*Clase 03. Introducción a la programación con Python (Parte II)*

# Nociones básicas de programación

## Variable, asignación, expresiones

En esta sección veremos los bloques conceptuales más básicos de la programación con Python.

Pensemos en la siguiente expresión matemática: x + y = z

En esta expresión queremos representar que la suma de los números x e y dará como resultado el número z.

Pensemos ahora esto en términos de programación: si quisiéramos hacer un programa que realice una suma, podemos plantearlo de la siguiente manera:

* Ingresar un número, que cumplirá el rol de la x
* Ingresar otro número, que cumplirá el rol de la y
* Decirle a la computadora que efectúe la suma de los dos números
* Al resultado lo haremos cumplir el rol de la z
* Mostrar al usuario el valor de z, esto es, el resultado de sumar los dos números ingresados x e y.

Todo este proceso está pensado detalladamente para programación.

Toda vez que tenemos un símbolo al que le damos un valor concreto (por ejemplo x = 2), estamos haciendo una asignación a una variable. El símbolo x es una variable porque puede tomar cualquier valor. En este caso particular le damos (mediante el proceso de asignación) el valor 2.

En Python podemos trabajarlo de la siguiente manera:

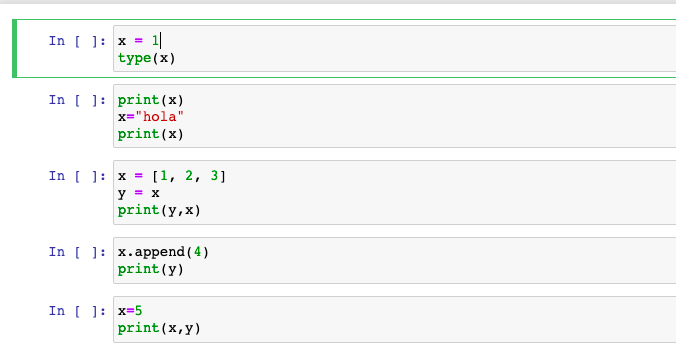
| x = 2 y = 3 z = x + y z 5 |
| --- |

La primera línea asigna a la variable x el valor 2. La segunda línea asigna a la variable y el valor 3. La tercera línea suma x e y. La cuarta línea muestra el resultado.

Cada parte del código “compuesta” que represente un valor en concreto (por ejemplo x + y) es una expresión. En principio, podemos hacer la analogía con las expresiones matemáticas sin inconvenientes.

### Ejercicios

¿Cuáles son las salidas de los siguientes bloques de código?



## Objetos y punteros

En Python todo es un objeto. Un objeto, en términos de programación, es una entidad que tiene datos y además tiene metadatos (atributos o propiedades) y funcionalidad propia (métodos). Consideremos el siguiente caso:

| x=1.0 x.is\_integer()  x=1.4 x.is\_integer() |
| --- |

En estos casos, is\_integer() es una funcionalidad de x que dice si es entero o no

| print(x.real,x.imag) |
| --- |

Aquí, real e imag son propiedades de x que muestran su parte real y su parte imaginaria.

Por otra parte, el ejemplo de la sección anterior nos muestra la forma de trabajar con punteros (pointers). A diferencia de otros lenguajes donde las variables se comportan como contenedores de datos, las variables en Python son punteros. Esto significa que no contienen un valor, sino que apuntan a un dato que se encuentra en la memoria. De esta forma, si asignamos una variable a otra (y = x), simplemente le estamos diciendo a ambas que apunten al mismo dato (la lista [1, 2, 3]). En su forma más específica, la operación “=” realiza la conexión entre el puntero y el dato. En cambio, la aplicación de métodos (por ejemplo append()) cambia el dato de destino.

Esta diferencia es sutil y en la práctica no afecta el trabajo de Data Science. No obstante trae mejoras sustanciales en la performance y es importante tenerla en cuenta para no cometer errores en la programación.

## Operaciones

Python define una serie de operaciones básicas. Estas operaciones se muestran a continuación:

Las operaciones aritméticas son las más básicas. Prestar atención a algunas específicas que pueden ser de mucha utilidad.

| Aritméticas |  |
| --- | --- |
| a + b | Suma |
| a - b | Resta |
| a \* b | Multiplicación |
| a / b | División |
| a // b | División entera (resultado sin decimal) |
| a % b | Módulo (resto de la división entera) |
| a \*\* b | Exponenciación |
| -a | Negativo |

Además del operador de asignación “=”, existen los siguientes, que son equivalentes a las operaciones de la columna de la derecha, y nos pueden hacer ahorrar mucho tiempo.

| Asignaciones |  |
| --- | --- |
| a += b | a = a + b |
| a -= b | a = a - b |
| a \*= b | a = a \* b |
| a /= b | a = a / b |
| a //= b | a = a // b |
| a %= b | a = a % b |
| a \*\*= b | a = a \*\* b |

Los comparadores sirven para, precisamente, comparar objetos. Se comportan de manera análoga a los correspondientes símbolos matemáticos que representan.

| Comparadores |  |
| --- | --- |
| a == b | a igual a b |
| a != b | a distinto de b |
| a < b | a menor a b |
| a > b | a mayor a b |
| a <= b | a menor o igual que b |
| a >= b | a mayor o igual que b |

Los operadores de identidad van más al fondo del detalle de los objetos y punteros. Si dos objetos son el mismo (operador “is”), significa que apuntan al mismo dato. Por otra parte, si queremos saber si un dato está contenido en otro, no importa en qué parte, nos será de mucha utilidad el operador “in”. Esto último se usa frecuentemente en las listas de datos.

| **Identidad y pertenencia** |  |
| --- | --- |
| a is b | a es el mismo objeto que b |
| a is not b | a no es el mismo objeto que b |
| a in b | a está contenido en b |
| a not in b | a no está contenido en b |

# Estructuras de control

Para controlar el flujo de ejecución de un programa, utilizaremos estructuras de control. Existen varias estructuras de acuerdo a los objetivos que persigamos. Veremos los más utilizados en Data Science a continuación.

