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

Federico E. Benelli

14 de noviembre de 2019

Índice

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.^{es}ta 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 "Trueτ "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.^{o}btengo un 1, y si lo realizo en el booleano "False.^{o}btengo 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:
    codigo a utilizar si se cumple la condicion
else:
    codigo a utilizar si la condicion 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.

Como se dijo previamente, la declaración *for* designa un iterable, usualmente los iterables que se utilizan son:

list En el caso de declarar *for element in list:*, la varibale *element* tomará los valores de los elementos de la lista en la iteración. También puede utilizarse la función *enumerate*, con la cual además de iterar sobre los valores de la lista se obtiene el índice de cada elemento, la declaración se realiza de la forma *for i,element in enumerate(list):*

range(in,fin,step) La función *range(in,fin,step)* devuelve un objeto iterable el cual consiste en un rango de números, donde *in* es el número inicial, *fin* el número final y *step* el paso en el cual se realiza el conteo. La declaración será, por ejemplo la declaración *for i in range(0,15,2):* asignará a *i* los valores entre 0 y 15, de dos en dos. Nótese que el valor final jamás es alcanzado, por lo que si utilizo *range(0,10)* se obtendrán los valores 0,1..9.

dictionary Si se itera en un diccionario mediante la declaración *for* la variable asignada obtendrá los valores de las *"key"* del mismo, en formato *string*

Nótese que si bien existen convenciones de utilizar el nombre de variable *'i'* en una declaración, no es una condición necesaria para que el código funcione, esta puede tomar cualquier nombre designado por el autor.

4.3. Funciones

En Python pueden utilizarse funciones para simplificar código que se repite en múltiples etapas. las funciones se declaran y llaman de la siguiente manera:

```
def function(input1,inputb,...):
    metodo a implementar
    return result

a = 2
b = 3
function(a,b)
```

Una función aplica métodos al momento de ser llamada, las variables utilizadas dentro de ellas no afectan a las variables globales del código aunque pueden leerlas, pueden recibir elementos sobre los cual trabajar y devuelve un resultado (también puede aplicar un método que no devuelve resultado). Se pueden generar variables globales dentro de una función antecediendo a la misma con *"global"*

```
def function(input1,input2,...):
    result = input1 + input2
    global variable_nueva
    variable_nueva = result + 10
    return result

a = 2
b = 3

suma = function(a,b)

print(suma)
>> 5
print(variable_nueva)
>> 15
```

4.4. Objetos y clases

Una clase es una definición general de un elemento el cual posee ciertos atributos y al cual se le pueden ser aplicados ciertos métodos, por ejemplo, una lista es un objeto al cual pueden aplicársele métodos como **.append(elemento)* Los pasos de utilización de una clase se van a realizar mediante el siguiente ejemplo

4.4.1. Definición de una clase

Se va a definir una clase llamada *Persona*, la cual posee características básicas de una persona y se va a tener un método mediante el cual puede modificarse su domicilio.

```
1  """
2  En primer medida se define el nombre de la clase y puede utilizarse una clase "madre" de la
       cual se heredan todos los metodos y atributos de ese tipo de clase ademas de los que se
       agregan en la defiinicion, por el momento solo se utilizara "object"
3  """
4  class Persona(object):
5
6      def __init__(self, nombre, apellido, domicilio, edad):
7          """
8          Los atributos del nuevo objeto son definidos por la funcion "__init__", la cual
       siempre comienza con el parametro "self", el cual hace referencia al objeto y luego es
       seguida por los atributos.
9          Dentro de la funcion los atributos son definidos como self.atributo = atributo
10         """
11         self.nombre = nombre
12         self.apellido = apellido
13         self.domicilio = domicilio
14         self.edad = edad
15
16     def info():
17         """
18         Metodo que devuelve los datos del objeto al ser utilizado
19         """
20         print(self.nombre)
21         print(self.apellido)
22         print(self.domicilio)
23         print(self.edad)
24
25     def mudar(nueva_direccion):
26         """
27         Metodo que reasigna un nuevo domicilio al objeto
28         """
29         self.domicilio = nueva_direccion
```

