Introducción a Python

Paula Muñoz Lago
María Esther Ruiz-Capillas Muñoz

Facultad de Informática, UCM

December 19, 2018

Programar: Dar una serie de instrucciones a nuestro ordenador para que las cumpla con éxito y así obtener el resultado deseado.
Los pasos que hay que seguir son los siguientes:

1. Escribir el código
2. Compilarlo
3. Ejecutarlo

Escribir el código

Dependiendo del sistema operativo que uses, y la cantidad de código que vayas a escribir, te recomiendamos algunos programas...

- **Notepad++ o SublimeText:** Son editores de texto que no disponen de compilador integrado, por lo que todo lo que programemos en ellos habrá que compilarlo y ejecutarlo desde la terminal de comandos. El primero se encuentra en Windows y el segundo en Linux.
- **Visual Studio Code:** También es un editor de texto, en el cual podremos escribir nuestro código e instalar a continuación extensiones (plug-ins) para poder ejecutar código. Se encuentra en ambos sistemas operativos.

Qué significa "Compilar"

El proceso de compilación se encarga de comprobar que no hay fallos en el código escrito por el/la programador(a), y convertirlo a lenguaje de máquina (no legible por los programadores).

Si la compilación da algún error, el proceso terminará y nos indicará la linea de código donde está el error.

Para comprobar que la compilación ha terminado con éxito bastará con ver que se ha generado correctamente el archivo .exe (ejecutable)

Ejemplo de compilación y ejecución

1. Creamos Test.py y escribimos:

$$print("ASAP\ mola!") \tag{1}$$

2. Si trabajamos en Visual Studio Code y hemos instalado la extensión Code Runner bastará con darle al botón de "play" que se encuentra en la esquina superior derecha, si trabajamos desde un editor sin compilador, o no tenemos dicha extensión, bastará con que escribamos en la terminal el siguiente comando:

$$python\ -u\ Test.py \tag{2}$$

   De esta forma, veremos que en la pantalla aparece "ASAP mola!"

Python

Razones por las que aprender Python

Lenguaje "sencillo", ideal para iniciarse en la programación.

Ligero, de compilación rápida.

Además, es uno de los lenguajes pricipales con los que se trabaja en el campo del **procesamiento del lenguaje natural**[1]. Un proyecto interesante de PLN es PICTAR[2] para traducir frases a pictogramas para mejorar la comprensión del lenguaje de personas Autistas.

---

[1] https://es.wikipedia.org/wiki/Procesamiento_de_lenguajes_naturales

[2] http://hypatia.fdi.ucm.es/pictar/

Interacción con el usuario: print

Lo más importante a la hora de programar es la comunicación con el usuario.
Para poder imprimir mensajes usaremos **print()**. El texto que se quiera imprimir tendrá que estar escrito entrecomillado entre de los paréntesis.

Ejemplo: print

```
print("Hola ASAP")          "Hola ASAP"
print()                     "\n"
```

Interacción con el usuario: input

Para poder recibir información desde la terminal usaremos **input()**. Podremos insertar texto dentro de los paréntesis (se mostrará como un print y recogerá la información de la terminal) o no (únicamente recogerá la información de la terminal). Debido a que no queremos perder la información que recojamos, la guardaremos en una variable.

Ejemplo: input

```
asociación = input("¿Cuál es el nombre de tu asociación?: ")
asociación = input()
```

Ambos ejemplos son válidos pero siempre se debe orientar al usuario sobre qué información introducir.

Tipos de Datos

En programación, siempre tenemos que guardar datos en algún punto de nuestro programa. Para ello, debemos conocer bien los diferentes tipos de datos con los que nos podemos encontrar. En esta introducción veremos los tipos más comunes:

Numéricos **Booleanos** **Strings** **Listas**

Tipos de Datos: Numéricos

Datos numéricos

int floating point

Ejemplo: Prueba con datos de tipo int

```
x = 2
y = 3
print(x + y)
"5"
```

Ejemplo: Prueba con datos de tipo float

```
x = 2.56
print(type(x))
"<class 'float'>"
```

Tipos de datos: Booleanos

Este tipos de datos únicamente puede tomar dos valores: verdadero o falso. En general, todas las variables pueden considerarse como booleanas. Aquellos elementos nulos o vacíos adoptarán el valor de falso mientras que todos los demás serán verdaderos. Además, el número 1 actuará como verdadero y el 0 como falso. Este tipo de datos se utilizará más adelante como condición de control.

Ejemplo:

```
ASAPExiste = true
```

Tipos de Datos: Strings

Un string es una cadena de caracteres, sobre los cuales podemos operar de la siguiente forma:

Ejemplo: Operaciones sobre una cadena

```
frase = "ASAP mola"
print(frase)                    "ASAP mola"
print(frase[3])                 "P"
```

Entenderemos las frases como una lista de caracteres, por eso en el segundo ejemplo, intentamos acceder a la posición 3 de la cadena "ASAP mola". **En programación las listas empiezan en la posición 0**.

Tipos de datos: Listas

Como hemos dicho antes, una lista es una cadena de cualquier tipo de datos. Pueden ser de un mismo tipo (números, strings, etc) o de varios.

