Universidad Nacional de Rosario (UNR)
Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Curso introductorio a la Programación

Aprendiendo a programar desde cero utilizando Python

(Módulo I)

Autor: Lic. Leonardo Bartocci

Desarrollado con la expectativa de ayudar a dar el primer paso a toda persona que desee introducirse en el mágico mundo de la programación

Septiembre del 2019

Índice general

Capítulo 1

Introducción

Este curso está dirigido a quienes desean dar el primer paso en el mundo de la programación.

El lenguaje utilizado como herramienta para el aprendizaje será *Python*. Pero se intentará utilizar sentencias de de usos comunes a la mayoría de los lenguajes de uso masivo de forma que quién realice este curso luego pueda sentirse familiarizado con las sentencias usuales de cualquier otro lenguaje que necesite utilizar en el futuro.

Es importante que el estudiante entienda desde el primer momento que este curso tiene como objetivo brindarle una introducción informal a la programación en general y que *Python* es sólo el lenguaje elegido como herramienta, pero la herramienta podía haber sido cualquier otro lenguaje de programación. Resumiendo, este curso intenta enseñar a programar independientemente del lenguaje de programación a utilizar.

En este documento (correspondiente al Módulo I) se presentan los conceptos y nociones básicas de programación.

Se pretende que luego de leer este documento el estudiante haya adquirido la motivación y los conocimientos necesarios para continuar con el **módulo II** en donde realizará una serie de ejercicios reales de programación.

1.1. Pre-requisitos

Como pre-requisitos se asume que el estudiante tiene algunos conocimientos generales sobre el sistema operativo que utiliza como son el manejo de archivos y la

instalación y gestión de aplicaciones de uso general.

Se requiere además tener instalado el entorno de *Python*. En caso de no tenerlo aún, se recomienda acceder al siguiente link para ver las instrucciones paso a paso para la descarga y la instalación: Instrucciones para descargar e instalar el entorno Python

1.2. Consideraciones generales

Los temas se irán presentando de forma gradual y sin entrar en detalles formales rigurosos, mas bien se pretende dar una introducción informal y varios ejemplos y ejercicios para que el lector valla entendiendo la utilidad y asimilando la forma de aplicación de cada tema.

1.3. Nomenclatura para nombres de archivo de programas

Respecto de los ejercicios, algunos se indican realizar directamente sobre el IDLE y otros proponen escribir un programa para guardar y ejecutar. Un programa es un archivo cuyas lineas son instrucciones de programación escritas en un lenguaje determinado, en nuestro caso Python.

En el enunciado de cada ejercicio donde se solicita escribir un programa, se indica la palabra **Ejercicio_X_Y**. El programa correspondiente a ese ejercicio se almacena en el archivo de nombre **ejer_X_Y.py** dentro de la carpeta Scripts del directorio de instalación de Python.

Por ejemplo, si se presenta el ejercicio **Ejercicio_4_1** entonces el programa se debe almacenar con el nombre **ejer_4_1.py**.

Capítulo 2

Conceptos básicos

En este capítulo se introduce un conjunto de conceptos básicos que se utilizan en la jerga de la programación. No es necesario que el lector comprenda todos los conceptos en una primera lectura ya que se espera que los valla asimilando en la medida que progrese con el curso.

2.1. Sentencias

Una **sentencia** es una instrucción de programación indivisible, esto generalmente es una línea de código. Por ejemplo la siguiente línea de código es una sentencia *Python* que asigna un valor a una variable.

```
varNombre="un valor"
```

2.2. Programas

Un **programa** es un conjunto de sentencias escritas en un cierto orden para realizar una tarea determinada. En Python un programa se guarda en un archivo de extensión **py**. Al contenido de un archivo **py** se lo llama código fuente.

