Introducción

Es importante para un programador definir los datos de entrada y el formato adecuado para ellos. Igualmente es valioso saber cómo realizar cálculos y aplicar funciones predefinidas por Python, para generar unos resultados que deben ser entregados al usuario en un formato legible y claro.

En este componente formativo se explicará paso a paso cómo codificar la entrada y salida de datos, el uso de instrucciones secuenciales para manejo de constantes y variables y la aplicación de funciones definidas por el lenguaje. Además, se detallará el procedimiento para importar las librerías disponibles de Python con funciones de fecha, de números aleatorios y de matemáticas.

1. Entrada de datos

Cuando se usa la consola o el terminal, es común solicitar al usuario introducir datos a través del teclado. A continuación se detallará la codificación para realizar la entrada de información por consola.

1.1 Entrada estándar

Para solicitar al usuario que introduzca algún dato a través del teclado, se debe usar el método **input().**

Este método recibe como parámetro un mensaje al usuario entre comillas:

```
Ejemplo

>>> edad = input(“¿Cuántos años tienes?”)

¿Cuántos años tienes? 28

>>> edad

'21'
```

Otra forma de realizar la entrada de datos a través del teclado sería:

```
Ejemplo:

print("¿Cómo se llama?")
```

```
nombre = input()

print("Usted”, nombre, “es un aprendiz SENA: ” )
```

La entrada siempre es tipo cadena de caracteres **(str)**. Esto es útil para la entrada de datos tales como nombre, ciudad, cargo, deporte, etc.

Por tanto, si lo que se necesita es un dato de cierto tipo especial, por ejemplo, un **int**, habrá que hacer la conversión correspondiente en **input**, de esta manera:

```
Ejemplo:

>>> celular = int(input(“Danos tu número de celular:”))

Danos tu número de celular: 3125320125

>>> celular

3125320125
```

Pero si lo que se requiere es un dato de tipo ***float***, habrá que hacer la siguiente conversión:

```
Ejemplo:

>>> estatura = float(input(“Cuál es tu estatura?”)

Cuál es tu estatura? 1.75

>>> estatura

1.75
```

1.2 Entrada por script

Hasta ahora se ha escrito código directamente en la consola del intérprete, pero es necesario aprender a realizar programas informáticos que contengan todas las instrucciones en archivos llamados *scripts* (o guiones de instrucciones). Luego se envía este archivo al intérprete desde la terminal y se ejecuta todo el bloque de instrucciones. Guzdial y Vidal (2013) recomiendan aplicar buenas prácticas en programación.

De esta forma se pueden realizar todas las variaciones deseadas, sencillamente modificando el bloque de instrucciones almacenadas en un archivo con **extensión .py**

Para poder crear y ejecutar scripts se necesita un editor (IDE) y una terminal.

Ejemplo

Si el bloque de instrucciones se almacena en el archivo **"programa1.py"** en la carpeta **c:\sena** conteniendo las siguientes instrucciones:

```
print("hola, bienvenido a tu primer script")

a=input("digite un nombre")

b=int(input("digite un número"))

print("su nombre es ", a)

print("su número es ", b)

c=b**2

print("el valor del número al cuadrado es", c)
```

Se ejecuta el bloque de instrucciones en el computador digitando este comando:

c:\sena> python programa1.py <ENTER>

```
C:\sena>python programa1.py

Hola, bienvenido a tu primer script

digite un nombre MARIA

digite un número 5

su nombre es MARIA

su número es 5

el valor del número al cuadrado es 25
```

Realizando cambios en las instrucciones y regrabando el archivo, se verifican nuevamente los resultados ejecutando el programa1.py en el intérprete de Python.

IDE para Python

Existen múltiples IDE (Entornos de Desarrollo Integrado o *Integrated Development Environment*), los cuales pueden usarse para digitar bloques de código en lenguaje Python.

Según Salazar (2019), un IDE consta de un editor de código fuente, un resaltador de sintaxis, unas herramientas de construcción automáticas y un depurador. La mayoría de los IDE tienen auto-completado inteligente de código.

