Table of Contents

Tenga en cuenta que estamos usando Python 3.7 (**NO** Python 2! Python 2 está muerto)

In [3]:

```
# Let's make sure we are using Python 3
import sys
print(sys.version)
```

```
3.7.6 (default, Jan  8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)]
```

¡Empecemos!

In [12]:

```
print('Hello AiSaturdays!')
```

```
Hello AiSaturdays!
```

`print` es una función incorporada. Lo usaremos varias veces.

Tipos de datos básicos: números, booleanos y cadenas

Los tipos de datos básicos en python3 son: https://realpython.com/python-data-types/ (https://realpython.com/python-data-types/)

- Numérico: enteros, flotantes y números complejos
- Texto: cadenas
- Lógica: booleana

Numérico

Nota:

- Use la función `type()` para obtener el tipo de una variable
- Los números pueden ser enteros ('int'), como 3, 5 y 3049
- Los números decimales son flotantes ('float'), como 2.5, 3.1 y 2.34938493
- Los números imaginarios se definen con la letra `j`

In [15]:

```
a_int = 5
print(type(a_int))
print(a_int)

a_float = 2.3
print(type(a_float))
print(a_float)

a_img = 2j
print(type(a_img))
print(a_img)
```

```
<class 'int'>
5
<class 'float'>
2.3
<class 'complex'>
2j
```

Operadores matemáticos: +, -, \ , /, *Los operadores matemáticos le permiten realizar operaciones matemáticas.

Nota: es el operador de exponenciación

In [7]:

```
a = 2
b = a + 1
print(b)
c = a - 1
print(c)
d = a * 2
print(d)
e = a / 2
print(e)

f = a ** 2
print(f)
```

```
3
1
4
1.0
4
```

Operadores matemáticos abreviados

`a + = 1` es la abreviatura de `a = a + 1`

In [18]:

```
a = 5
a += 1
print(a)
a *= 2
print(a)
a /= 4
print(a)
a -= 2
print(a)
```

```
6
12
3.0
1.0
```

> **Warning :** `+=` it's not the same as `=+`.

In [25]:

```
print(f'{a}')
```

```
1.0
```

In [29]:

```
a = 1
a += 4
print(a)
```

```
5
```

In [30]:

```
a = 1
a =+ 4
print(a)
```

```
4
```

Booleanos y operadores lógicos

In [31]:

```
im_true = True
im_false = False

print(type(im_true))
print(im_true)
```

```
<class 'bool'>
True
```

Operadores logicos

Los operadores lógicos (`==` y `!=`) Le permiten comparar los valores de las variables en el lado izquierdo y derecho.

In [32]:

```
print(im_true == im_false)
print(im_true != im_false)
```

```
False
True
```

The `and` operator will return `True` only if both elements are `True` .

In [33]:

```
print(im_true and im_false)
```

```
False
```

The `or` operator will return `True` if any of the variables are `True` .

In [34]:

```
print(im_true or im_false)
```

```
True
```

In [37]:

```
im_true + im_true
```

Out[37]:

```
2
```

> **Warning :** Lógico Verdadero a menudo se trata como entero 1 y Falso como 0. Uso de operaciones matemáticas, por ejemplo, `+`: Verdadero + Verdadero será 2.

In [43]:

```
True + True
```

Out[43]:

```
2
```

In [44]:

```
True and True
```

Out[44]:

```
True
```

In [46]:

```
True + False
```

Out[46]:

```
1
```

In [47]:

```
True and False
```

Out[47]:

```
False
```

1.3 Cuerdas

Puede usar comillas simples o dobles para cadenas.

my_string = 'Ai' my_other_string = "Saturdays" print(my_string, my_other_string)

Métodos de cadena

Cuerdas de concatenación:

In [52]:

```
another_string = 'Hello' + my_string + my_other_string
print(another_string)
```

```
HelloAiSaturdays
```

> **Cuidado con los espacios.**

In [51]:

```
another_string = 'Hello, ' + my_string + " " + my_other_string
print(another_string)
```

```
hello, Ai Saturdays
```

Obtenga la longitud de la cadena:

In [11]:

```
print(len(another_string))
```

```
22
```

`len` es otra función incorporada

Las cadenas se pueden multiplicar.

In [ ]:

```

```

Tipos de datos de contenedor

Algunos contenedores incorporados son

- Lista y tuplas
- Establecer
- Dict

Se pueden dividir en mutables o inmutables:

- Inmutables: Tuplas
- Mutable: lista, dict y conjuntos.

Listas

Una `lista` de Python almacena múltiples elementos, cuyos tipos pueden ser diferentes.

In [56]:

```
my_list = ['a', 'b', 'c', 3485]
print(my_list)
```

```
['a', 'b', 'c', 3485]
```

Puede acceder a un elemento en una lista con la siguiente sintaxis:

> **Note :**, el primer elemento de una lista tiene un índice de cero.

