenciación ****
3. **Multiplicación * y División /, con igual precedencia**
4. **Sumas + y restas - con igual nivel de precedencia**
5. **A igual precedencia se evalúa de izquierda a derecha**

Almacenemos algunos cálculos : Variables.

```
>>> (1 + 3) * 4 +2
18
>>> _ + 1
19
>>> X = 5
>>> X
5
>>> Y = 1
>>> X + Y
6
```

_ memoriza lo último devuelto por el REPL

X e Y son unas etiqueta con la que nos referimos a un valor

Estas etiquetas se conocen habitualmente como variables.

Reglas para crear etiquetas o variables:

- Los nombres de variables pueden ser arbitrariamente largos.
- Pueden contener letras y números.
- No puede comenzar con un número.
- Puedes usar mayúsculas pero se recomienda empezar con una letra minúscula.
- El carácter _ (guión bajo), puede utilizarse y es común en nombres múltiples como:

mi_nombre = 'Patricio'
distancia_en_km = 56

Asignación múltiple

```
>>> x,y = 2,7
>>> x + y
9
>>> x = y = z = 1
>>> x + y + z
3
```

Impresión múltiple

```
>>> print ( x , y , z)
1 1 1
>>> print ('x es ', x ,' y z es ', z)
x es  1  y z es  1
```

Operadores de comparación

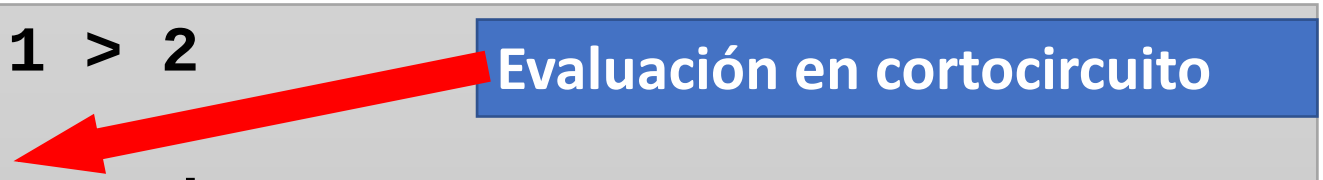
```
>>> 1 == 2
False
>>> 2 != 3
True
>>> 7 < 1
False
>>> 2 > 2
False
>>> 2 <= 3
True
>>> 2 >= 3
False
>>> 0<=2<=1
False
```

== comparador igual
!= comparador distinto
< comparador menor que
> Comparador mayor que
<= Comparador menor o igual
>= Comparador mayor o igual

True y False son un tipo de dato de la clase **bool**

Operadores lógicos: ***not, and, or***

```
>>> not 1 > 2
True
>>> 2>= 0 and 2<=1
False
>>> 2>= 0 or 2<=1
True
>>> 2<7 and 7<6 or 4>8
False
>>> 2<7 and 1<6 or 9>8 and 7<4
True
>>> 2<7 and (1<6 or 9>8) and 7<4
False
```



Operadores a nivel de bit:

```
>>>4 & 1
0
>>> 4 | 1
5
>>> ~4
-5
>>> 4 ^ 2
6
>>>4 >> 1
2
>>>4 << 1
8
```

&	and binario bit a bit
|	or binario bit a bit
~	not binario bit a bit
>>	Desplazar un bit a la derecha
<<	Desplazar un bit a la izquierda

Nota: Representación binaria ->

```
>>> bin(4)
'0b100'
```

```
>>> 0b100
4
```

Operadores de asignación:

= += -= *= /= %= **= //=

c = a + b	se asigna el valor de a + b en c
c += a	es lo mismo que c = c + a
c -= a	es lo mismo que c = c - a
c *= a	es lo mismo que c = c * a
c /= a	es lo mismo que c = c / a
c %= a	es lo mismo que c = c % a
c **= a	es lo mismo que c = c ** a
c //= a	es lo mismo que c = c // a

Operadores especiales:

is - Es True si los operadores son idénticos
is not - Es True si los operadores no son idénticos
in - Es True si el valor o variable se encuentra en la secuencia
not in - Es True si el valor o variable no se encuentra en la secuencia

Palabras reservadas:

- Python reserva una lista de palabras claves que no pueden usarse como variables:

and, as, assert, break, class, continue, def, del, elif, else, except, finally, for, from, global, if, import, in, is, lambda, nonlocal, not, or, pass, raise, return, try, while, with, yield

```
>>> while = 5
  File "<stdin>", line 1
    while=5
         ^
SyntaxError: invalid syntax
>>>
```

Tipos básicos:

- `Clase int` → Enteros
- `Clase float` → Reales
- `Clase bool` → Lógicos
- `Clase str` → Cadenas de texto
- `Clase complex` → Números complejos

Función **type**: Para saber el tipo que estamos manejando:

```
>>> type('Hola mundo')
<type 'str'>

>>> 4==5
False
>>> type(_)
<type 'bool'>

>>> type(17)
<type 'int'>

>>> type( 3+4.14j )
<class 'complex'>
```

