Introducción a Python

Carlos Malanche

6 de febrero de 2018

1. El lenguaje de programación `Python`

Durante el curso vamos a hacer uso de herramientas un poco más sofisticadas que *excel*, o *matlab*. Todo girará alrededor de Python, un lenguaje de programación cuyo código fuente está disponible y se puede utilizar de manera gratuita. Hay varias razones por las que usaremos `Python`:

- Es muy fácil de entender y de empezar a usar, pues realiza ciertas tareas de manera automática (por ejemplo, asignación de tipo de variables).
- Es interactivo: al ser un lenguaje interpretado, es posible ejecutar instrucciones aisladas paraq observar el resultado inmediatamente.
- Es *aceptablemente* rápido, hay trucos (como `cpython`) que permiten obtener la velocidad de un lenguaje compilado. Sí, C++ podrá ser más rápido, pero a veces importa más que la velocidad de *implementación* de un código sea la parte más eficiente, lo cual `Python` hace muy bien.
- Hay muchos, muchos, muchos lugares en los cuales obtener ayuda en internet, ya que es un lenguaje muy popular (vaya, el sitio de la NASA, Bitbucket e Instagram están montados con Python).
- Es estable: Hay lenguajes recientes que son más eficientes al momento de programar y en su ejecución pero aún se encuentran en desarrollo. `Julia` es un buen ejemplo, aún no sale la versión 1.0 y por ello códigos que funcionan hoy pueden dejar de funcionar en 1 semana.
- El curso **no es sobre Python**, `Python` es sólo una herramienta que es conveniente en este caso.

Vamos directo a su instalación, y los códigos más pequeños y lindos para familiarizarnos con `Python`.

2. Instalación de Python

En general, todas las instrucciones para instalar `Python` se encuentran bien documentadas en su sitio web. Vamos a utilizar la última versión de `Python` (3.6.4 al momento de crear este documento). Para Windows no es necesario utilizar `Anaconda`, ya que `scikit-learn`, `numpy`, `pandas` y `scipy` se pueden instalar fácilmente con la herramienta `pip` (administrador de paquetes) de `Python`. Por el momento sólo utilizaremos `Python` con `numpy`.

Si necesitan ayuda instalando `numpy`, díganme y los asistiré.

3. Los primeros pasos

Como ya es tradición en cualquier curso donde se use un lenguaje de programación por (probablemente) primera vez, vamos a escribir el programa *hello world* para `Python`.

Python es un lenguaje de programación imperativo, lo que significa que ejecuta una instrucción tras otra. Estas instrucciones pueden ser declaraciones de variables, condiciones, asignación de valores llamadas a funciones u otras cosas.

En mi caso, voy a utilizar Visual Studio Code como editor de archivos (algo así como el bloc de notas pero en su versión con *adamantium* inyectado), pero ustedes pueden utilizar cualquier herramienta con la que se sientan cómodos. Para practicar, les recomiendo hagan un folder del nombre que gusten, por ejemplo *aml-inicio* donde van a guardar los *scripts* de `Python` de esta clase. Desde la terminal, puedo editar un archivo de nombre `helloworld.py` escribiendo lo siguiente:

```
C:\Users\Carlos\Codes\adml-inicio
λ code helloworld.py
```

A continuación, escribio la siguiente línea dentro del archivo *helloworld.py*

Listing 1: Código fuente de helloworld.py

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

Para ejecutar este código, desde la terminal ejecuto `python` dándole como argumento el nombre del código fuente:

```
C:\Users\Carlos\Codes\adml-inicio
λ python helloworld.py
Hello world
```

La función print escribe a la terminal la cadena de caracteres que le es proporcionada. Python reconoce una cadena de caracteres por estar contenida entre comillas ".

`Python` también puede ejecutarse en su versión interactiva, basta con escribir `python` en la terminal, y *Python* irá ejecutando las instrucciones que le hagamos llegar. En modo interactivo, existe un espacio de trabajo (*workspace*) en donde se van guardando variables, definiciones de funciones, etc. de cada sesión.

```
C:\Users\Carlos\Codes\adml-inicio
λ python
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> x = 1.56
>>> x * 10
15.600000000000001
>>>
```

Hmmmm... eso no es $1{,}56 \times 10$. Bueno, *ahí les queda de tarea* investigar por qué ocurrió eso.

3.1. Declaración de variables

Tal como se puede ver en la última captura de pantalla, en `Python` se pueden declarar variables. Para declarar una variable en `Python` basta darle un nombre y darle un valor (ya sea un entero, un flotante, una cadena de caracteres, etc.)

Listing 2: Código fuente de variabledecl.py

```
# Texto de salida
texto = "El valor es: "
# Declara variable x con valor 10
x = 10
# Declara variable y con dos veces el valor de x
y = x * 2
# Muestra el resultado en la terminal
print(texto + str(y))
```

Para `Python`, las líneas de código que comienzan con # son ignoradas, con lo que podemos usar dicho signo para añadir comentaris al código.

El resultado de ejecutar este *Script* es:

```
C:\Users\Carlos\Codes\adml-inicio
λ python variabledecl.py
20
```

Para variables de tipo entero o flotante, se pueden realizar las operaciones de multiplicación (*), división (/), suma (+), resta (-), módulo o residuo (%), división con redondeo (//) y exponenciación (**).

Además hay operadores de comparación que devuelven como resultado verdadero (*True*) o falso (*Falso*) como la equidad (==), no equidad (!=), *mayor que* y *menor que*, *mayor o igual que* y *menor o igual que* (<, >, <=, >=).

Jueguen un rato con estas operaciones para que se familiaricen.

3.1.1. Cadenas de caracteres

Una cadena de caracteres en `Python` se declara entre comillas. Se puede acceder a los elementos de la cadena de caracteres por medio de los paréntesis cuadrados. El primer elemento lleva el índice 0.