2.3. Variables

Los programas principalmente manipulan datos, los datos se almacenan en elementos del programa llamados **variables**. Las variables se identifican por un nombre y contienen un valor determinado que se le es asignado durante la ejecución del programa. Por ejemplo, si durante la ejecución del programa se ejecuta la siguiente sentencia:

```
edad=28
```

Entonces la variable *edad* pasará a tener el valor *28*.

2.4. Funciones

Las **funciones** contienen un conjunto de instrucciones de uso común y se puede reutilizar las veces que sea necesario dentro de los programas. Phyton provee de forma pre-definida un gran número de funciones y a su vez el usuario puede definir otras tantas funciones como desee.

Una función puede o no recibir valores de entrada y realizar alguna acción en función de esos valores. Los valores de entrada se llaman **argumentos** o **parámetros** de la función. Una función se invoca por su nombre y una lista de uno o más parámetros delimitados por paréntesis y separados por coma.

Por ejemplo, la siguiente sentencia invoca a la función `calcularArea` pasando como parámetros los números 3 y 2.

```
calcularArea(3, 2)
```

2.5. Código compilado vs código interpretado

En muchos lenguajes de programación, como por ejemplo Java, el código fuente no se ejecuta directamente sino que debe ser previamente compilado y lo que se ejecuta es el *código compilado*. El proceso de compilación lo que hace es interpretar el código escrito por el programador y generar un *código a nivel de máquina*. El código compilado es más eficiente para la máquina pero ilegible para las personas.

Cuando el código no es compilado se dice que se ejecuta en modo interpretado. A los programas que ejecutan en modo intérprete también se los suele llamar *Scripts*.

A los programas Python se los puede ejecutar tanto en modo interpretado como en modo compilado, es por ello que en la web es usual que se mencione Script Python para hacer referencia a un programa Python.

Por simplicidad en este curso comenzaremos utilizando el modo interpretado.

Capítulo 3

Tipos de datos primitivos

En un programa principalmente se manipulan datos, los datos se almacenan en **variables**. Todos los lenguajes masivos de programación poseen al menos los tipos de datos primitivos **Numérico, String y Booleano**. Luego cada lenguaje puede presentar otros tipos de datos adicionales. En las siguientes secciones se enumeran los tipos de datos primitivos de *Python* más comunes.

3.1. Tipos de datos numéricos

Los tipos de datos numéricos que utilizaremos son **enteros (int) y flotantes (float)**. El tipo entero representa un número entero y el tipo flotante representa un número con punto decimal.

Por ejemplo, podemos escribir la siguiente instrucción para definir una variable de nombre *varEntera* que almacena el número 10:

```
varEntera=10
```

Y de la misma forma podemos definir la variable de nombre *varFlotante* que almacene el número 10,5 (observar que se debe utilizar el punto como separador decimal)

```
varFlotante=10.5
```

Ejercicio 3_1: Ingresar lo siguiente desde el IDLE

```
>>> varEntera=10
>>> varFlotante=10.5
>>> varEntera
>>> varFlotante
```

Observar que al tipear *varEntera* el intérprete retorna el valor que contiene la variable varEntera y lo mismo sucede con la variable varFlotante.

La figura 3.1 ilustra el ejercicio realizado desde el IDLE.

Figura 3.1: Ejer_3_1

3.2. Tipo de dato String

Un string es una cadena de caracteres. Los strings se declaran entre comillas dobles de la siguiente manera:

```
varString="Este es un string"
```

Ejercicio 3_2: Definir la variable varString asignándole el valor "Este es un string" en el IDLE y luego introducir el nombre de la variable para inspeccionar su contenido.