Entre los IDE más utilizados para Python tenemos: *Sublime Text, PyCharm, Atom, Pythonista, Eclipse, Komodo, CodePad, VIM y Spyder Python*.

Utilizar el IDLE (*Integrated DeveLopment Environment for Python*) el cual es un entorno gráfico que permite editar, guardar y ejecutar programas en Python.

IDLE es también un entorno interactivo similar a una consola, en el que se pueden ejecutar instrucciones directas de Python.

En Windows, el IDLE viene junto con el intérprete de Python, es decir, al instalar Python en Windows también se instala el IDLE.

La ventana del IDLE tiene un tamaño predeterminado que es posible modificar con el menú ***Options > Configure IDLE > General.*** La gama original de colores de la ventana del IDLE se puede modificar con el menú ***Options > Configure IDLE > Highlights***.

Para crear un nuevo programa se selecciona File New-File, o se oprime el atajo CTRL+N.

El IDLE aplica color al texto de acuerdo con su sintaxis. Los colores facilitan identificar los distintos tipos de datos y permiten detectar errores:

Las palabras reservadas de Python se muestran en color naranja.

Las cadenas de texto se muestran en verde.

Las funciones se muestran en púrpura.

Los resultados de las instrucciones se escriben en azul.

Los mensajes de error aparecen en rojo.

Para guardar el programa se selecciona File Save o se oprimen CTRL+S. El programa queda almacenado con el nombre deseado y la extensión de Python que es **.py**.

Para ejecutar el programa digitado, seleccionar en el ***menú Run > Run Module*** u oprimir la tecla F5.

2. Salida de datos

Para lograr la salida de datos se pueden combinar textos y variables separados con comas. Es una buena práctica acompañar esta información con mensajes claros al usuario, de manera tal que este identifique el tipo de dato generado y la unidad de medida en la cual se expresa.

Ahora se detallará la sintaxis para codificar la salida de información por pantalla.

2.1 Sentencia print

Para mostrar datos por pantalla, con frecuencia se utiliza la función **print**.

La sintaxis básica de **print** para presentar un mensaje es mencionando directamente el mensaje o almacenando el mensaje en una variable de texto y luego imprimir la variable, de la siguiente manera:

```
Ejemplo:

print (“Hola Mundo”)

Hola Mundo

texto=”SENA es Colombia”

print(texto)

SENA es Colombia
```

Mientras que si se desea mostrar el valor de una variable numérica, debe hacerlo así:

```
print (salario)
```

Dentro del mensaje se pueden utilizar los siguientes caracteres especiales:

\n = salto de línea. ‘\n’ termina la línea y mueve el cursor a otra línea.

\t = es el carácter “tab horizontal" y se usa para simular sangrías de texto.

Una forma de colocar varias cadenas de texto o intercalarlas con variables es colocarlas separadas por comas, el lenguaje incluye un espacio entre ellas.

Ejemplo:

```
print (“El salario devengado por “, empleado, “es la suma de “, salario)

El salario devengado por MARIA es la suma de 234500

print("El salario devengado por", empleado, "\nes la suma de $", salario)

El salario devengado por MARIA es la suma de $ 234500

print("El salario devengado por \t", empleado, "\tes la \tsuma de $", salario)

El salario devengado por MARIA es la suma de $ 234500
```

2.2 Salida de datos con formato

Python es compatible con la salida de datos con formato. El caracter módulo % es un operador integrado en Python. Este es conocido como el operador de interpolación. Se deben digitar los formatos deseados, el signo % (porcentaje), seguido por paréntesis con los datos que necesitan ser convertidos.