- El tipo de una variable es el tipo del valor al que se refiere

```
>>> frase = 'Hola mundo'
>>> type(frase)
<type 'str'>
```

- Si una secuencia de caracteres va entre comillas simples o dobles, es una cadena:

```
>>> type('17')
<type 'str'>

>>> type("17 34 56")
<type 'str'>
```

- Uso de la comilla simple o doble:

```
>>> frase ='Javi dijo "OK"'
>>> print(frase)
Javi dijo "OK”


>>> print('Hola "mundo"',“estoy muy 'feliz'")
Hola "mundo" estoy muy 'feliz'
>>>
```

Ejemplo de sobrecarga de operadores:

- Operador de concatenación de cadenas **+**

```
>>> 'esto une' + ' las cadenas'
'esto une las cadenas'
>>> primero =‘100’
>>> segundo =‘150’
>>> print (primero + segundo)
100150
```

Pruebe a hacer un type(_) del último resultado, para ver el tipo de dato

```
>>> saludo='Hola "mundo" ' + “estoy muy 'feliz'"
>>> saludo
'Hola "mundo" estoy muy \'feliz\''
>>>
```

\‘ es el carácter de escape de la comilla

- Operador producto en cadenas *****

```
>>> valor='3'
>>> valor * 5
'33333'
```

```
>>> 'feliz'*5
'felizfelizfelizfelizfeliz'
```

- Algunos argumentos de la función **print()**

print(value1, ..., sep=' ', end='\n' …..)

```
>>> print(‘a’,’b’)
a b
>>> print(‘a’,’b’,sep=‘’)
ab
>>> print(‘a’,’b’,sep=‘:-)’)
a:-)b

>>> print('Esta línea no salta', end='')
Esta línea no salta>>>
```

Ejecución de archivos con código python (script)

Siempre que queramos ejecutar nuestros programas, no vamos a reescribirlo todo en la consola de python. Lo que hacemos es:

- Crear un fichero de texto con extensión .py , que contenga nuestro código en python. Ej: holamundo.py
- Lo abrimos con el editor que deseemos y escribimos en el:

```
print('Hola mundo')
```

- Grabamos nuestro archivo de script y lo ejecutamos con el comando:

```
$ python3 holamundo.py
```

Petición de información

Python proporciona una función que recibe entradas desde teclado.

Input()

```
>>> entrada = input()
Cualquier cosa
>>> print(entrada)
Cualquier cosa
>>>
```

TODO lo que se recibe por **input**
son cadenas de caracteres (str)

Petición de información

Es bueno dar un mensaje solicitando la entrada:

input (argumento), el argumento es una cadena de caracteres con un mensaje o prompt de entrada

```
>>> entrada = input('Dame tu nombre:\n')
Dame tu nombre:
Juanjo
>>> print(entrada)
Juanjo
>>>
```

El carácter de escape \n representa un **newline,** que es un carácter especial que provoca un salto de línea.

Petición de información

Si se espera un valor entero como entrada, se puede usar la función ***int()*** para convertirlo. Igualmente con ***float()*** si se espera un número real, o **complex()** en caso de complejos:

```
>>> entrada = input('Altura media:\t')
Altura media:	65
>>> int(entrada)+ 1
66
```

El carácter de escape \t representa un tabulador

OJO ! Si se intenta convertir a int o float, una cadena de texto, que no lo es, dará un error. Para eso veremos el tema de las excepciones en python 3

```
>>> int('45')
45
>>>type(_)
<class 'int'>
```

```
>>> int('101010', base=2)
42
```

Comentarios

- Para documentar el código podemos usar el carácter **#**:

```
V = 5   # velocidad en metros/segundo
```

- O bien para párrafos largos de comentario, sin repetir **#** en cada línea, podemos usar tripe ***comilla doble***, al principio y al final de los comentarios:

```
“ “ “
Todo este texto
es un comentario
“ “ “
```

Forma de introducir un texto largo con saltos de línea en una variable:

```
frase = """Como podemos ver en el gráfico,
en 2020 la línea que marca las ventas de coches eléctricos
empieza a inclinarse hacia arriba.
Pero es a partir de 2025 cuándo coge mayor velocidad llegando al 25%
de las ventas.
Una cifra que no dejará de crecer hasta llegar a cerca del 60% para
2040
"""
```

Ejercicios:

Ej.2.1- ¿Qué está mal en el código siguiente?

```
>>> primt 'Hola munddo'
```

Ej.2.2- Escribe un programa que use ***input*** para pedirle al usuario su nombre y luego darle la bienvenida.

Ej.2.3- Escribe un programa para pedirle al usuario el número de horas y la tarifa por hora para calcular el salario bruto.

Introduzca Horas: 35
Introduzca Tarifa: 2.75
Salario : 96.25