” Python Basics for Data Science” - Resumen escrito en LaTeX

Federico E. Benelli

November 4, 2019

Tabla de contenidos

1 Introducción

Python es un lenguaje de programación

2 Basics

2.1 Tipos básicos de datos

Integer Números enteros

Float Números flotantes. Es como se denomina a los números decimales en programación, esta denominación viene dada porque por defecto las computadoras no realizan con total precisión cálculos de fracciones. Por ejemplo, si en la consola de Python se escribe "0.1 + 0.2" esta devolverá "0.30000000000000004" en lugar del esperado "0.3" (explicación más detallada en https://www.youtube.com/watch?v=PZRI1IfStY0)

String Strings son secuencias de caracteres (usualmente texto), se escriben entre " " o ' ', también se pueden hacer bloques de texto entre """ """

Boolean Son los booleanos "True" y "False"

2.2 Funciones básicas de Python

Se describen las funciones básicas presentes en Python 3.7.

type(x) Retorna el tipo de variable que es x, por ejemplo si realizo:

>> type('texto de ejemplo')
>><class 'str'>

print(x) Muestra en la consola la variable x, soporta muchos tipos de variables, no solo string

>> print('texto de ejemplo')
>> texto de ejemplo

round(x) Redondea el valor de un número flotante x, por defecto redondea totalmente

>> round(2.53)
>> 3
>> round(2.53,1)
>> 2.5

float(x) Retorna el valor de x convertido en tipo número flotante, puede recibir íntegros o strings, siempre y cuando la string sea solo un número.

>> float("2")
>> 2.0

int(x) Retorna el valor íntegro de x, solo puede recibir variables del tipo float o string si es que esta es solo un int. Si utilizo la función int en el booleano "True" obtengo un 1, y si lo realizo en el booleano "False" obtengo un 0 >> int(2.731)
>> 2

2.3 Expresiones en Python

En Python pueden realizarse las operaciones básicas de matemáticas, utilizando '*' y '**' para la multiplicación y potenciam, respectivamente. Además se pueden utilizar el símbolo '%' para obtener el resto de una división y '//' para obtener el cociente entero. En el caso de strings, estas también pueden ser sumadas y multiplicadas por un íntegro.

Ejemplos

>> 5+5 >> 10 >> "abc"+"def" >> "abcdef"	>> 5-5 >> 0 >> 5*5 >> 10	>> 5/5 >> 1 >> 5**5 >> 125	>> 25%4 >> 1 >> 25//4 >> 6

2.4 Variables en Python

Las variables se utilizan para almacenar valores, los cuales pueden ser reutilizados luego, las variables también pueden almacenar el resultado de expresiones.

>> variables = 25 + 25
>> variable
>> 50

Si cambio el valor de una variable, esto afectara el valor de otras variables que la incluyan

>> variable1 = 25
>> variable2 = variable1+25
>> variable2
>> 50

Al reasignar variable1 y volver a llamar a variable2

>> variable1 = 50
>> variable2
>> 75

3 Data Structure

3.1 Tipos de estructuras de datos

list Las listas son conjuntos ordenados de valores de cualquier tipo de dato, se expresan como [dato1, dato2, dato3, etc].
Se almacenan como variables y pueden contener variables dentro de ellas.

>> variable1 = 25
>> variable2 = variable1+25
>> lista = [variable1, variable2, 'tercer elemento', 4.23]

Los elementos de las listas se llaman de la siguiente manera:

>> **lista[0]** *Llamado al primer elemento de la lista, en Python todos los contadores comienzan en cero, devuelve 25*

>> **lista[-1]** *Llamado al último elemento de la lista, devuelve 4.23*

>> **lista[1:3]** *Llama a los elementos del 1 al 3 de la lista, en este caso devuelve* [50, 'tercer elemento', 4.23]

>> **lista[a:b:c]** *Llama a los elementos de 'a' a 'b', yendo de c en c, si no se dan valores de a y/o b se considera la lista completa,* lista[::2] *devuelve* [25,'tercer elemento']*, si el valor de c es negativo el conteo será de fin a principio*

Las listas son mutables, eso quiere decir que sus elementos pueden ser modificados.
>> lista = [1, 2, 3]
>> lista[1] = 25
>> lista
>> [25, 2, 3]

Si defino una variable a una lista y luego otra variable a esa variable, cualquier cambio que le realice a la lista de la primer variable también será realizado en la segunda variable, esto ocurre debido a que ambas variables apuntan a la misma lista ubicada en la memoria.
>> lista1 = [1, 2, 3]
>> lista2 = lista1
>> lista1[1] = 23
>> lista2
>> [1, 23, 3]

Esto puede ser evitado 'clonando' a la primer variable, esto se realiza llamando a todos los elementos de la primer variable al momento de definir la segunda
>> lista1 = [1, 2, 3]
>> lista2 = lista1[:]

Métodos aplicables en listas

lista.extend(iterable) Añade los elementos de un iterable a una lista.

lista.append(object) Añade el objeto al final de la lista.

