Introducción a Python

Julián Jiménez Cárdenas

juliano.jimenezc@konradlorenz.edu.co

Semillero de Programación en Python
Fundación Universitaria Konrad Lorenz, Bogotá.

25 de febrero de 2017

I. Reseña

Según varios censos realizados, Python es uno de los lenguajes más utilizados en diversas tareas, que van desde los cálculos numéricos hasta la recolección de información. Uno de los motivos por los que ésta es una herramienta predilecta para cualquier tipo de tarea es su versatilidad y su simpleza, sin perder robustez (alto nivel). A continuación aprenderá las cosas básicas del lenguaje.

II. Introducción a los Tipos de Datos

En todo lenguaje de programación orientado a objetos, tenemos lo que se conoce como objetos primitivos, aquellos que tienen una memoria **predeterminada** en la memoria. En Python son los siguientes:

i. Enteros

Para convertir cualquier tipo de dato en un entero (mientras sea coherente, si no, seguramente saldrá un error), se usa la palabra reservada `int(object)`, donde *object* es el objeto que se desea transformar en entero. Para saber qué tipo de objeto es un elemento, se usa la palabra reservada `type(object)`.

```
x=2
```

En la línea anterior se le asigna a la variable x el valor de 2. A continuación veremos algunas operaciones comunes que se pueden realizar entre enteros.

1. Suma (Resta).

   ```
   >>> x+3
   5
   ```

2. Multiplicación.

   ```
   >>> x*5
   10
   ```

3. División.

   ```
   >>> x/3
   0.666666666666
   ```

   Hay que tener cuidado cuando se realiza esta operación en Python, porque, por ejemplo, en Python2.X, la división es clausurativa, es decir, la división de números enteros resulta en un entero. Para evitar ésto, se recomienda multiplicar por $1.*x$, para hacer un *casting explícito*[1], y así el resultado será un número decimal.

4. Potenciación.

   ```
   >>> x**3
   8
   ```

5. Módulo.

   ```
   >>> x % 3
   1
   ```

   Recordemos que la operación módulo retorna el residuo resultante al dividir x entre 3.

[1] Convertir deliberadamente un tipo de dato a otro; en este caso, un entero en un flotante. También se puede usar `float(x)`.

II. Flotantes

Los flotantes comparten las mismas operaciones que los enteros, a excepción del módulo. El *casting explícito*, como se vio en la subsección anterior, es `float(object)`.

III. Booleanos

Este objeto puede tomar dos valores, `True` o `False`. Más tarde éste será útil para construir condicionales. Por ahora hay que enfocarse en las operaciones fundamentales que se pueden hacer con booleanos. Su *casting explícito* es `bool(object)`

```
>>> x=True
>>> y=False
>>> x and y
False
>>> x or y
True
>>> x==y
False
>>> x!=y
True
>>> not(x)
False
```

Por orden de aparición, las operaciones realizadas son: y, o, si y sólo si, ó, negación.

IV. Cuerdas/Cadenas/Strings

Este tipo de dato no tiene tamaño en memoria predeterminado, pero como es uno de los más sencillos, se explicará en esta sección. Las cadenas son un tipo de dato muy vérsatil, pueden almacenar un conjunto de valores alfanuméricos consecutivamente. Además, son muy flexibles en lo que refiere al *casting explícito* (es `str(object)`). Las cadenas tienen dos operaciones fundamentales.

1. Concatenación

   ```
   >>> x="Hola␣"
   >>> y="Mundo"
   >>> x+y
   "Hola␣Mundo"
   ```

   Como se puede apreciar, las cadenas van delimitadas por comillas sencillas (') o comillas dobles ("). Se recomienda usar comillas dobles porque en el inglés se tiende a usar las comillas sencillas gramaticalmente.

2. Producto por entero.

   ```
   >>> x*3
   "Hola␣Hola␣Hola␣"
   ```

III. Ciclos

I. For

Los ciclos *for* se suelen usar cuando se necesita repetir cierta instrucción un número determinado de veces. En Python es importante **identar (tabular)** las instrucciones que va a realizar un ciclo o un condicional. A continuación se verán algunos ejemplos.

```
>>> for x in range(1,10):
        print(x)
```

`range(a,b,c)` es una función predeterminada de Python. Lo que hace es generar una lista de elementos enteros, desde `a` hasta `b-1`, en pasos de `c`. Se recomienda usar la instrucción para que se familiarice con la idea. `print(object)` imprime en pantalla el valor de *object*. **Nota Importante:** Los ciclos y condicionales deben terminar siempre con **:**, para indicarle al compilador que lo que sigue irá identado y es lo que le corresponde al entorno ejecutar.

```
>>> for x in "a␣b␣c␣d".split("␣"):
        print(x)
```

Ejercicio: Verificar qué hace el bloque de código anterior, así como la utilidad de la función `split`, propia de las cadenas.

II. While

Este ciclo indica que se realice determinada acción hasta que algo ocurra. Hay que tener cuidado con ellos, pues si no se usan con precaución, pueden generar ciclos infinitos, que

podrían llegar a consumir la memoria RAM del computador. A continuación un ejemplo.

```
>>> x=0
>>> while (x<20):
        print(x)
        x=x+1
```

Ejercicio: ¿Qué hace este programa?
Ejercicio: Demostrar que todo ciclo *while* puede escribirse como un ciclo *for* y viceversa.
Ejercicio: Investigar cómo usar `break` para detener un ciclo.

IV. Condicionales

1. If

Este condicional básicamente evalúa si se cumple una proposición, y si sí, procede a ejecutar el código identado respecto a sí. Véase los siguientes ejemplos y note la diferencia.

```
>>> x=0
>>> if (x<0):
        print(x+"<0")

>>> x=0
>>> if (x>0):
        print(x+">0")
```

¿Qué imprimirá en programa?
Ejercicio: Investigar acerca del uso del `else` y `elif`.

V. Try/Except

Esta instrucción se usa cuando uno quiere verificar si se puede realizar una acción, y si no, hacer un procedimiento alterno. Este concepto se conoce como manejo de errores, útil para evitar la finalización de la ejecución de un programa.

```
>>>x=input("Ingrese numero >")
>>>try:
        int(x)

>>>except:
        print(x+" not number")
```

Ejercicio: ¿Qué hace este programa?
Ejercicio: Hacer un programa que reciba una cadena de caracteres y verifique si en ella hay números.

VI. Tarea

Si usted es capaz de hacer todos estos ejercicios, significa que entendió a cabalidad los temas expuestos en la primera sección.

1. Hacer un programa que calcule y muestre en pantalla los números primos que hay hasta 1000.
2. Hacer un programa que encuentre los primeros 100 números de la sucesión de Fibonacci.
3. Hacer un programa que determine los primeros 1000 números primos.
4. Si siente que está siendo muy fácil, le propongo hacer los siguientes ejercicios `https://github.com/julian20250/SemilleroPython/blob/master/EjerciciosInteresantes/EjerciciosIntroduccion.pdf`
   Si eres matemático, también debes:
5. Verificar la demostración del problema de Basilea.
6. Comprobar que la serie armónica es divergente.