Guía 1: Introducción a Python 3.X

Profesor: John Serrano - john.serrano@usach.cl

1 ¿Que es Python 3.X y por qué lo utilizamos en este curso?

Python es un lenguaje de programación de alto nivel, interpretado y de propósito general. Lo anterior quiere decir varias cosas:

- Python es fácil de leer y escribir. Es un lenguaje parecido al que nosotros hablamos y a diferencia de otros lenguajes de programación, su sintaxis no es tan estricta.
- Python es muy amigable para los principiantes o nuevos en el mundo de la programación: Al ser fácil de leer y no tener una sintaxis estricta, es un lenguaje perfecto para aquellos que nunca han programado
- Python es interpretado: Esto quiere decir que no debemos hacer un proceso conocido como "compilación", lo cual, en palabras simples, traduce el código que nosotros escribimos en Python a lenguaje que el computador realmente entiende (0s y 1s). En otros lenguajes de programación, este proceso de "compilación" debe ser realizado manualmente cada vez que modificamos nuestro código y queremos ejecutarlo. Python hace este proceso sin que nosotros tengamos que hacer algo adicional, por lo que solo debemos correr o ejecutar el código para poder ver inmediatamente los resultados del código.

1.0.1 IMPORTANTE CONSIDERAR

Dado que este curso es del Módulo Básico de Ingeniería (MBI), no se requiere que los códigos construidos sean eficientes, o con pocas líneas, entre otros aspectos. Se entiende que muchos pueden estar programando por primera vez, por lo que, por ejemplo, para una PEP si una persona resuelve el o los problemas en 100 líneas de código, es válido. Pero si una persona lo hace en 1000 líneas de código, también es válido. En la pauta se evalúan algunos aspectos respecto al código, como la capacidad de comentar el código, si este resuelve el problema planteado, entre otros aspectos (Esto lo iremos conversando a medida que se vaya acercando la PEP).

2 Mi primer código de Python

Cuentan las leyendas antiguas que el primer código que un programador escribió en Python fue el famoso "Hola Mundo!". En realidad, esto no se sabe si es así, pero se ha tomado como una costumbre que el primer código muestre por pantalla "Hola Mundo!". Para ello, debemos utilizar la función **print()**. Adelantando un poco el contenido, si recuerdan funciones de cálculo y/o álgebra 1, una función en Python es similar: Puede recibir algo como una entrada, hace algo y luego devuelve un resultado. En la segunda unidad del curso ya hablaremos de funciones propiamente tal, pero por ahora, el primer comando o función que debemos tener en consideración es **print()**.

print() recibe como entrada una cadena de texto, conocido en Python como un **String**. Un String en Python puede ser escrito con comillas dobles o comillas simples (CUIDADO: NO PUEDE SER UNA COMBINACIÓN DE AMBAS!). Por lo tanto, para escribir el famoso código de Hola Mundo! en Python, debemos utilizar el comando **print()** con el string "Hola Mundo!" dentro de los paréntesis de **print()**.

[1]:
```
# Mostramos por consola el mensaje "Hola Mundo!"
print("Hola Mundo!")
```

```
Hola Mundo!
```

Como se puede ver del código anterior, Python simplemente nos retorna el mensaje Hola Mundo!. Lo que hace la instrucción print() es imprimir (o "printear") un mensaje mediante la consola de Python. En otras palabras, muestra un mensaje al usuario, de acuerdo al string que se coloca dentro de los paréntesis de print.

3 Tipos de Datos en Python

Hablemos de tipos de datos. Ya hemos mencionado uno, el **String**, que corresponde a una cadena de texto o un texto simple. Los otros tipos de datos a considerar para el curso son:

- **Int (Integer o Entero):** Corresponde a un número entero, tal cual como lo conocemos en Cálculo o Álgebra. Por lo tanto, los Ints son los números negativos, los números positivos y el 0.
- **Float (Número de Punto Flotante o Decimales):** Corresponden a los números decimales, con la gran diferencia de que en Python los decimales se escriben con un punto para separar la parte entera de la parte decimal. He ahí porque se les conoce como "Punto Flotante".
- **Bool (Booleanos o Valores Lógicos):** Los booleanos puede ser solamente dos valores: **True** (Verdadero) o **False** (Falso). Se debe tener en consideración que la primera letra se escribe en mayúscula y el resto en minúsculas. Variaciones como TRUE, FALSE, true, false, TrUE, faLSE, etc no serán reconocidas por Python.
- **List (Lista):** Corresponde a un conjunto de elementos. Se escriben con un paréntesis cuadrados al inicio y un paréntesis cuadrados al final ([]) y una lista puede contener a otros tipos de datos. Ya veremos listas al final de la primera unidad.
- **Complex (Números Complejos):** Representan a los números complejos, que tienen una parte real y una parte imaginaria aunque tienen unas pequeñas variaciones de cómo funcionan realmente los números complejos. Curiosamente, en Python se utiliza la letra j en vez de la letra i para la parte imaginaria. Más curioso aún, se suelen mencionar en la primera parte del curso y luego nunca más vuelven a aparecer (Pero por si acaso igual lo incluyo en la guía).