Ejemplo: Trabajo con listas

```
info_ASAP = ["ASAP", "UCM", "Filología", 2018]

print("Información sobre la asociación", info_ASAP[0])
print("Pertenece a la facultad: ", info_ASAP[2], info_ASAP[1])
print("Año de creacción: ", info_ASAP[3])
print("Numero de datos que disponemos de ",
info_ASAP[0], ": ", len(info_ASAP))
```

Tipos de datos: Listas

Una vez creada una lista se le podrán añadir nuevos elementos con **.append()**
Si queremos mostrar los elementos de la lista podremos usar **print(nombre_de_la_lista)**. En cambio, si lo que queremos es mostrar el contenido de la lista como un único string separado por espacios podremos usar **print(" ".join(nombre_de_la_lista))**

Ejemplo: Addición e impresión

```
info_ASAP = ["ASAP", "UCM", "Filología", 2018]

print(info_ASAP)                    "[ASAP, UCM, Filología, 2018]"
info_ASAP.append("Mola")
print(" ".join(infor_ASAP))         "ASAP UCM Filología 2018
Mola"
```

Operaciones lógicas

Las operaciones lógicas son aquellas que nos permiten relacionar ciertos datos de una manera determinada y nos proporciona un resultado. Más adelante veremos que esto tendrá utilidad en las condiciones.
Tanto estos datos como el resultado únicamente podrán ser de tipo booleano.
Las operaciones lógicas más usadas son **and** y **or**.

AND	**True**	**False**
True	True	False
False	False	False

OR	**True**	**False**
True	True	True
False	True	False

Comparaciones

Las comparaciones también proporcionan un resultado booleano aunque, a diferencia de las operaciones lógicas, no es necesario que los datos tratados sean de tipo booleano.
También se les encontrará la utilidad más adelante, para las condiciones.
Para comprobar si dos datos son iguales se utilizará == y para comprobar si son diferentes !=.

Ejemplo: Comparaciones

```
x = 5
y = 3
print(y == x)
"False"
```

Estructuras

Estructuras de control: If

Todo programa informático necesita tomar decisiones por lo que, gracias a las condiciones booleanas y a los bloques **if**, dotaremos a nuestro programa de la posibilidad de ejecutar un fragmento de código en caso de que una condición se cumpla.

Ejemplo: Uso de if

```
socios = 300
if socios == 300:
        print("ASAP tiene 300 socios, impresionante")
```

Estructuras de control: else, elif

¿Y si el fragmento de código que queremos ejecutar depende del caso en el que nos encontremos? Podremos usar las cláusulas **elif** (si se van a contemplar más casos después) o **else** (si no se cumple ninguno de los casos anteriores).

Ejemplo: Uso de if

```
socios = 300
if socios > 500:
        print("ASAP tiene más de 500 socios, impresionante")
elif socios > 250 and socios <= 500:
        print("ASAP tiene entre 251 y 500 socios!")
else:
        print("ASAP tiene ", socios, " socios")
```

Bucles: While

Para ejecutar un código varias veces, y hacerlo de una forma limpia y ordenada, siempre vamos a tener que hacer uso de algún tipo de bucle.

Ejemplo: Erroneo

```
print("ASAP mola")
print("ASAP mola")
```

Ejemplo: Uso de bucle while

```
veces_que_lo_he_dicho = 0
veces_que_quiero_decirlo = 2
while veces_que_lo_he_dicho < veces_que_quiero_decirlo:
        print("ASAP mola")
        veces_que_lo_he_dicho += 1
```

Bucles: For

Otro tipo de bucles comúnmente utilizados en programación son los bucles **for**. En este caso encontramos diferentes sintaxis que pueden cumplir la misma funcionalidad.

Ejemplo: Uso de bucle for

```
veces_que_quiero_decirlo = 2
for veces_que_lo_he_dicho in range(veces_que_quiero_decirlo):
        print("ASAP mola")
```

Bucles: For

Ejemplo: Uso de bucle for con listas

```
frutas = ["manzana", "platano", "naranja"]
for fruta in frutas:
        if fruta is "platano":
                print("¡Gracias!")
        else:
                print("Ug no quiero ", fruta, " gracias!")
```

También se puede usar el número de elementos de una lista como valor máximo para el bucle con **len(nombre_de_la_lista)**

Ejemplo: Uso de len()

```
for i in range(len(frutas)):
        frutas[i] = pera
```

Ahorcado

Enunciado

En primer lugar introduciremos una palabra, la cual nuestro contrincante tendrá que adivinar. El programa leerá dicha palabra y la almacenará en una variable. A continuación pedirá al usuario letras mientras va comprobando si existen en nuestra palabra o no, con un máximo de 6 fallos. Con cada letra que acierte el programa irá mostrando el estado del juego, las letras sin adivinar como "_" y las letras adivinadas en su posición.

Preguntas

Paula Muñoz Lago: pmunoz06@ucm.es
Esther Ruiz-Capillas Muñoz: mruizcap@ucm.es

¡Muchas gracias por la atención! Esperamos que esta charla despierte vuestro interés por Python y que sigáis trabajando en ello.