Básicos de Python, IPhython & Jupyter Notebook

Manuel I. Gómez G.

8 de diciembre de 2018

Resumen

En el presente trabajo daremos un vistazo a los comandos más básicos de Python con la fin de comenzar a programar códigos sencillos, aprender acerca del manejo de Jupyter Notebook, la cual es una aplicación que permite programar en Python con mucha facilidad y también veremos algunas de la librerías más utilizadas al momento de realizar análisis de datos.

1. Python, un lenguaje interpretador

En la programación existen dos tipos de lenguajes: los compilados, aquellos donde después de haber creador todo el código de nuestro programa procedemos a compilarlo, ésto consiste en crear un ejecutable que nos permita correr el archivo; por otro lado, los interpretadores son lenguajes en los que la programación se da por bloques de código pues a medida que terminamos uno de estos se ejecuta y procedemos a realizar el siguiente bloque, así sucesivamente hasta terminar nuestro programa.

Python es un lenguaje de cómputo interpretador, por lo tanto, corre un programa mediante una declaración a la vez. Ejecutar Python es tan sencillo como abrir una terminal y escribir **python** en ella, y tan sencillo de cerrar como usar el comando **exit ()** o presionar la combinación **Ctrl-D**.

```
Manuels-MacBook-Pro:~ migg98$ python
Python 3.6.5 |Anaconda, Inc.| (default, Apr 26 2018, 08:42:37)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> exit()
Manuels-MacBook-Pro:~ migg98$
```

Figura 1: Ejecución y cierre de python desde la terminal.

Para abrir archivos de python simplemente ejecutamos el comando para iniciarlo seguido del nombre de nuestro archivo con terminación *.py*.

2. Jupyter Notebook para análisis de datos

Python es una lenguaje de programación fácil de aprender y bastante útil para el análisis de datos, y es así que surgen los cuadernillos de trabajo, entre ellos tenemos el creado por Project Jupyter, el Jupyter Notebook.

Para comenzar a trabajar con este cuadernillo es necesario ir a la dirección donde se encuentran los archivos que usaremos en nuestro código (archivos de texto, imágenes, datos numéricos, etc) desde la terminal, posteriormente escribir el comando **juyter notebook**, al hacer esto se genera un link, el cual se muestra en nuestro navegador, dándonos acceso al cuadernillo de trabajo donde podremos hacer uso de Python y ciertas facilidades que proporciona Jupyter Notebook.

Comenzamos creando un archivo de **Python3** desde la interfaz de Jupyter, esto nos genera nuestro cuadernillo donde lo primero que veremos será un bloque de código. La ventaja de usar Jupyter es la implentación de las sugerencias al introducir un comando, el autocompletado de aquello que escribimos, la interfaz que presenta, la visualización de la información que estamos manejando y generando, entre otras.

2.1. Autocompletado

Una función interesante es el **autocompletado**, basta con simplemente usar la tecla **TAB** para que se despliegue una lista de opciones con comando del sistema y/o variables creadas durante el programa.

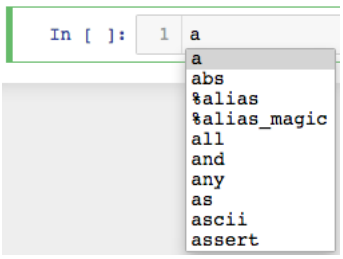

Figura 2: Autocompletado usando la tecla TAB.

2.2. Introspección

La función de **introspección** permite al usuario conocer información acerca de la función o variable en cuestión. Para hacer uso de esto es necesario agregar un signo de interrogación (**?**) al principio o al final del objeto por analizar (Figura 3).

2.3. Atajos de teclado para la terminal

Para mejorar la interacción entre el usuario y la interfaz del cuadernillo de trabajo también se cuenta con algunas combinaciones de teclas que permiten desplazarnos, eliminar, agregar, editar o manejar de diversas formas nuestro código. Existen diversos combinaciones para ello pero en el presente trabajo mostraremos sólo algunas de ellas (Figura 4).

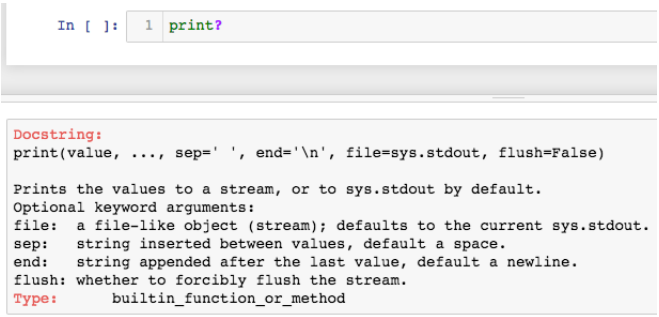

Figura 3: Agregando '?' a un objeto brinda información extra.