Por defecto, podemos acceder a elementos con índices negativos en `Python`, y el comportamiento de `cadena[-i]` es igual al de `cadena[i]`

Los dos puntos nos permiten acceder a un rango de caracteres, puede ser desde un valor a hasta un valor $b-1$ (`cadena[a:b]`), o desde un caracter hasta el final de la cadena, por ejemplo:

```
C:\Users\Carlos
λ python
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> x = "no voy a pasar el curso!"
>>> x[3:]
'voy a pasar el curso!'
>>>
```

También se puede acceder desde el principio de la cadena hasta el caracter $b-1$ (`cadena[:b]`).

Hay un tercer argumento que sirve para dar *brincos* entre caracteres. La instrucción `cadena[a:b:c]` generará la cadena de caracteres empezando en a, seguida de los caracteres en los índices $a + i * c$ (con $i > 0$ entero) tal que no se exceda ni iguale b (Los valores por defecto de a, b y c son 0, la longitud de la cadena y 1 respectivamente). Por ejemplo, si `cadena = "Hoy es Martes"`, entonces `cadena[::2]` resultará en `"Hye ats"`.

3.1.2. Listas

Un tipo de variable que usaremos mucho son las *listas*. Son series de elementos simplemente (no tienen que ser del mismo tipo). La declaración de una lista es por medio de los paréntesis cuadrados, y con los mismos se puede acceder a los elementos de la lista, enumerados desde 0.

```
C:\Users\Carlos\Codes\adml-inicio
λ python
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> miLista = [1, 5, 8, 10]
>>> miLista
[1, 5, 8, 10]
>>> miLista[1]
5
>>>
```

3.2. Instrucciones básicas

Al programar nos auxiliamos de muchas cosas, desde funciones complejas hasta simples instrucciones que permiten al código *tomar* decisiones. Las siguientes tres instrucciones ejecutan un bloque de código que se separa con una indentación dada una condición (en `Python` la estructura del código es importante, la indentación denota bloques de código). Además las instrucciones siguientes se anuncian con dos puntos al final.

3.2.1. if else

A veces queremos que el código tenga cierto comportamiento, dependiendo de algún parámetro como puede ser la información que un usuario nos da. Para ello existe el par `if else`. A `if` le sigue una condición *c*. Al momento de la ejecución, si la condición se satisface se ejcuta el código indentado después de la instrucción `if`. Adicionalmente, se puede agregar la instrucción `else` que es lo mismo que escribir `if !c` (*si no-c*), que le dice al python qué hacer si la condición no fue satisfecha.

Listing 3: Código fuente de par.py

```
# Pide variable al usuario y convierte en entero
x = int(input("Ingrese un valor entero: "))
# Verifica si el residuo al dividir por 2 es cero.
if (x %2) == 0:
    print("El valor es par!")
else:
    print("El valor no es par!")
```

3.2.2. for

Existe también la instrucción `for`, que permite la ejecución repetida de una sección de código.

Listing 4: Código fuente de forloop.py

```
# Repite una instruccion 5 veces
for x in range(0,5):
    # x toma valores de 0 a 4
    print(x)
```

La función `range(start, stop)` genera una lista de valores desde `start` hasta `stop` sin incluír este último.

3.2.3. while

La instrucción `while` ejecuta repetidamente un bloque de código hasta que una condición queda satisfecha.

Listing 5: Código fuente de whileblock.py

```
import sys
# Lee x de la terminal
x = int(input("Ingrese un natural: "))
#Implementacion de una verificacion no optima de la conjetura de Collatz
while x != 1:
    if x %2 == 0:
        x = x / 2
    else:
        x = x * 3 + 1
    sys.stdout.write(str(int(x)) + ", "x)

print("La cadena regresa al 1.")
```

El código de ejemplo verifica la conjetura de Collatz para el número que el usuario ingresa.

El código hace uso de una nueva instrucción, `import`, que permite el uso de funciones dentro del paquete `sys`. Un paquete es justo eso, un conjunto de herramientas disponibles al programador, en este caso utilizamos la habilidad de escribir a la terminal sin dejar saltos de línea, como lo hace `print`.

Las 3 instrucciones pasadas dependen de condiciones. Para pedir el cumplimiento de más de una condición (o de alguna condición entre varias) se utilizan las palabras `and` y `or`.

Estas 3 instrucciones brindan por sí mismas bastantes posibilidades, pero vamos a necesitar un poco más para poder hacer análisis de datos.

4. Definición de funciones

Muchas veces vamos a escribir un pedazo de código que nos gustaría ejecutar en diversos puntos de un script. Para ello se pueden definir funciones en *Python* con ayuda de la palabra `def`.

Listing 6: Código fuente de promedio.py

```
# Define una funcion llamada promedio que recibe un argumento
def promedio(valores):
    # Saca el promedio de una lista
    return sum(valores)/len(valores)

misValores = [1,2,3,4,5,6,7,8,9,10]

print(promedio(misValores))
```

5. Ejercicios

5.1. suma.py: Suma de los primeros n números

Haga un programa que reciba una variable n y calcule la suma de los números desde 1 hasta n.

5.2. palindromo.py: Palíndromos

Haga un programa que pida al usuario una cadena de caracteres, y el programa diga si es o no un palíndromo.

5.3. ppalindromo.py: Producto palíndromo

Encuentre el número más grande i inferior a $n = 100000$ que al ser multiplicado con el número resultante de poner las cifras en el orden contrario genera un palíndromo. *Ejemplo: 21* pues $21 \times 12 = 252$.

Recuerden: el internet ya llegó, y páginas como stackoverflow son la mejor ayuda que pueden conseguir.