

Programación en Python

Módulo 1

Sumário

1	Introducción	3
1.1	Introducción	3
1.2	Consideraciones generales	3
1.2.1	En qué archivos almacenar los programas	3
2	Conceptos básicos.....	3
2.1	Sentencias	3
2.2	Programas	3
2.3	Funciones	3
2.4	Código compilado vs código interpretado	4
3	Tipos de datos primitivos	4
3.1	Tipos de datos numéricos	4
3.2	Tipo de dato String.....	5
3.3	Tipo de dato Booleano.....	5
3.4	Tipos de datos Lista.....	5
4	Sentencias de iteración con el usuario	6
4.1	Función print (para la salida de datos).....	6
4.2	Función input (para la entrada de datos)	7
5	Comentarios en el código Python	8
5.1	Comentarios en una línea	8
5.2	Comentarios multi líneas	8
6	Operaciones con tipos de datos.....	8
6.1	Chequeo del tipo de dato de una variable.....	9
6.2	Conversiones de datos	9
6.2.1	Errores de conversiones de datos.....	10
6.3	Operadores	10
6.4	Operadores sobre tipo de datos numéricos	10
6.4.1	Operadores aritméticos	10

1 Introducción

1.1 Introducción

En este documento (correspondiente al **Módulo I**) se brinda una introducción informal a la programación utilizando como herramienta el lenguaje y entorno Python. Aquí se explican los conceptos básicos de programación y se introducen las nociones de entrada y salida, tipos de datos, operadores sobre tipos de datos, conversiones de datos, sentencias condicionales y bucles.

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 se desarrollará un programa complejo.

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.

1.2.1 En qué archivos almacenar los programas

Respecto de los ejercicios, algunos se solicitan realizar directamente sobre el IDLE y otros solicitan escribir un programa para guardar y ejecutar. En el enunciado de cada ejercicio donde se solicita un programa, se indica la palabra **Ejercicio_X**: donde X indica la sección donde se presenta el ejercicio. El programa correspondiente a ese ejercicio se debe almacenar en el archivo de nombre **ejer_X.py** dentro de la carpeta Scripts del directorio de instalación del Python.

Por ejemplo, si se presenta el ejercicio **Ejercicio_4_1** en la **sección 4.1** entonces el programa se debe almacenar con el nombre **ejer_4_1.py**.

2 Conceptos básicos

En este capítulo se introduce un conjunto de conceptos básicos que serán de utilidad a lo largo del curso.

2.1 Sentencias

Una **sentencia** es una instrucción atómica, esto es una línea de código que contiene una unidad ejecutable indivisible.

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 Funciones

Las funciones implementan un bloque de código de uso común que se puede reutilizar las veces que sea necesario dentro de los programas. Python provee un gran número de estas 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. Esta se invoca por su nombre y una lista de uno o más parámetros de entrada 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)
```

2.4 Código compilado vs código interpretado

En muchos lenguajes, 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. Cuando el código no es compilado se dice que se ejecuta en modo **intérprete**. 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 intérprete.

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
```

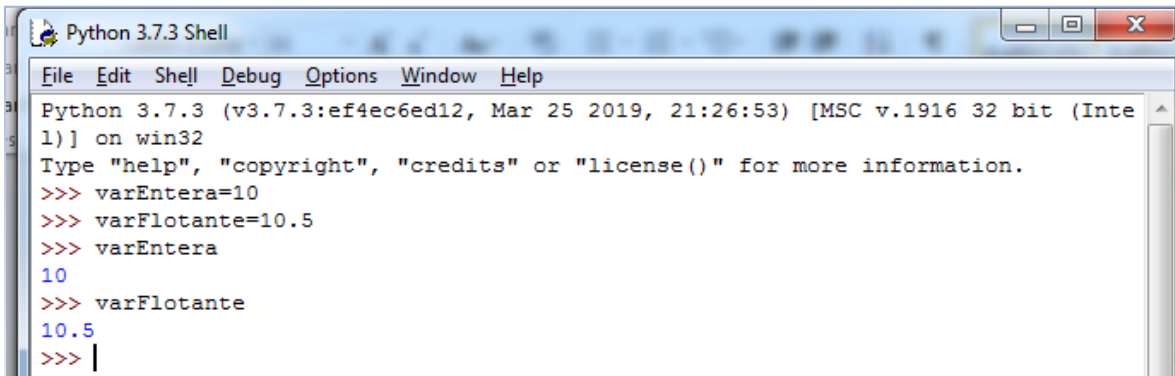
Y de la misma forma podemos definir la variable de nombre varFlotante que almacene el número 10,5 (observar que se debe utilizar el punto como separador decimal)

```
varFlotante=10.5
```