Si tenemos dudas de si un elemento es de un tipo de dato específico, podemos utilizar la función **type()**, cuya entrada es cualquier elemento al que le queramos reconocer su tipo de dato. **type()** nos indicará si se trata de un int, float, bool, list o complex. A continuación, se presentan algunos ejemplos.

3.0.1 Ejemplos de tipo de dato: Int

[2]:
```
# Se comprueba el tipo de dato de 159
type(159)
```

[2]: int

[3]:
```
# Se comprueba el tipo de dato de -1189
type(-1189)
```

[3]: int

[4]:
```
# Se comprueba el tipo de dato de 0
type(0)
```

[4]: int

[5]:
```
# Se comprueba el tipo de dato de 2147483648
type (2147483648)
```

[5]: int

[6]:
```
# Se comprueba el tipo de dato de -2147483648
type (-2147483648)
```

[6]: int

3.0.2 Ejemplos de tipo de dato: Float

[7]:
```
# Se comprueba el tipo de dato de 5.0
type(5.0)
```

[7]: float

[8]:
```
# Se comprueba el tipo de dato de -99.23
type (-99.23)
```

[8]: float

[9]:
```
# Se comprueba el tipo de dato de 0.0
type(0.0)
```

[9]: float

[10]:
```
# Se comprueba el tipo de dato de 3.14159265359
type (3.14159265359)
```

[10]: float

3.0.3 Ejemplos de tipo de dato: String

[11]:
```
# Se comprueba el tipo de dato de "Hello World"
type ("Hello World!")
```

[11]: str

[12]:
```
# Se comprueba el tipo de dato de "Hola Mundo!"
type ('Hola Mundo!')
```

[12]:
```
str
```

[13]:
```
# Se comprueba el tipo de dato de "Hola!"
type ('Hola!")
```

```
  Cell In[13], line 2
    type ('Hola!")
          ^
SyntaxError: unterminated string literal (detected at line 2)
```

Nótese que Python nos indicó un error ya que reconoció que hay un string que no tiene las comillas al final, pero en realidad si las tiene. Esto es porque Python espera que, si al inicio se usan comillas simples, al final también deberían utilizarse comillas simples y vice versa.

[14]:
```
# Se comprueba el tipo de dato de la oración
type ("Los estudiantes de FPI de la sección A-2 deberian estar leyendo esta␣
↪guía")
```

[14]:
```
str
```

[15]:
```
# Se comprueba el tipo de dato de "True"
type ("True")
```

[15]:
```
str
```

[16]:
```
# Se comprueba el tipo de dato de "1232323"
type ("1232323")
```

[16]:
```
str
```

3.0.4 Ejemplos de tipo de dato: Bool

[17]:
```
# Se comprueba el tipo de dato de True
type(True)
```

[17]:
```
bool
```

[18]:
```
# Se comprueba el tipo de dato de False
type(False)
```

[18]:
```
bool
```

[19]:
```
# Se comprueba el tipo de dato de false
type (false)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[19], line 2
      1 # Se comprueba el tipo de dato de false
----> 2 type (false)

NameError: name 'false' is not defined
```

[20]:
```
# Se comprueba el tipo de dato de true
type (true)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[20], line 2
      1 # Se comprueba el tipo de dato de true
----> 2 type (true)

NameError: name 'true' is not defined
```

3.0.5 Ejemplos de tipo de dato: Listas

[21]:
```
# Se comprueba el tipo de dato de [1, 2, 3]
type ([1, 2, 3])
```

[21]: list

[22]:
```
# Se comprueba el tipo de dato de ["hola", "mundo", 2]
type (["hola", "mundo", 2])
```

[22]: list

[23]:
```
# Se comprueba el tipo de dato de ["hello", [3, 5, 1], 19, "a todos"]
type (["hello", [3,5,1], 19, "a todos"])
```

[23]: list

3.0.6 Ejemplo de tipo de dato: Complex

[24]:
```
# Se comprueba el tipo de dato de 2+4j
type (2+4j)
```

[24]: complex

[25]:
```
# Se comprueba el tipo de dato de -9-4j
type(-9-4j)
```

[25]: complex

[26]:
```
# Se comprueba el tipo de dato de 10j
type(10j)
```

[26]: complex

[27]:
```
# Se comprueba el tipo de dato de 0 + 0j
type (0 + 0j)
```

[27]: complex

Debemos entonces tener en consideración que cada tipo de dato tiene una forma específica en la que se escribe y también a veces existen distintas representaciones de un número en distintos tipos de datos (Por ejemplo, para el 0, existe el 0 en int, el 0.0 en float y el 0+0j en complex).