Más métodos y funciones pueden consultarse con help(list())

tuple Las tuplas son similares a las listas, solo que se definen entre paréntesis (elemento1, elemento2, elemento3, etc) y son inmutables, por lo que una vez que son definidas no pueden ser modificadas.
Los elementos de las tuplas se llaman de igual manera que en las listas.

set Son conjuntos de elementos que no poseen un orden en particular, se definen como elemento1, elemento2, etc, pueden ser vistas como conjuntos de *Venn*. Los elementos repetidos se eliminan automáticamente

Métodos

set.add(element) Añade el elemento al set.

set.remove(element) Remueve el elemento del set.

set1.intersection(set2) Devuelve un set con la intersección de ambos sets.

set1.union(set2) Devuelve un set con la unión de ambos sets.

set1.difference(set2) Devuelve un set con los elementos fuera de la intersección de ambos sets.

set1.issubset(set2) Devuelve True o False dependiendo si es subset o no.

set1.issuperset(set2) Equivalente al anterior

Más métodos pueden hallarse con help(set())

dictionary Los diccionarios son conjuntos de elementos con índices para facilitar el acceso a los elementos. Se definen como {"índice1":{elemento1,elemento2},"índice2":{elemento3, elemento4}} Los elementos de los diccionarios se definen y llaman de la siguiente manera:

>> **dic['índice1'] = {elemento1,elemento2}** Asigna el conjunto de elementos al índice 1, si el índice 1 no existía aún lo crea automáticamente.

>> **dic['índice1']** Devuelve los elementos correspondientes al índice 1, elemento1, elemento2

4 Programming Fundamentals

4.1 Condiciones

En Python pueden realizarse distintos tipos de comparaciones entre objetos/expresiones, las cuales devuelven un valor booleano del tipo *True* o *False* dependiendo de la comparación.
Los distintos tipos de comparaciones que se pueden realizar en Python son:

A > B Devuelve *True* si A es mayor que B

A < B Devuelve *True* si A es menor que B

A == B Devuleve *True* si A es igual a B (funciona con cualquier tipo de objeto, no solo números)

A != B Devuelve *True* si A es distinto a B (con cualquier tipo de objeto)

A >= B Devuelve *True* si A es mayor o igual a B

A <= B Devuelve *True* si A es menor o igual a B

Estas comparaciones son útiles a la hora de condicionar la ejecución de parte del código, así esta parte solo es ejecutada si y solo si cumple con cierta condición. para esto se utiliza la declaración *if* de la siguiente manera:.
if *condicion*:
....código a utilizar si se cumple la condición
else:
....código a utilizar si la condición no se cumple (opcional)

4.2 Loops

Los loops son formas de simplificar código que va a repetirse reiteradas veces o de manera iterativa, también facilitando su modificación en el caso de que sea necesario, por ejemplo, existen dos maneras de hacer que un código repita 5 veces el comando *print("Hello, world!")*:

```
print("Hello, world")
print("Hello, world")
print("Hello, world")
print("Hello, world")
print("Hello, world")
```

```
for i in range(0,5):
....print("Hello, world!")
```

Siguiendo con el ejemplo, si se quisiera cambiar el código por uno el cual la línea se escriba 20 veces en el primer caso sería necesario reescribir esa misma linea 15 veces más cuando en el segundo caso con simplemente cambiar

el número "5" por un "20". Queda claro que la segunda opción es mucho más simple, y al mismo tiempo estas complejidades incrementan exponencialmente si el código se vuelve más complejo.

while La declaración *while* se utiliza de manera similar a la declaración *if*, funciona con una condición y, a diferencia de la declaración *if* que reproduce el código una sola vez, esta repite el bloque de código hasta que la condición sea *True*

for La declaración *for* funciona designando valores a una nueva variable dentro de esta, reasignándole valores en función de un objeto iterable.

4.3 Funciones

4.4 Objetos y clases

5 Working with Data

5.1 Leyendo y escribiendo archivos

5.2 Cargar data con Pandas

6 Working with numpy arrays

References

[1] IBM, *IBM: PY0101EN Python Basics for Data Science*, edX.org, 2017.