Ejercicio: Ingresar lo siguiente desde el IDLE (luego de cada línea pulsar <Enter>)

```
>>> 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.



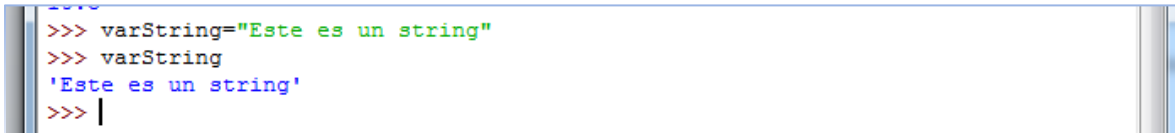
```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> varEntera=10
>>> varFlotante=10.5
>>> varEntera
10
>>> varFlotante
10.5
>>> |
```

3.2 Tipo de dato String

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

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

Ejercicio: 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.



```
>>> varString="Este es un string"
>>> varString
'Este es un string'
>>> |
```

3.3 Tipo de dato Booleano

Una variable de tipo booleano puede almacenar los valores **verdadero** o **falso** (**True** o **False** en *inglés*). 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: utilizando el IDLE definir las variables `varBoolean1` y `varBoolean2` asignandoles valores `True` y `False` respectivamente y luego explorar el valor de las mismas.

3.4 Tipos 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) que pueden ser de distintos tipos (numéricos, string, booleanos, etc). Una lista se define utilizando `[]` como delimitador y los elementos se separan por comas. A continuación la definición de una lista.

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

Esta sentencia define la lista con 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]
```

4 Sentencias de iteración con el usuario

Antes que nada un comentario sobre las funciones. Las funciones

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 un string.

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

Imprime: *Hola*.

Ejemplo: imprime la concatenación de dos strings.

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

Imprime: *Hola Juan*

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

```
nombre="Juan"
```

```
print("Hola", nombre)
```

Imprime: *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?")
```

Imprime: *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)
```

Imprime: *1 + 2 es 3*

Ejercicio 4_1: Primer programa Python!!

Escribir un programa Python con el nombre de archivo *ejer_4_1.py* que defina las variables nombre con el valor “Juan”, la variable edad con el valor 20 y muestre “{nombre} tiene {edad} años”.

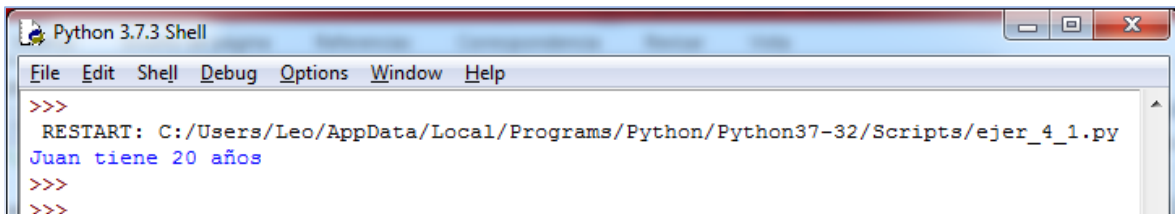
Por ser el primer programa se enumeran los pasos para editar el archivo y ejecutarlo:

- Desde el IDLE: File -> New File

- Escribir las siguientes instrucciones en el nuevo archivo:

```
nombre="Juan"  
edad=20  
print(nombre, "tiene", edad, "años")
```
- Guardar el archivo: File -> Save (en la carpeta Scripts con nombre ejer_4_1.py)
- Ejecutar el programa: <F5>

Se visualizará el siguiente resultado en el IDLE:



```
Python 3.7.3 Shell  
File Edit Shell Debug Options Window Help  
>>>  
RESTART: C:/Users/Leo/AppData/Local/Programs/Python/Python37-32/Scripts/ejer_4_1.py  
Juan tiene 20 años  
>>>  
>>>
```

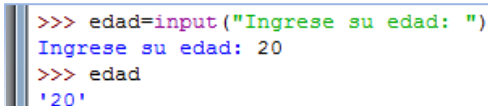
4.2 Función input (para la entrada de datos)

Comunmente 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: Desde el IDLE escriba la sentencia del ejemplo anterior y a continuación inspeccione el valor de la variable *edad* para comprobar que coincide con el ingresado por teclado.