Keyboard shortcut	Description
Ctrl-P or up-arrow	Search backward in command history for commands starting with currently entered text
Ctrl-N or down-arrow	Search forward in command history for commands starting with currently entered text
Ctrl-R	Readline-style reverse history search (partial matching)
Ctrl-Shift-V	Paste text from clipboard
Ctrl-C	Interrupt currently executing code
Ctrl-A	Move cursor to beginning of line
Ctrl-E	Move cursor to end of line
Ctrl-K	Delete text from cursor until end of line
Ctrl-U	Discard all text on current line
Ctrl-F	Move cursor forward one character
Ctrl-B	Move cursor back one character
Ctrl-L	Clear screen

Figura 4: Atajos desde el teclado (Wes McKinney).

2.4. Comandos mágicos

Otra característica interesante son los comandos "mágicos", estos son una serie de comandos especiales los cuales facilitan tareas comunes y permiten tomar control del comportamiento del sistema. Entre los comandos tenemos uno de los más destacados para hacer gráficas, el comando **%matplotlib inline**. Este comando permite generar gráficas dentro del cuadernillo de trabajo, además de evitar que la gráfica interfiera en la sesión de la consola.

Al igual que con los atajos para la terminal, tenemos una variedad de opciones para los comandos mágicos y existen muchas otras usando **%magic**. En la figura 5 vemos algunos ejemplos de ello.

3. Básicos del lenguaje Python

A continuación se presentarán algunos aspectos básicos del lenguaje Python con los cuales podremos comenzar a crear nuestros propios códigos y comprender otros que nos parezcan interesantes en Internet. Los aspectos a ver incluyen la semántica del lenguaje, los formatos y el control de flujo de un programa.

Command	Description
`%quickref`	Display the IPython Quick Reference Card
`%magic`	Display detailed documentation for all of the available magic commands
`%debug`	Enter the interactive debugger at the bottom of the last exception traceback
`%hist`	Print command input (and optionally output) history
`%pdb`	Automatically enter debugger after any exception
`%paste`	Execute preformatted Python code from clipboard
`%cpaste`	Open a special prompt for manually pasting Python code to be executed
`%reset`	Delete all variables/names defined in interactive namespace
`%page OBJECT`	Pretty-print the object and display it through a pager
`%run script.py`	Run a Python script inside IPython
`%prun statement`	Execute `statement` with `cProfile` and report the profiler output
`%time statement`	Report the execution time of a single statement
`%timeit statement`	Run a statement multiple times to compute an ensemble average execution time; useful for timing code with very short execution time
`%who, %who_ls, %whos`	Display variables defined in interactive namespace, with varying levels of information/ verbosity
`%xdel variable`	Delete a variable and attempt to clear any references to the object in the IPython internals

Figura 5: Algunos de los comandos mágicos disponibles (Wes McKinney).

3.1. Semántica del lenguaje

Indentación, no llaves .

A diferencia de muchos otros lenguajes de programación, Python hace uso de la indentación, la cual consiste en los espacios o las tabulaciones que le apliquemos al código para separarlo y estructurarlo. De esta manera al realizar nuestro código resultará más sencillo leerlo y que otros también lo hagan.

Todo es un objeto .

Otra característica importante en Python es su *modelo de objeto*, esto consiste en el hecho de que cada número, carácter, estructura de datos, función, clase, modulo y demás es un objeto dentro del lenguaje de Python. Cada objeto tiene un tipo e información interna. Este hecho hace que Python sea un lenguaje bastante flexible.

Comentarios .

Para agregar comentarios en Python es necesario utilizar el gato (#) antes de la línea de código que deseemos omitir pues todo lo que se encuentre a la derecha del # será ignorado. Esta es una manera sencilla y útil de hacer comentarios o inhabilitar comandos sin borrarlos, de esta manera podríamos utilizarlos nuevamente después.

Llamando funciones y objetos .

Para llamar a las funciones basta con agregar un par de paréntesis al final y agregar un cero o más argumentos, también es posible asignar el resultado de alguna función a una variable y, dado que son objetos, es posible aplicarle funciones internas que permiten al usuario conocer información específica acerca de ellas (Véase la figura 6).

Traspaso de variables y argumentos .

Cuando le asignamos una variable (o nombre) a un elemento, lo que estamos haciendo es crear una referencia a él mismo. Tomemos como ejemplo una variable **A** con los valores 1, 2 y 3.