La sintaxis de la instrucción es:

```
print (“cadena con formato” % (variables separadas por comas))
```

Se utilizan los siguientes formatos para la salida de datos:

```
%c = un caracter

%s = str, cadena de caracteres

%d = int, enteros

%f = float, flotantes
```

Ejemplo:

```
titulo = "raíz cuadrada de tres"

valor = 3**0.5

print ("el resultado de %s es %f " % (titulo, valor))

el resultado de raíz cuadrada de tres es 1.732051

print "Cliente: %s, ¿Activar S o N?: %c" % ("PatPrimo", "S")

Cliente: PatPrimo, ¿Activar S o N?: S

>>> print "Nro. factura: %d, Total a pagar: $ %f" % (567, 12500.83)

Nro. factura: 567, Total a pagar: $ 12500.83
```

Es posible controlar el formato de salida, por ejemplo, para obtener el valor con ocho (8) dígitos después del punto decimal, se digita %.8f; para una salida con 2 decimales se digita %.2f

```
Ejemplo:

titulo = "raíz cuadrada de tres"

valor = 3**0.5

print ("el resultado de %s es %.8f " % (titulo, valor)) #salida con 8 decimales

el resultado de raíz cuadrada de tres es 1.73205081

print ("el resultado de %s es %.1f " % (titulo, valor)) #salida con 1 decimal

el resultado de raíz cuadrada de tres es 1.7
```

2.3 Impresión de cadenas

Las cadenas tienen varias formas de ser impresas en pantalla. Veamos el siguiente cuadro:

```
frase1= “El perro”

frase2= “Manchas”

frase3= “ladra en la oscuridad”
```

Notación	Uso	Ejemplo	Salida en pantalla
+	Concatena cadenas	frase1+" " + frase2+ " " + frase3 + "?"	El perro Manchas ladra en la oscuridad?
*	Repite cadenas	cadena4= frase2 *3	ManchasManchasManchas
cadena.capitalize ()	Inicia con mayúscula	frase3.capitalize()	Ladra en la oscuridad
cadena.center(ancho)	Centra en el ancho dado.	frase1.center(18)	" El perro "
cadena.lower()	Pasa todo a minúscula	frase2.lower()	manchas
cadena.upper()	Pasa todo a mayúscula	frase2.upper()	MANCHAS
cadena.title()	Mayúsculas iniciales	frase3.title()	Ladra En La Oscuridad

3. Instrucciones secuenciales con Python

Una vez leídos los datos de entrada, se debe proceder a realizar los cálculos necesarios para obtener la información requerida por el usuario. Estas instrucciones se deben digitar una tras otra, verificando que el procedimiento lógico sea correcto y los operadores empleados sean los adecuados. Es indispensable cumplir con la sintaxis aceptada por Python.

En la codificación de las instrucciones se pueden utilizar comentarios, constantes, variables, asignaciones, operadores aritméticos y funciones.

3.1 Comentarios en Python

Ortega (2018) define un comentario en Python como una línea de texto no ejecutable, es decir, no es una línea de código.

Es una buena práctica en programación documentar el código para mayor claridad. Se pueden hacer comentarios en Python para ilustrar acerca de lo que hace el programa. Se pueden incluir tantos comentarios como se requieran.

Hay dos formas para hacer comentarios:

Digitando el símbolo # al comienzo del comentario o dentro de la línea de código.

Digitando triple comilla **(“)** al principio y al final del comentario, cuando este ocupa varias líneas.

```
Ejemplo:

# SENA: Colombia

# Regional: Santander

a=30 # a representa el número de aprendices por Ficha

b=9 # b representa el número de Fichas de programación de software

"""Estos son datos de los 8 centros de formación ubicados en el departamento de Santander."""
```

3.2 Instrucciones de asignación

Una variable es una forma de identificar un dato que se encuentra almacenado en la memoria del computador. El valor de la variable puede ir cambiando a medida que se ejecutan las instrucciones del programa.

Nombres de variables válidos en Python:

Secuencias de letras y dígitos.

El nombre de la variable debe iniciar con una letra.

Se puede usar el guion bajo (_).

A las variables se les puede asignar un valor de cualquier tipo (int, float, str, booleano, lista) o una expresión aritmética.

En la siguiente tabla se verán varios operadores aritméticos que pueden ser utilizados en una asignación:

3.3 Inicialización de variables

Algunas variables requieren ser inicializadas con un valor, usualmente ese valor es cero.