## Estructura FOR

La estructura FOR repite un comando una cantidad fija de veces. Veamos algunos ejemplos:

| for i in range(1,10):  print(i) for i in [1,4,6,2]:  print(i) |
| --- |

## Estructura WHILE

La estructura WHILE repite una secuencia de comandos “mientras” una condición se cumpla. Cuando la condición no se cumple más, termina la repetición.

| i = 1 while i < 10:  print(i)  i += 1 |
| --- |

## Estructura condicional (IF)

La estructura condicional trabaja con opciones de código. Si se cumple una condición, se ejecuta una secuencia de comandos. En caso contrario, se ejecuta otra secuencia de comandos. Pueden manejarse más de dos opciones con el comando elif. Veamos un ejemplo.

| x = 1 if x < 10:  print(x, "es menor a 10") elif x > 10:  print(x, "es mayor a 10") else:  print(x, "es 10") |
| --- |

### Ejercicios

* El factorial x! se define como el producto de todos los números desde 1 a x. Escribir el código para calcular el factorial de 5.
* J. B. Büttner, maestro de un colegio alemán, castigó a todos los niños poniéndoles como tarea sumar los 100 primeros números naturales para tenerlos entretenidos y callados un buen rato. Carl Friedrich Gauss obtuvo la respuesta casi de inmediato, encontrando una fórmula para la suma de una serie de números naturales.   
  No pedimos aquí llegar a la fórmula. Lo que pedimos en este ejercicio es hacer un programa que lo calcule sin utilizar la fórmula.

Ejercicio 1:

| fac = 1 for i in range(1,6):  # print(fac)  fac \*= i fac |
| --- |

Ejercicio 2:

| suma = 0 for i in range(1,101):  print(i, suma)  suma += i suma |
| --- |

# Funciones, argumentos y retorno

Una forma de organizar los comandos que se utilizan en forma repetitiva es definir funciones. Una función es un agrupamiento de instrucciones que tienen un nombre definido. Las funciones pueden tener argumentos, o sea entradas de datos. Finalmente, las funciones tienen un valor de retorno, esto es, la salida que se obtiene al ejecutar la función. Veamos un ejemplo:

| print(x) print(x,y) |
| --- |

Las funciones se escriben con un nombre y los argumentos entre paréntesis. Si no existen argumentos, se escriben igualmente los paréntesis vacíos “()”

Además de las muchas funciones que ofrece Python, podemos incorporar funciones propias. Veamos lo siguiente:

| def suma(x,y):  z = x + y  return z |
| --- |

luego ingresemos:

| suma(2,3) |
| --- |

### Ejercicios

* Escribir una función para calcular el factorial de un número cualquiera
* Escribir una función para calcular la suma de una serie comenzando por un número cualquiera y terminando en otro número que debe ser mayor al primero.

# Tipos de datos

Python maneja dos grupos de tipos de datos: simples y estructurados.

## Datos simples

Los datos simples son los siguientes:

| Tipos de datos |  |  |
| --- | --- | --- |
| int | x = 1 | Enteros |
| float | x = 1.0 | Punto flotante (decimales) |
| complex | x = 1 + 2j | Complejos (parte real e imaginaria) |
| bool | x = True | Booleanos o lógicos: verdadero / falso |
| str | x = 'abc' | Texto |
| NoneType | x = None | Tipo especial para indicar valores nulos |

Podemos saber el tipo de un dato x con la función type(x).

## Datos estructurados

Los tipos de datos estructurados son agrupamientos de tipos de datos simples. A continuación se muestran los tipos de datos estructurados predeterminados en Python:

| Estructurados | |  |
| --- | --- | --- |
| list | [1, 2, 3] | Lista ordenada |
| tuple | (1, 2, 3) | Lista ordenada inmutable |
| dict | {'a':1, 'b':2, 'c':3} | Diccionario: conjunto de pares clave:valor |
| set | {1, 2, 3} | Conjunto, a la manera de un conjunto matemático |

Decimos que la estructura list es *mutable* porque permite que sus elementos sufran modificaciones una vez definida. Es decir, siempre podremos cambiar a nuestro antojo los elementos que conforman la lista. Por otro lado, las *estructuras inmutables* como las tuplas no admiten esta reasignación de elementos en tiempo de ejecución del programa.

Otro ejemplo de estructura mutable es la *estructura dict, de manera que siempre* podremos modificar los valores de un dict. Sin embargo, cabe destacar que no es posible modificar sus claves.

Las formas de organizar los tipos de datos estructurados tienen distintas aplicaciones de acuerdo a cómo está pensado cada uno. Para comenzar en Data Science, usaremos el tipo de dato list.