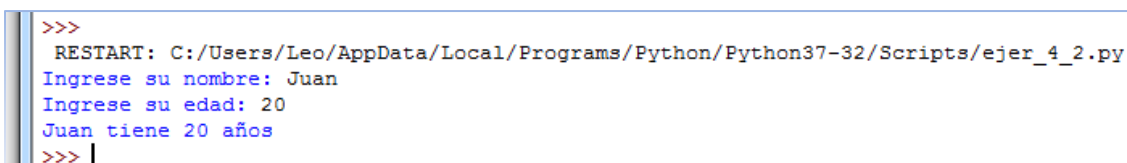


```
>>> edad=input("Ingrese su edad: ")  
Ingrese su edad: 20  
>>> edad  
'20'
```

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.

```
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.



```
>>>  
RESTART: C:/Users/Leo/AppData/Local/Programs/Python/Python37-32/Scripts/ejer_4_2.py  
Ingrese su nombre: Juan  
Ingrese su edad: 20  
Juan tiene 20 años  
>>> |
```

5 Comentarios en el código Python

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 multilínea.

5.1 Comentarios en una línea

Todo texto que aparezca luego del carácter '#' y 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 multilíneas se delimitan con tres comillas dobles. A continuación un ejemplo.

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

Ejercicio 5_2: 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")
```

6 Operaciones con tipos de datos

En este capítulo se mencionan las operaciones elementales que se pueden realizar sobre los tipos de datos.

Básicamente se explicará cómo inspeccionar el tipo de datos de una variable, cómo convertir de un tipo de datos a otro y las operaciones que se pueden realizar sobre cada tipo de datos.

6.1 Chequeo del tipo de dato de una variable

Más adelante se verá la importancia de conocer el tipo de datos de una variable. En otros lenguajes de programación como Java, el tipo de datos de una variable se indica explícitamente al declarar la variable. En cambio en Python, el tipo de dato es inferido por el intérprete de acuerdo al valor que se le asigne a la variable.

Por ejemplo, si se asigna un valor entero a una variable, Python asume que la variable es de tipo entero. Si se le asigna un string, Python asume que la variable es de tipo string y así con todos los tipos.

Para chequear el tipo de datos de una variable se utiliza la función **type**. Dicha función recibe el nombre de la variable como parámetro e indica de qué tipo (o clase) es esa variable.

Ejemplo:

```
>>> varEntero=10
>>> type(varEntero)
<class 'int'>
```

Indica que varEntero es de tipo int (entero).

Ejercicio: Desde el IDLE asignar un valor de cada tipo (entero, flotante, string, booleano, lista) a una variable y chequear el tipo de la variable con la función **type**.

```
>>> varTest=10
>>> type(varTest)
<class 'int'>
>>> varTest=10.5
>>> type(varTest)
<class 'float'>
>>> varTest="hola"
>>> type(varTest)
<class 'str'>
>>> varTest=True
>>> type(varTest)
<class 'bool'>
>>> varTest=[10, "hola"]
>>> type(varTest)
<class 'list'>
```

6.2 Conversiones de datos

Muchas veces puede ser necesario convertir de un tipo de datos a otro. Por ejemplo, pasar una variable de tipo str a int para poder aplicar sobre esta un operador aritmético.

Python brinda un conjunto de funciones de conversión de datos que se enumeran en la siguiente tabla.

Función	Convierte a
str()	str – Cadena de caracteres
int()	int - entero
float()	float - flotante
bool()	bool - Booleano

Al aplicar cualquiera de estas funciones sobre una variable, esta retornará el valor convertido al tipo indicado. Luego ese valor puede ser utilizado en una expresión o reasignarse a la misma variable haciendo que esta cambie de tipo. A continuación algunos ejemplos.