Cuando asignamos un valor a una variable por primera vez, se dice que se define e inicializa la variable. En un programa Python se pueden definir las variables en cualquier lugar del programa. Sin embargo, es una buena práctica definir las variables al principio del programa.

Para asignar un valor a una variable se utiliza el operador de asignación =.

No es necesario declarar inicialmente una variable con un tipo de datos; al asignar un valor a una variable, se declara e inicializa la variable con ese valor.

```
# Ejemplo:

b= 100 #b se inicializa con un valor entero

c= b**3 #c se inicializa con una expresión

print(c)
```

Si intentamos usar una variable que no ha sido inicializada, el intérprete mostrará un error:

```
# Ejemplo:

>>> print(z)

Traceback (most recent call last):

File "<input>", line 1, in <module>

NameError: name 'z' is not defined
```

3.4 Uso de constantes y variables

Una constante es un campo de memoria donde su valor o contenido no cambia: la única diferencia es que las constantes van en mayúscula, por ejemplo, CONSTANTE1. Desde el punto de vista funcional no existe ninguna diferencia entre las constantes y variables en Python. Es recomendable, como buena práctica, que no se modifique el valor de una constante posteriormente.

```
# Ejemplo:

IVA = 0.19

precioinicial = 3000

preciofinal = precioinicial * (1.0+IVA)

print(preciofinal)

3570
```

Una **variable** es una unidad de datos que puede cambiar de valor. El programador puede decidir el nombre de las variables, se recomienda que sea claro y conciso, y también puede escribir funciones que trabajen con los valores contenidos en ellas.

Para modificar el valor de una variable, simplemente se le asigna un nuevo valor en el programa, este puede ser un literal, una expresión, una llamada a una función o una combinación de todos ellos.

Un literal es un dato puro, que puede ser un número, una cadena de caracteres o un booleano.

```
# Ejemplo:

a= 10000 # a se inicializa con un número entero

b= a * 2/100 # b se inicializa con una expresión

c= "JUAN CARLOS" # c se inicializa con una cadena

print( c, “debe la suma de $”, a+b)

JUAN CARLOS debe la suma de $ 10200
```

Es posible en Python asignar un mismo valor a múltiples variables a la vez. Se pueden definir varias variables con un mismo dato, así:

```
a = b = c = 10 # Inicializan a, b y c con el valor entero 10

>>> print(a)

10

>>> print(b)

10
```

```
>>> print(c)

10
```

También está permitido, en una sola instrucción, inicializar varias variables con un valor diferente para cada una:

```
>>> a, b, c = 100, 200, 300

>>> print(a)

100

>>> print(b)

200

>>> print(c)

300
```

Existen unas variables especiales denominadas acumuladores y otras denominadas contadores.

Un **acumulador** es una variable a la que le vamos sumando un determinado valor.

Ejemplo: suma=suma+edad

La variable suma, acumula las edades.

Un **contador** es una variable a la cual le vamos sumando/restando un valor constante.

Ejemplo: total=total+1

La variable total se incrementa en 1 cada vez que se ejecuta esta instrucción.

3.5 Funciones predefinidas de Python

Existe una diversidad de funciones predefinidas en Python o funciones internas. Buttu (2016) menciona algunas de las funciones más utilizadas y conocidas:

3.6 Librerías de fecha, random y matemáticas

En Python se pueden utilizar funciones predefinidas para manejar fechas, generar números aleatorios y aplicar funciones matemáticas, importando algunas librerías, las cuales, según Caballero (2019), se pueden aplicar en Big Data.

A

Librerías de fecha

Si se necesita usar fechas en un programa, es necesario importar las librerías Python: ***datetime y date***

```
from datetime import date

from datetime import datetime

hoy = date.today() #fecha actual

print(“Hoy es el dia: “, hoy)

Hoy es el día: 2020-10-28

fecha = datetime.now() #Fecha actual con hora completa

print("La fecha completa de hoy es: ",fecha)

La fecha de hoy es: 2020-10-28 17:42:17.961801
```

B

Librerías random

Si se requiere generar números aleatorios (randómicos), es necesario cargar la librería random.