4 Transformaciones de tipos de datos

Utilizando ciertas funciones, podemos realizar transformaciones de un tipo de dato a otro. Hay que tener un poco de cuidado, ya que ciertas transformaciones no son posibles y a veces el resultado de una transformación puede darnos algo totalmente distinto a lo esperado.

- Para transformar un elemento a un número entero, podemos utilizar la función **int()**. int() solo acepta como entrada flotantes, booleanos y strings (en ciertos casos).

[28]:
```
# Se transforma el float 6.0 al int 6
int(6.0)
```

[28]: 6

[29]:
```
# Se transforma el string "6" al int 6
int("6")
```

[29]: 6

[30]:
```
# Se transforma el float 6.1 al int 6
int (6.1)
```

[30]: 6

[31]:
```
# Se transforma el float 99.343894382329 a int 99
int (99.343894382329)
```

[31]: 99

[32]:
```
# Se transforma el bool True a int 1
int(True)
```

[32]:
```
1
```

[33]:
```
# Se transforma el bool False al int 0
int(False)
```

[33]:
```
0
```

[34]:
```
# Se intenta transformar el string "h" a un int
int ("h")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[34], line 2
      1 # Se intenta transformar el string "h" a un int
----> 2 int ("h")

ValueError: invalid literal for int() with base 10: 'h'
```

Solo se pueden transformar strings que sean números a enteros. Intentar transformar algún texto o letra resultará en un error.

- Para transformar a un número flotante podemos utilizar la función **float()**. float() solo acepta ints, booleanos y strings.

[35]:
```
# Se transforma el int 3 al float 3.0
float(3)
```

[35]:
```
3.0
```

[36]:
```
# Se transforma el int 0 al float 0.0
float(0)
```

[36]:
```
0.0
```

[37]:
```
# Se transforma el bool True al float 1.0
float(True)
```

[37]:
```
1.0
```

[38]:
```
# Se transforma el bool False al float 0.0
float (False)
```

[38]:
```
0.0
```

[39]:
```
# Se transforma el string "22.32323" al float 22.32323
float ("22.32323")
```

[39]: 22.32323

[40]:
```
# Se transforma el string "22" al float 22.0
float ("22")
```

[40]: 22.0

- Para transformar a un booleano (valor lógico) podemos utilizar la función bool(). bool() solo acepta ints, floats y strings.

[41]:
```
# Se transforma el int 2 al bool True
bool(2)
```

[41]: True

[42]:
```
# Se transforma el int 0 al bool False
bool(0)
```

[42]: False

[43]:
```
# Se transforma el float 1.0 al bool True
bool(1.0)
```

[43]: True

[44]:
```
# Se transforma el float 0.0 al bool False
bool(0.0)
```

[44]: False

[45]:
```
# Se transforma el string "10.0" al bool True
bool("10.0")
```

[45]: True

[46]:
```
# Se transforma el string "0" al bool True
bool("0")
```

[46]: True

- Para transformar a una lista, podemos usar la función **list().** list() solo acepta strings.

[47]:
```
# Se transforma el string "elementos" a una lista
list("elementos")
```

[47]: ['e', 'l', 'e', 'm', 'e', 'n', 't', 'o', 's']

[48]:
```
# Se transforma el string "232312238" a lista
list("232312238")
```

[48]:
```
['2', '3', '2', '3', '1', '2', '2', '3', '8']
```

- Para transformar a un número complejo, podemos usar la función **complex()**. A diferencia de las otras funciones, complex() acepta dos números como entrada, el primero siendo la parte real y el segundo la parte imaginaria. Los números pueden estar en formato int o float.