Listing 1: Definición de una clase

4.4.2. Utilización de clases

```
1
2  # Comienzo definiendo variables a un objetos Persona
3  alumno_1 = Persona("Jose", "Perez", "Independencia 1153, Cordoba", "28")
4
5  # Si no se aclara el atributo el programa asume que es el de la misma posicion que en la
       definicion de la Clase (obviando el "self"), la siguiente definicion (si bien caotica) es
       valida
6  alumno_2 = Persona(direccion="Santa Fe 24, Cordoba", "Gonzalez", nombre="Marcos" ,"52")
7
8  # Ahora pueden aplicarse los metodos o llamarse a los atributos individuales
9
10 print(alumno_1.edad)
11 >> 28
12
13 alumno_2.info()
14 >> Marcos
15 >> Gonzalez
16 >> Santa Fe 24, Cordoba
17 >> 52
18
19 alumno_1.mudar("Bv. Chacabuco 729, Cordoba")
20 alumno_1.info()
21 >> Jose
22 >> Perez
23 >> Bv. Chacabuco 729, Cordoba
24 >> 52
```

Listing 2: Utilización de objetos de una clase, output de consola presentado como " >> "

5. Working with Data

5.1. Leyendo y escribiendo archivos

Los archivos pueden ser abiertos y modificados en Python utilizando la funcione *open(a, b)*, donde *a* es el nombre del archivo y b el modo en el cual se abre el archivo (*r* para leer y *w* para escribir), en ambos casos la funcion *open()* genera un objeto al cual es necesario aplicarle métodos para poder utilizarlo. A la hora de leer archivos los dos métodos principales a utilizar son **.read()*, **.readline()* y *.readlines(). Los archivos abiertos con la función *open()* se leen caracter por caracter cada vez que se aplica un método y la posición del cursor queda registrada por lo que no se puede volver a leer caracteres ya leidos.

```
1  # Defino una variable al archivo, en modo de lectura
2  feed = open("archivo.txt", "r")
3
4  # El metodo "read()" lee todos los datos del archivo y los devuelve como string
5  datos = feed.read()
6
7  # El metodo "readline()" lee linea por linea cada vez que es llamado y devuleve una string.
8  # En este caso, ambas variables tendran el valor de una string nula debido a que todo el
       archivo fue leido previamente al utilizar feed.read()
9
10 primer_linea = f.readline()
11 segunda_linea = f.readline()
12
13 # El metodo "readlines()" lee todas las lineas y las devuelve como una lista, donde cada
       elemento es una linea
14 lineas = f.readlines()
15
```

```
# Al terminar de leer el archivo este debe cerrarse, lo recomendable es cerrarlo lo antes
    posible para evitar perdida de datos en caso de haber un error en el programa
f.close()

# Una forma mas facil de trabajar es la siguiente, se utiliza el modo de escritura en el
    ejemplo pero puede usarse en lectura tambien. mediante la declaracion "with" el archivo se
     abre y es definido con la variable declarada luego de "as", el archivo se cierra
    automaticamente tras correr el codigo del bloque indentado
with open(file, "w") as w:
    nuevos_datos = "Estoy sobreescribiendo datos en el archivo!"
    w.write(nuevos_datos)

# Si quisiera agregar datos al archivo debo primero leerlo
with open(file, "r") as f:
    feed = f.read()

with open(file, "w") as w:
    nuevos_datos = "esta vez voy a agregar datos a los existentes"
    feed += nuevos_datos
    w.write(feed)
```

Listing 3: Leer y escribir datos de un archivo

5.2. Cargar data con Pandas

Pandas es una librería de Python con foco en estructuras de datos y herramientas de análisis de datos fáciles de utilizar y de alta performance.[2]

En Pandas se trabajo con objetos llamados *DataFrames*, los cuales son equivalentes a las listas en Python, y usualmente se trata de tablas de dos dimensiones (similar a una tabla Excel)

5.2.1.

5.2.2. Métodos de Pandas

df.head() Devuelve las primeras 5 filas del DataFrame

df[['columna']] Devuelve la columna de título 'columna', también puede utilizarse el número de columna

df.loc['Fila1':'Fila2', 'Columna2':'Columna5'] Devuelve los valores entre las filas y columnas designadas en función de sus títulos

df.iloc[a:b, c:d] Devuelve los valores entre entre las filas a y b, y las columnas c y d; donde a, b, c y d son los valores de índice numéricos.

Más información puede consultarse en la documentación de Pandas [3]

6. Working with NumPy arrays

Referencias

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

[2] Pandas, PyData.org, url=https://pandas.pydata.org/

[3] Pandas Documentation, PyData.org, url=`https://pandas.pydata.org/pandas-docs/stable/`