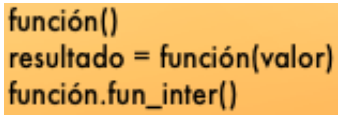

Figura 6: Algunas opciones con una función.

$$\mathbf{A = [1, 2, 3]}$$

Posteriormente igualamos **A** con otra variable **B**.

$$\mathbf{B = A}$$

Dado que ambas variables son una referencia a la lista de valores, si a **A** le agregamos el valor de 4, **B** también lo tendrá.

$$\mathbf{a.append(4)}$$
$$\mathbf{b = [1, 2, 3, 4]}$$

Atributos y métodos .

Los objetos en Python tiene atributos (otros objetos de Python "dentro"de los objetos) y métodos (funciones asociadas a los objetos que pueden acceder a la información interna del objeto), para acceder a estas funciones simplemente debemos de escribir el nombre de la variable, seguido de un punto '.' y presionamos la tecla **TAB** para ver sus opciones (Véase la figura 7).

```
In [1]: a = 'foo'

In [2]: a.<Press Tab>
a.capitalize  a.format      a.isupper     a.rindex      a.strip
a.center      a.index       a.join        a.rjust       a.swapcase
a.count       a.isalnum     a.ljust       a.rpartition  a.title
a.decode      a.isalpha     a.lower       a.rsplit      a.translate
a.encode      a.isdigit     a.lstrip      a.rstrip      a.upper
a.endswith    a.islower     a.partition   a.split       a.zfill
a.expandtabs  a.isspace     a.replace     a.splitlines
a.find        a.istitle     a.rfind       a.startswith
```

Figura 7: Atributos para una cadena de caracteres.

Importando .

En Python un modulo puede ser cualquier archivo que cuente con la extensión *.py* y contenga un código de Python. Para acceder a todo su contenido es necesario importarlos a nuestro código.

import nombre-modulo as n-m (abreviación del modulo)

Y posteriormente podemos llamar alguna función, variable o lo que deseemos del modulo usando la siguiente sintaxis.

n-m . nombre-variable-o-funcion() (nombres unidos por el .)

Operadores binarios y comparaciones .

La mayoría de los operadores binarios y de comparación son como lo esperaría cualquier persona. Existen una lista completa con los operadores que podemos trabajar (Véase la figura 8).

Operation	Description
`a + b`	Add a and b
`a - b`	Subtract b from a
`a * b`	Multiply a by b
`a / b`	Divide a by b
`a // b`	Floor-divide a by b, dropping any fractional remainder
`a ** b`	Raise a to the b power
`a & b`	`True` if both a and b are `True`; for integers, take the bitwise AND
`a | b`	`True` if either a or b is `True`; for integers, take the bitwise OR
`a ^ b`	For booleans, `True` if a or b is `True`, but not both; for integers, take the bitwise EXCLUSIVE-OR
`a == b`	`True` if a equals b
`a != b`	`True` if a is not equal to b
`a <= b, a < b`	`True` if a is less than (less than or equal) to b
`a > b, a >= b`	`True` if a is greater than (greater than or equal) to b
`a is b`	`True` if a and b reference the same Python object
`a is not b`	`True` if a and b reference different Python objects

Figura 8: Operadores disponibles en Python (Wes McKinney).

3.2. Tipo escalar

Dentro del lenguaje Python existen ciertos tipos en para manejar la información numérica, cadenas de caracteres, valores booleanos y fechas y horas. Todos estos valores usualmente son llamados tipos escalares (Vease la figura 9).

Type	Description
`None`	The Python "null" value (only one instance of the `None` object exists)
`str`	String type; holds Unicode (UTF-8 encoded) strings
`bytes`	Raw ASCII bytes (or Unicode encoded as bytes)
`float`	Double-precision (64-bit) floating-point number (note there is no separate `double` type)
`bool`	A `True` or `False` value
`int`	Arbitrary precision signed integer

Figura 9: Tipos existentes en Python.

Tipos numéricos .

Los tipos primarios para números en Python son **int** y **float**. Los números del tipo **int** pueden almacenar números arbitrariamente grandes. Por otro lado, los números de punto flotante son el tipo **float** y pueden expresarse con notación científica.

Finalmente, cuando hagamos divisiones existen dos tipos de resultados posibles; al dividir dos números enteros, el resultado puede no ser entero, es entonces cuando se generar un número del tipo **float**.

Entrada: 3/2 (Números **int**)

Salida: 1.5 (Número **float**)

Sin embargo, si deseamos que el resultado se mantenga siendo un número del tipo **int** debemos agregar una diagonal (/) más a la operación.

Entrada: 3//2 (Números **int**)

Salida: 1 (Número **int**)

Cadenas de caracteres .