Figura 3.2: Ejer_3_2

3.3. Tipo de dato Booleano

Una variable de tipo booleano puede almacenar los valores **True** o **False** (**verdadero** o **falso**). Este tipo de variables se utilizan principalmente en sentencias condicionales

que veremos más adelante. De momento solo nos interesa aprender a declararlas. A modo de ejemplo se indica como declarar las variables *varBooleanT* y *varBooleanF* asignándoles valores *True* y *False* respectivamente.

```
varBooleanT=True
varBooleanF=False
```

Ejercicio_3_3: utilizando el IDLE definir las variables *varBoolean1* y *varBoolean2* asignándoles valores *True* y *False* respectivamente y luego explorar el valor de las mismas.

```
>>> varBoolean1=True
>>> varBoolean2=False
>>> varBoolean1
True
>>> varBoolean2
False
```

3.4. Tipo de datos Lista

Además de los tipos de datos simples enumerados hasta aquí, Python presenta algunos tipos de datos complejos como son las listas. Una lista es una colección de elementos (datos) de cualquier tipo (numérico, string, booleano, etc), incluso un elemento de una lista puede ser otra lista. Una lista se define utilizando [ ] como delimitadores y los elementos se separan por comas. A continuación la definición de una lista.

```
listaEjemplo=["String", 10, True]
```

Esta sentencia define la lista de nombre listaEjemplo que tiene los valores "String" de tipo string, 10 de tipo numérico y True de tipo booleano.

Por ejemplo, se podría definir una lista indicando el nombre, la edad, el sexo, el peso y la altura de una persona de la siguiente manera:

```
listaPersona=["Juan", 22, "M", 72.5, 1.65]
```

Se accede a un dato de una lista a través de su índice, esto es su posición. El primer elemento corresponde al índice 0. Por ejemplo para conocer el tercer elemento de la lista anterior se puede ejecutar lo siguiente:

```
>>> listaPersona=[2]
'M'
```

Las listas son **mutables**, esto significa que sus elementos se pueden modificar en cualquier momento. Por ejemplo, podemos modificar el primer elemento de la lista anterior de la siguiente manera:

```
>>> listaPersona[0]="Jose"
>>> listaPersona
["Jose", 22, "M", 72.5, 1.65]
```

3.4.1. Listas anidadas

Una lista puede contener a otra lista como elemento. El siguiente ejemplo define en primer lugar las listas *persona1* y *persona2* con dos elementos que indican el nombre y la edad de una persona. Y luego define la lista *personas* que contiene como elementos a las listas *persona1* y *persona2*.

```
>>> persona1=['Juan', 23]
>>> persona2=['Pedro', 21]
>>> personas=[persona1, persona2]
>>> personas
[['Juan', 23], ['Pedro', 21]]
```

Para acceder a un elemento de una lista anidada se requiere indicar en primer lugar el índice del elemento de la lista principal y en segundo lugar el índice del elemento de la sub-lista. Por ejemplo, si se quiere acceder al primer elemento de *personas2* se puede utilizar la siguiente instrucción:

```
>>> personas[1][0]
'Pedro'
```

Ejercicio_3_4_1: utilizando el IDLE definir listas anidadas y explorar los distintos elementos de esas listas.

3.5. Otros tipos de datos de Python

Además de los tipos de datos mencionados, Python cuenta con otros tipos de datos nativos como son las tuplas, diccionarios y conjuntos. Por ser estos menos comunes que los restantes no se van a considerar en este módulo ya que como se remarcó anteriormente el propósito de este curso es que el lector aprenda a programar con sentencias y tipos de datos comunes a la mayoría de los lenguajes de programación.

Capítulo 4

Sentencias de iteración con el usuario

La miseración del usuario con el programa se realiza a través de entrada y salida de datos. En este capítulo se describen las funciones que se utilizan para solicitar datos de entrada al usuario y las que se utilizan para mostrar los datos de salida al usuario.

4.1. Función print (para la salida de datos)

En *Python* se utiliza la función **print** para mostrar información al usuario, esto es, imprimir texto en pantalla. Esta función puede recibir como parámetros una lista de valores e imprime un texto resultante de la concatenación de esos valores.

A continuación varios ejemplos de uso de la función **print**.