[49]:
```
# Se utilizan los flotantes 2.4 y 3.4 para formar el complejo 2.4+3.4j
complex(2.4, 3.4)
```

[49]:
```
(2.4+3.4j)
```

[50]:
```
# Se utilizan los ints 2 y 3 para formar el complejo 2+3j
complex(2,3)
```

[50]:
```
(2+3j)
```

[51]:
```
# Se utilizan los ints 0 y 2 para formar el complejo 2j
complex(0,2)
```

[51]:
```
2j
```

[52]:
```
# Se utiliza los ints 10 y 0 para formar el complejo 10+0j
complex(10,0)
```

[52]:
```
(10+0j)
```

5 Operadores aritméticos

En Python podemos realizar operaciones matemáticas similar a como lo hacemos hoy en día en nuestro lenguaje, aunque existen algunas diferencias a tener en consideración.

- Para realizar una suma en Python, usamos el operador +

[53]:
```
# Suma de dos ints
2 + 1
```

[53]:
```
3
```

[54]:
```
# Suma de dos flots
10.23 + 33.2
```

[54]:
```
43.43000000000001
```

[55]:
```
# Suma de un int y un flot
20 + 27.2
```

[55]: 47.2

- Para realizar una resta en Python, usamos el operador **-**

[56]:
```
# Resta de dos ints
3 - 10
```

[56]: -7

[57]:
```
# Resta de dos floats
22.2 - 0.2
```

[57]: 22.0

- Para realizar una multiplicación en Python, usamos el operador *

CUIDADO: No se utiliza el punto ni tampoco x para las multiplicaciones, solo el asterisco.

[58]:
```
# Multiplicación de dos ints
3 * 2
```

[58]: 6

[59]:
```
# Multiplicación de dos floats
10.5 * 10.5
```

[59]: 110.25

[60]:
```
# Multiplicación de un float y un int
10000000000000000.0 * 0
```

[60]: 0.0

- Para realizar una división en Python, usamos el operador /. No existe la división con dos puntos. Los resultados automáticamente se transforman en números de tipo float, independiente de que los números utilizados y el resultado sean ints.

[61]:
```
# División de dos ints
50 / 5
```

[61]: 10.0

[62]:
```
# División de dos ints
53 / 2
```

[62]: 26.5

[63]:
```
# División de un float y un int
100.0 / 10
```

[63]: 10.0

[64]:
```
# División de dos floats
27.2 / 9.2
```

[64]: 2.956521739130435

- Para realizar una división entera, usamos el operador //. Aquí los resultados serán ints a menos que se utiliza un número flotante. Basta que uno de los números sea float para que el resultado sea float.

[65]:
```
# División entera de dos floats
27.2 // 9.2
```

[65]: 2.0

[66]:
```
# División entera de dos ints
100 // 10
```

[66]: 10

[67]:
```
# Diisión entera de dos ints
50 // 5
```

[67]: 10

[68]:
```
# División entera de dos floats
100.0 // 10.0
```

[68]: 10.0

[69]:
```
# División entera de un int y un float
64 // 8.0
```

[69]: 8.0

- Para obtener el resto de una división, se utiliza el operador %. Este operador se conoce como **modulo** y no tiene que ver con el porcentaje.

[70]:
```
# Modulo (resto de la división) entre dos ints
5 % 2
```

[70]: 1

[71]:
```
# Modulo (resto de la división) entre dos ints
10 % 2
```

[71]: 0

[72]:
```
# Modulo (resto de la división) entre un int y un float
572 % 127.3
```

[72]:
```
62.80000000000001
```

[73]:
```
# Modulo (resto de la división) entre un int y un float
573.2 % 198.4
```

[73]:
```
176.40000000000003
```

Para elevar un número se utiliza el operador **. Esta operación también se conoce como exponenciación.

[74]:
```
# Un int elevado a otro int
2**6
```

[74]:
```
64
```

[75]:
```
# Un int elevado a otro int
20**2
```

[75]:
```
400
```

[76]:
```
# Un float elevado a un int
8.0**3
```

[76]:
```
512.0
```

[77]:
```
# Un int elevado a un float
4 ** 2.2
```

[77]:
```
21.112126572366314
```

6 Orden en operadores aritméticas

Operación	Operador	Aridad	Asociatividad	Precedencia
Exponenciación	**	Binaria	Derecha	1
Identidad	+	Unaria	-	2
Negación	-	Unaria	-	2
Multiplicación	*	Binaria	Izquierda	3
División	/	Binaria	Izquierda	3
Módulo o resto	%	Binaria	Izquierda	3
División parte entera	//	Binaria	Izquierda	3
Suma	+	Binaria	Izquierda	4
Resta	-	Binaria	Izquierda	4

La tabla anterior describe el orden que se debe respetar junto con la asociatividad. Es importante tener en consideración que al igual que en Cálculo y/o Álgebra, el uso de paréntesis puede modificar el orden de operaciones