In [57]:

```
print(my_list[2])
print(my_list[0])
```

```
c
a
```

Reasignación de elementos en una lista:

In [58]:

```
my_list[0] = 'Ai'
print(my_list)
```

```
['Ai', 'b', 'c', 3485]
```

Agregar / eliminar elementos de una lista:

In [59]:

```
my_list.append('Saturdays')
print(my_list)

my_list.pop()
print(my_list)
```

```
['Ai', 'b', 'c', 3485, 'Saturdays']
['Ai', 'b', 'c', 3485]
```

Acceso a múltiples elementos en una lista:

In [16]:

```
print(my_list[0:2]) # Access elements in index 0, 1 and 2
print(my_list[2:]) # Access elements from index 2 to the end
print(my_list[:2]) # Access elements from the beginning to index 2
```

```
['delta', 'b']
['c', 3485]
['delta', 'b']
```

Tuplas

Una tupla es una secuencia de objetos Python inmutables. Las tuplas son secuencias, al igual que las listas. Las diferencias entre las tuplas y las listas son que las tuplas no se pueden cambiar a diferencia de las listas y las tuplas usan paréntesis, mientras que las listas usan corchetes.

In [60]:

```
a = (2, 'a')
print(a)

print(a[0])
print(type(a))
```

```
(2, 'a')
2
<class 'tuple'>
```

Para acceder a todas las propiedades y métodos de un objeto, use la función incorporada `dir` .

In [63]:

```
dir(a)
```

Out[63]:

```
['__add__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__getnewargs__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__mul__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__rmul__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'count',
 'index']
```

Diccionarios

Los diccionarios contienen pares clave / valor y son útiles para almacenar información.

In [65]:

```
my_dict = { 'key_one': 'Ai', 'key_two': 'Saturdays' }
my_dict
```

Out[65]:

```
{'key_one': 'Ai', 'key_two': 'Saturdays'}
```

Acceda a un valor de un diccionario mediante una clave:

In [66]:

```
print(my_dict['key_one'])
print(my_dict['key_two'])
```

```
Ai
Saturdays
```

Recorriendo los valores de un diccionario:

> **Note :** consulte las siguientes secciones a continuación para obtener información sobre los bucles.

In [69]:

```
for key in my_dict:
    print("The key is " + key)
```

```
The key is key_one
The key is key_two
```

In [70]:

```
for key, value in my_dict.items():
    print("The key is " + key + ", and the value is " + value)
```

```
The key is key_one, and the value is Ai
The key is key_two, and the value is Saturdays
```

Conjuntos

Los conjuntos son similares a las listas, pero solo pueden contener valores únicos.

In [72]:

```
my_set = {1, 2, 3, 3, 3, 3, 'hello'}
print(my_set)
```

```
{'hello', 1, 2, 3}
```

Al definir un conjunto con el mismo valor presente varias veces, solo se agregará un elemento al conjunto.

3. Funciones

Una función es un bloque de código reutilizable que realiza una determinada acción. Una vez que haya definido una función, ¡puede usarla en cualquier parte de su código!

Definiendo una función:

In [73]:

```
def am_i_happy(happiness_level):
    """ Function that prints if I'm happy or not."""

    if happiness_level >= 10:
        return "I'm very happy."
    elif happiness_level >= 5:
        return "I'm happy."
    else:
        return "I am not happy."
```

Llamar a una función:

In [74]:

```
print(am_i_happy(0))
```

```
I am not happy.
```

In [75]:

```
print(am_i_happy(5))
```

```
I'm happy.
```

Flujo de control

Hay varias oraciones de control de flujo. Los fundamentos son:

- Declaración if / else:
- bucles *Para bucles* Lista de comprensión * Mientras bucles

If/Else

In [26]:

```
sleepy = True
hungry = False

if sleepy and hungry:
    print("Eat a snack and take a nap.")
elif sleepy and not hungry:
    print("Take a nap")
elif hungry and not sleepy:
    print("Eat a snack")
else:
    print("Go on with your day")
```

```
Take a nap
```

Bucles

Iterar sobre objetos iterables, como cadenas, matrices, rango, lista, tuplas, conjuntos, dictados.

bucles 'while'

In [78]:

```
counter = 0
while (counter < 10): # this is the exit condition
    print("You have counted to", counter)
    counter = counter + 1 # Increment the counter

print("You're finished counting.")
```

```
You have counted to 0
You have counted to 1
You have counted to 2
You have counted to 3
You have counted to 4
You have counted to 5
You have counted to 6
You have counted to 7
You have counted to 8
You have counted to 9
You're finished counting.
```

bucles 'for'

Recorrer una lista:

In [79]:

```
cool_animals = ['cat', 'dog', 'lion', 'bear']
for animal in cool_animals:
    print(animal + "s are cool")
```

```
cats are cool
dogs are cool
lions are cool
bears are cool
```

Loop over a dict:

In [80]:

```
animal_sounds = {
    'dog': 'bark',
    'cat': 'meow',
    'pig': 'oink'
}

for animal, sound in animal_sounds.items():
    print("The " + animal + " says " + sound + "!")
```

```
The dog says bark!
The cat says meow!
The pig says oink!
```

Lista / comprensión dict

Otra sintaxis para usar `for-loops` en python se llama comprensión de lista. Son mucho más legible y ampliamente utilizado.

Este sintax devuelve una `lista` .

In [81]:

```
cool_animals = ['cat', 'dog', 'lion', 'bear']
text = [animal + "s are cool" for animal in cool_animals]
```

In [82]:

```
print(text)
```

```
['cats are cool', 'dogs are cool', 'lions are cool', 'bears are cool']
```

In [84]:

```
print('Applause')
```

```
Applause
```