Ejemplo: imprime el string constante "Hola".

```
>>>print("Hola")
Hola
```

Ejemplo: imprime la concatenación de dos strings.

```
>>> print("Hola", "Juan")
Hola Juan
```

Ejemplo: imprime la concatenación del string constante "Holaz el valor de la variable nombre.

```
>>> nombre="Juan"
>>> print("Hola", nombre)
Hola Juan
```

Ejemplo: imprime la concatenación de un string constante, el valor de la variable nombre y otro string constante.

```
>>> nombre="Juan"
>>> print("Hola", nombre, ". ¿Cómo estás?")
Hola Juan. ¿Cómo estás?
```

Ejemplo: imprime la concatenación de un string constante y el valor del resultado de una expresión aritmética.

```
>>> print("1 + 2 es ", 1 + 2)
1 + 2 es 3
```

Ejercicio 4_1: Primer programa Phyton!!

Escribir un programa Phyton con el nombre de archivo *ejer_4_1.py* que defina las variables *nombre* con el valor *"Juan"*, *edad* con el valor *20* y muestre en pantalla "[nombre] tiene [edad] años de edad".

Por ser el primer programa se enumeran los pasos para editar el archivo *py* y ejecutarlo.

- Desde el IDLE: *File -> New File*
- Escribir las siguientes instrucciones en el nuevo archivo:
  - nombre="Juan"
  - edad=20
  - print(nombre, "tiene", edad, ".ªños de edad")
- Guardar el archivo: *File -> Save (en la carpeta Scripts con nombre ejer_4_1.py)*
- Ejecutar el programa: pulsar la techa *<F5>*

Al ejecutar el programa desde el IDLE se visualizará el siguiente resultado:

```
Juan tiene 20 años de edad
```

4.2. Función input (para la entrada de datos)

Comúnmente la función input se utiliza en una sentencia que pide un valor y lo almacene en una variable.

A modo de ejemplo, la siguiente sentencia pide la edad de una persona y almacena el dato ingresado en la variable *edad*.

```
edad=input("Ingrese su edad:  ")
```

Ejercicio 4_2: Escribir un programa con nombre de archivo ejer_4_2.py que pida el nombre y la edad de una persona y muestre el mensaje: nombre tiene edad años de edad.

```
nombre=input("Ingrese su nombre: ")
edad=input("Ingrese su edad: ")
print(nombre, "tiene", edad, "años")
```

Ejecutar el programa, ingresar nombre y edad y visualizar el resultado.

Capítulo 5

Comentarios

Los comentarios son bloques de texto que utilizados de forma conveniente pueden ser de gran ayuda al programador para entender y mantener el código de los programas. Generalmente un comentario indica qué es lo que hace una instrucción o un bloque de código. El intérprete de Python ignora los comentarios, o sea, que los comentarios no tienen ningún efecto en el momento de la ejecución del programa.

Los comentarios pueden ser escritos en una línea o en varias líneas, a estos últimos se los llama comentarios multi líneas.

5.1. Comentarios en una línea

Todo texto que aparezca luego del carácter "#ƨ hasta finalizar la línea de texto se considera un comentario. A continuación algunos ejemplos.

```
# Este es un comentario en una línea.

print("Hola")   # Este comentario aparece luego de una instrucción
```

5.2. Comentarios multi líneas

Los comentarios multi líneas se delimitan con tres comillas dobles. A continuación un ejemplo.

```
"""Este es un comentario
escrito en más de una línea"""
```

Ejercicio 5_1: Escribir un programa con el siguiente código y ejecutarlo varias veces cambiando los valores de entrada.

```
""" Mi primer programa con comentarios
Variables de entrada del programa (ingresadas por el usuario):
  - nombre: nombre de la persona
  - edad: años de edad de la persona

Salida del programa:
  - Imprime el texto "{nombre} tiene {edad} años de edad"
"""

# Pide los datos de entrada
nombre=input("Ingrese el nombre: ")
edad=input("Ingrese la edad: ")

# Salida
print(nombre, " tiene ", edad, " años de edad")
```