[78]:
```
# Operacion sin parentesis
5 + 5 ** 3 / 5 * 4 - 10 * 3
```

[78]: 75.0

[79]:
```
# Operación con parentesis (cambia el orden dado los parentesis, los cuales␣
↪tienen preferencia)
(5 + 5) ** 3 / 5 * 4 - 10 * 3
```

[79]: 770.0

7 Variables y asignación

En Python, una variable es un lugar donde podemos guardar un valor en memoria para poder reutilizarlo en un futuro. Una variable puede ser modificada y es ahí donde se deben guardar valores importantes del problema, resultados, valores temporales, entre otros. Por buenas prácticas, siempre es recomendable colocarle a una variable un nombre representativo de lo que es, aunque se le puede colocar cualquier nombre en realidad siempre y cuando el nombre no contenga caracteres no aceptados por Python o espacios al inicio. Si se requiere un nombre de variable que lleve espacios,

los espacios deben reemplazarse por guiones bajos (EJ: nombre_variable) y si la variable contiene algo que no va a cambiar durante toda la ejecución del código, es decir, un valor fijo, entonces el nombre debe estar en mayúsculas (EJ: NOMBRE_VARIABLE). Para asignar un valor a un elemento, debemos utilizar el operador =. Podemos utilizar la instrucción **print()** para ver en la consola el contenido de una variable.

[80]:
```
# Guardo el valor 2 (int) en una variable de nombre "variable"
variable = 2

# Imprimo el contenido de la variable "variable"
print(variable)
```

```
2
```

[81]:
```
# Guardo el string "John" en la variable "nombre"
nombre = "John"

# Imprimo el contenido de mi variable junto con el mensaje "Hola! Mi nombre es:"
print("Hola! Mi nombre es:" , nombre)
```

```
Hola! Mi nombre es: John
```

Nótese que el ejemplo anterior demuestra que **print()** acepta distintas cantidades de strings, los cuales deben estar separados por una coma (,).

Las variables cuyo valor no cambian se conocen como **Constantes** y como se dijo anteriormente, deben escribirse en mayúsculas. Hay que tener un poco de cuidado al nombrar las variables, ya que, si bien tenemos libre libertad de colocar cualquier nombre de variable, existen algunas palabras reservadas que no pueden ser utilizadas como nombres de variables.

En las siguientes tablas se presentan nombres de variables y constantes adecuados y no adecuados para el curso:

Ejemplo	Correcto/Incorrecto
resultado	Correcto
sumaAcumulada	Incorrecto: las palabras se deben separar por (_)
suma_parcial	Correcto
x1	Incorrecto: nombre no representativo
cosa2	Incorrecto: nombre no representativo

Ejemplo	Correcto/Incorrecto
PI	Correcto
NUMERO_E	Correcto
gravedad	Incorrecto: no usa mayúsculas
numeroPi	Incorrecto: no usa mayúsculas

Por último, mencionar que en Python NO debemos utilizar tildes ni la letra Ñ, dado que eso puede ocasionar problemas con la interpretación de nuestro código al ejecutarlo (dado que Python se rige por el inglés más que por el español).

8 Extra (Pero importante para poder realizar los ejercicios): Entradas

Si utilizamos la función **input()**, podemos recibir una entrada por teclado por parte del propio usuario cuando se ejecute el código. Esta entrada puede ser guardada en una variable y dentro de los paréntesis de input podemos escribir un mensaje en formato String para indicar al usuario cual es el dato que se espera que se ingrese.

[82]:
```
# Se solicita que el usuario ingrese su edad
edad = input("Ingrese su edad: ")
```

```
Ingrese su edad:  23
```

Si se utiliza la función type() que se mencionó al inicio de la guía, se puede comprobar que edad es una variable de tipo string, a pesar de que se ingresó un número por el teclado. Esto es porque Python automáticamente transforma todas las entradas a formato String por defecto.

[83]:
```
# Se comprueba el tipo de dato de la variable edad
type(edad)
```

[83]:
```
str
```

Entonces, si queremos que una entrada si o si sea de un tipo de dato especifico, podemos combinar las funciones de transformación que también se vieron en la guía para transformar la entrada.

[84]:
```
# Se combina el uso de int() con input() para obtener una entrada con tipo de␣
 ↪dato int
edad = int(input("Ingrese su edad: "))
type(edad)
```

```
Ingrese su edad:  23
```

[84]: int

[85]:
```
flotante = float(input("Ingrese un numero flotante: "))
type (flotante)
```

```
Ingrese un numero flotante:  3
```

[85]: float