Aquí se presentan unos ejemplos de las posibles funciones con números aleatorios:

```
import random

a= random.randint(10,100) #genera un número entero aleatorio entre 10 y 100

print (a)

99

b= random.randrange(0,100,5) genera un número entero aleatorio entre 0 y 100 incrementos de 5

print (b)

25
```

```
c= random.random() #genera un número float aleatorio entre 0.0 y 1.0

print (c)

0.998394986213598

d= random.uniform(100, 200) #genera un número float aleatorio entre 100.0 y 200.0 inclusive.

print (d)

156.5039706179512
```

La función ***choice*** permite seleccionar al azar un dato desde un conjunto.

```
Import random

amigos = ['Luis', 'Mauricio', 'Patricia', 'Carlos']

ganador= random.choice(amigos)

print(“El ganador fue: “,ganador)

El ganador fue: Patricia
```

La función ***shuffle*** modifica el orden de los elementos de una lista.

```
Import random

naipes = [1, 2, 3, 4, 5, 6, 7, 10, 11, 12]

random.shuffle(naipes) #los naipes se barajaron al azar

print(naipes)

[3, 6, 2, 12, 5, 1, 4, 11, 10, 7]
```

La función ***sample.*** Esta función extrae una cantidad de números aleatorios de un conjunto.

```
Import random

naipes = [1, 2, 3, 4, 5, 6, 7, 10, 11, 12]

random.sample(naipes, 3) #se tomaron 3 cartas del total de naipes

print(naipes)
```

```
[4, 11, 3]
```

C

Librerías matemáticas

Cervantes (2017) hace referencia a unas funciones matemáticas que vienen predefinidas por Python y otras que requieren ser importadas de la librería ***math***.

La función ***trunc()*** elimina los decimales de un número ***float***.

```
import math

a=123.56

b=math.trunc(a)

print(“El valor truncado es: “,b)

123
```

La función ***fabs()*** calcula el valor absoluto de un número float, eliminando el signo.

```
import math

a= - 200.45

b=math.fabs(a)

print(“El valor absoluto es: “,b)

200.45
```

La función ***factorial()*** calcula el número de combinaciones posibles de una serie de objetos. El factorial se expresa como n!. ejemplo: 0! == 1. Es importante aclarar que la función factorial() solo se utiliza con números enteros.

```
import math

a= 6

b=math.factorial(a)

print(“El valor factorial de “,a, “ es: “,b) # 6! es igual a 6x5x4x3x2x1

720
```

La función ***fmod()*** calcula el residuo de una división entre dos números float.

```
import math

c= math.fmod(16,5)

print(“el residuo de dividir 16 entre 5 es ”,c)

el residuo de dividir 16 entre 5 es 1.0
```

La función ***sqrt()*** calcula la raíz cuadrada de un número entero.

```
import math

a=3

c= math.sqrt(a)

print(“la raíz cuadrada de 3 es: ”,c)

la raíz cuadrada de 3 es: 1.7320508075688772
```

Las funciones trigonométricas seno, coseno y tangente se realizan usando ***sin(), cos() y tan()***.

Las funciones trigonométricas en la librería math toman los valores de los ángulos expresados como radianes. Por esta razón, debe realizarse la conversión de grados a radianes con la función ***radians***.

```
import math

angulo= 60

radianes = math.radians(angulo)

print(“los radianes son:”,radianes)

los radianes son: 1.0471975511965976

seno= math.sin(radianes)

coseno= math.cos(radianes)

tangente=math.tan(radianes)

print(“el seno es:”,seno)

el seno es: 0.8660254037844386
```

```
print(“el coseno es:”,coseno)

el coseno es: 0.5000000000000001

print(“la tangente es:”,tangente)

la tangente es: 1.7320508075688767
```

Existen muchas funciones matemáticas adicionales tales como exponenciales, hiperbólicas, trigonométricas inversas, distancia entre coordenadas, entre otras.