Para crear una variable que contenga caracteres debemos de usar las comillas sencillas (') o dobles (") al principio y fin de los caracteres que deseamos almacenar; mientras que para almacenar caracteres en distintos renglones hacemos uso de tres comillas sencillas o dobles de igual modo que para un solo renglón.
Las cadenas de caracteres no son modificables pues una vez creada la variable no será posible alterar su contenido pero sí es posible convertir diversos objetos de Python en una cadena de caracteres usando la función de **str**.

Entrada 1: A = 5.6 (A es del tipo **int**)
Entrada 2: S = str(A) (S recibe A como una cadena de caracteres)
Entrada 3: print(S)
Salida 3: 5.6 (El valor es ahora una cadena de caracteres)

Booleanos .

Los únicos dos valores son **True** y **False**. Las comparaciones y otras expresiones condicionales al ser evaluadas arrojan uno de estos dos valores. Estos valores están relacionados con las funciones lógicas **and** y **or**.

Fecha y hora .

El modulo **datetime** de Python incluye tres tipos de objetos: (1) **datetime**, el tipo más usado de los tres, (2) **date**, utilizado para almacenar alguna fecha y (3) **time**, que almacena únicamente datos con formato de hora. Usualmente es más usado el tipo **datetime** ya que es posible usar un *método* (función interna del objeto) para tomar el valor de la fecha u hora, siempre y cuando cumpla con un formato reconocible (Véase la figura 10).

3.3. Control de flujo

Python tiene sus palabras clave para referirse a ciertas condiciones lógicas, bucles y otros conceptos para controlar el flujo del programa. A continuación veremos más acerca de esto.

Type	Description
%Y	Four-digit year
%y	Two-digit year
%m	Two-digit month [01, 12]
%d	Two-digit day [01, 31]
%H	Hour (24-hour clock) [00, 23]
%I	Hour (12-hour clock) [01, 12]
%M	Two-digit minute [00, 59]
%S	Second [00, 61] (seconds 60, 61 account for leap seconds)
%w	Weekday as integer [0 (Sunday), 6]
%U	Week number of the year [00, 53]; Sunday is considered the first day of the week, and days before the first Sunday of the year are "week 0"
%W	Week number of the year [00, 53]; Monday is considered the first day of the week, and days before the first Monday of the year are "week 0"
%z	UTC time zone offset as +HHMM or -HHMM; empty if time zone naive
%F	Shortcut for %Y-%m-%d (e.g., 2012-4-18)
%D	Shortcut for %m/%d/%y (e.g., 04/18/12)

Figura 10: Formatos disponibles para el tipo **datetime** (Wes McKinney).

If, else if y else .

La declaración de un **if** es una de las más comunes y al igual que en otros lenguajes lo que hace este comando es evaluar si la condición dada es verdadera y proceder a realizar la tarea que sea asignada para este caso o de igual manera evaluar otras situaciones hasta que la condición se cumpla, o no, para proceder. Se pueden agregar tantas condiciones/situaciones como se deseen, solamente se debe agregar el comando **elif** para cada una de esas condiciones extras.

if *condicion-1* **:**
accion-1
elif *condicion-2* **:**
accion-2
else *condicion-3* **:**
accion-3

For loops .

Este controlador se utiliza para llevar a cabo un proceso cierta cantidad de veces, modificando el valor de una variable dada hasta cumplir con la secuencia a seguir, haciendo que el proceso dado sea automatizado. La sintaxis es la siguiente:

for *variable* **in** *secuencia* **:**
accion-a-realizar

También es posible agregar un comando **break** dentro de este controlador y sirve para salir del proceso cuando se cumpla cierta condición que sea establecida en el código.

While loops .

El presente comando es parecido al visto anteriormente pues realizar un proceso iterativo, salvo que en esta ocasión la cantidad de veces que se ejecute la acción indicada depende de una condición lógica, y mientras esta dé como resultado verdadero (**True**), las iteraciones seguirán hasta que el resultado sea lo contrario.

while *condicion-logica* **:**
accion

Pass .

Este comando (**pass**) es un auxiliar más (al igual que el **break**) pero con la diferencia que al utilizarlo el proceso no es interrumpido sino que más bien simplemente se ignora está opción y se procede de forma habitual en el código.

Range .

El comando es utilizado para crear un iterador que sigue una secuencia de números enteros igualmente espaciados. Para crearlo es simplemente necesario insertar el número hasta el cual se generará, siendo de este modo el predeterminado (comenzando desde 0 y aumentando en 1 el valor del iterador), o bien, es posible indicar en qué número comienza, en cuál termina y el incremento que se recibirá).

range(*valor-inicial***,** *valor-final***,** *aumento***)**

Referencias

[1] Wes McKinney, *Python for Data Analysis. Data Wrangling with Pandas, NumPy, and IPhyton*, second edition, O'Reilly, (2017).