Ejemplo: convierte el tipo de la variable numero a int para poder utilizarlo en la suma.

```
>>>numero="10"
>>>int(numero) + 5
15
```

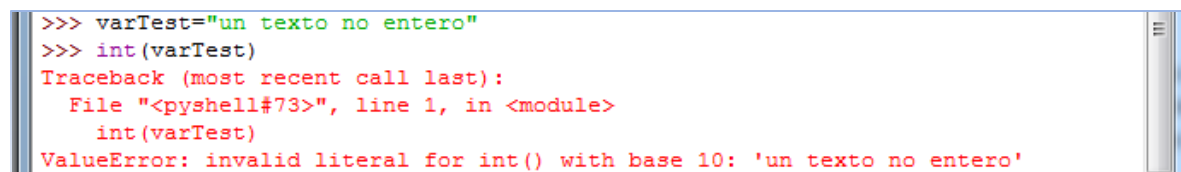
Ejemplo: reconvierte la variable número del tipo str al tipo int.

```
>>>numero="10"
>>>type(numero)
<class 'str'>
>>>numero=int(numero)
>>>type(numero)
<class 'int'>
```

6.2.1 Errores de conversiones de datos

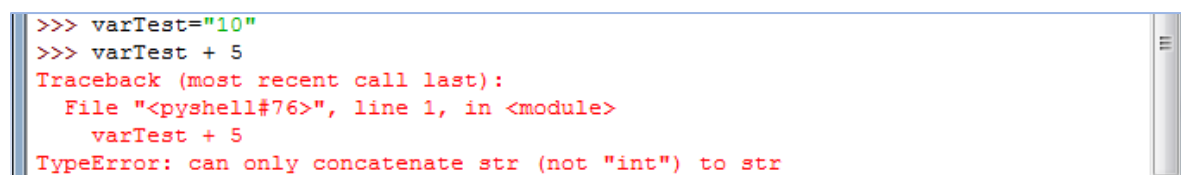
Cuando se intenta realizar una conversión u otra operación de tipo de datos sobre un tipo incorrecto el intérprete arroja un error indicando el tipo de problema.

Por ejemplo, la siguiente figura muestra el error que arroja el intérprete al intentar convertir a entero un string que tiene un valor que no representa un número entero.



```
>>> varTest="un texto no entero"
>>> int(varTest)
Traceback (most recent call last):
  File "<pyshell#73>", line 1, in <module>
    int(varTest)
ValueError: invalid literal for int() with base 10: 'un texto no entero'
```

De la misma forma, la siguiente figura muestra el error que arroja el intérprete al intentar aplicar la operación suma sobre un tipo no soportado.



```
>>> varTest="10"
>>> varTest + 5
Traceback (most recent call last):
  File "<pyshell#76>", line 1, in <module>
    varTest + 5
TypeError: can only concatenate str (not "int") to str
```

6.3 Operadores

Se llaman operadores a los símbolos (signos o palabras reservadas) que el intérprete de Python identifica como una operación a realizar sobre datos de un tipo determinado. Por ejemplo el símbolo + indica la operación suma entre dos datos de tipo numérico. A continuación se explicarán los operadores que brinda Python para cada tipo de datos.

6.4 Operadores sobre tipo de datos numéricos

6.4.1 Operadores aritméticos

Operador	Descripción
----------	-------------

+	Suma
-	Resta
-	Negativo
*	Multiplicación
**	Exponente
/	División
//	División entera
%	Residuo

Ejercicio: Desde el IDLE asignar el valor 3 a la variable num1 y el valor 2 a la variable num2. Luego hacer un print para mostrar el resultado de aplicar cada uno de los operadores a esas dos variables.

```
>>> n1=3
>>> n2=2
>>> print("n1+n2: ", n1+n2)
n1+n2: 5
>>> print("n1-n2: ", n1-n2)
n1-n2: 1
>>> print("-n1: ", -n1)
-n1: -3
>>> print("n1*n2: ", n1*n2)
n1*n2: 6
>>> print("n1**n2: ", n1**n2)
n1**n2: 9
>>> print("n1/n2: ", n1/n2)
n1/n2: 1.5
>>> print("n1//n2: ", n1//n2)
n1//n2: 1
>>> print("n1%n2: ", n1%n2)
n1%n2: 1
>>> |
```

