Introducción

Python es un lenguaje de programación de uso libre, permite crear sitios web, juegos, software científico, gráficos, entre otros, siendo uno de los lenguajes más utilizados hoy en la industria desarrolladora.

Creado a finales de 1989 en los países bajos por Guido Van Rossum, quien fue desarrollador de Google y ahora desarrollador de Dropbox, dicho lenguaje de programación hace parte de CWI (Centro de investigaciones holandés de carácter oficial).

Hinojosa (2016) menciona que el nombre "Python" viene dado por la afición de Van Rossum al grupo musical Monty Python Flying Circus, un grupo cómico británico de seis humoristas de los años 1960. Actualmente el logo de Python son dos serpientes pitón (python en inglés) en colores azul y amarillo, ver figura 1.

Figura 1. Logo actual de Python

En 1991 se publicó la versión 0.9.0. En 1994 se publicó la versión 1.0, en el 2000 se publicó la versión 2.0 y en el 2008 se publicó la versión 3.0.

Desde 2001 hasta 2018, la Python Software Foundation, es dueña de todo el código, documentación y especificaciones del lenguaje. Después del 2019, el desarrollo de Python está dirigido por un consejo de dirección de cinco miembros elegidos entre los desarrolladores de Python.

La Figura 2 presenta la fecha de publicación de las diferentes versiones de Python. Las versiones con punto rojo se consideran obsoletas, las versiones con punto azul se encuentran en actualizaciones y las versiones con punto blanco corresponden a futuras versiones.

Figura 2. Versiones del lenguaje Python

Python es un lenguaje de programación interpretado, por lo que funciona en cualquier sistema operativo que integre su interpretador.

Dicho lenguaje es multiplataforma y multiparadigma, sirve para desarrollar aplicaciones web o móviles. Este lenguaje de programación dispone de frameworks que apoyan el desarrollo de juegos y algoritmos científicos de cálculos avanzados. Además, es una herramienta conveniente para el área de Machine Learning.

Este lenguaje cuenta con una gran calidad en su sintaxis, utiliza bloques de código, los cuales llevan indentaciones, lo que garantiza una gran legibilidad en el código fuente. El código de Python 2.x no necesariamente debe correr en Python 3.0.

```
# Ejemplo

---------------------------------

Python 3.x

print (“Hola Mundo”)

---------------------------------

Python 2.x

print "Hola mundo"
```

1. Proceso de instalación de Python

Paso 2Paso 3Paso 4

Para realizar el proceso de instalación es necesario ir al sitio oficial de Python (http://www.python.org) y descargar el instalador ejecutable en el menú superior: la opción *Downloads.*

1.1 Comentarios en Python

Un comentario es una línea de texto no ejecutable, esto quiere decir que el compilador o intérprete no la tomará como una línea de código.

Es una buena práctica en programación documentar el código para mayor claridad en un proyecto. Se pueden hacer comentarios en Python para dar pequeñas explicaciones acerca de lo que hace el programa. Se pueden usar tantos comentarios como sean necesarios.

Hay dos formas para hacer comentarios:

1

Digitando el símbolo # al comienzo del comentario.

2

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

```
Ejemplo
```

```
# Autor: Pepito Pérez

# Ciudad: Bogotá

>>> x=0

>>> y=8 # Los omentarios también pueden estar dentro de una línea

""" Este es un comentario multilínea.
Podemos escribir varias líneas sucesivamente en la documentación"""
```

1.2 Errores de tecleo y excepciones

Si se digita incorrectamente una expresión, Python nos lo indicará con un mensaje de error. El mensaje de error proporciona información acerca del tipo de error cometido y del lugar en el que este ha sido detectado.

```
Ejemplo

>>> 1+2

File "<stdin>", line 1
1 + 2)
^

SyntaxError: unmatched ')'
```

En este ejemplo se ha cerrado un paréntesis cuando no había otro abierto previamente, lo cual es incorrecto. Python indica que ha detectado un error de sintaxis (SyntaxError) y señala con el carácter ^ al lugar en el que se encuentra.

En Python los errores se denominan excepciones. Cuando Python es incapaz de analizar una expresión, produce una excepción. Cuando el intérprete interactivo detecta la excepción, nos muestra por pantalla un mensaje de error.

2. Tipos de datos

En Python se encuentran diferentes tipos de datos con sus respectivas características y clasificaciones. A continuación se detallarán los tipos de datos básicos y otros tipos de datos predefinidos por el lenguaje.

Arias (2019) se refiere a los siguientes tipos de datos básicos de Python:

Numéricos

Booleanos

Cadenas de caracteres

Python también define otros tipos de datos, entre los que se encuentran:

Secuencias: list, tuple y range

Conjuntos: set

Mapas: dict

2.1 Conceptos: datos numéricos, booleanos, cadena de caracteres, otros tipos de datos

A

Datos numéricos

Python define tres tipos de datos numéricos: enteros, punto flotante y números complejos.

Números enteros

Se denominan ***int***. Este tipo de dato comprende el conjunto de todos los números enteros, cuyo límite depende de la capacidad de memoria del computador.

Un número de tipo ***int*** se crea a partir de un literal que represente un número entero, o como resultado de una expresión o como una llamada a una función.

```
# Ejemplo

# v es de tipo int y su valor asignado es -3

>>> v = -3
```

```
# m es de tipo int y su valor calculado es 5

>>> m = v + 8

>>>print (m)

5

# z es de tipo int y con la función redondeo es 2

>>> z = round(m/2)

>>> print(z)

2
```

También se pueden representar los números enteros en formato binario, octal o hexadecimal.

Para crear un número entero en binario, se antepone 0b a una secuencia de dígitos 0 y 1.

Para crear un número entero en octal, se antepone 0o a una secuencia de dígitos del 0 al 7.

Para crear un número entero en hexadecimal, se antepone 0x a una secuencia de dígitos del 0 al 9 y de la A la F.

```
Ejemplo

>>> decimal = 8

>>> binario = 0b1101

>>> octal = 0o11

>>> hexadecimal = 0xc

>>> print(decimal)

8

>>> print(binario)
```

```
13

>>> print(octal)

9

>>> print(hexadecimal)

12
```

Números de punto flotante

Se denominan ***float***. Se usa el tipo ***float*** para representar cualquier número real que represente valores de temperaturas, velocidades, estaturas y otras.

```
# Ejemplo

>>> real = 1.1 + 2.2 # real es un float

>>> print(real)

3.3000000000000003 # representación aproximada de 3.3

>>> print(round(real,1))

3.3 # real mostrando únicamente 1 cifra decimal
```

Números complejos

Este tipo de datos en Python se denomina ***complex***.

Los números complejos tienen una parte real y otra imaginaria y cada una de ellas se representa como un ***float***. Los números imaginarios son múltiplos de la unidad imaginaria (la raíz cuadrada de -1).

Para definir un número complejo, se hace así:

<parte_real> + <parte_imaginaria> j

Ejemplo: 4 + 7j

Para acceder a la parte real se usa el atributo real.

Para acceder a la parte imaginaria se usa el atributo *imag*.

```
# Ejemplo

>>> complejo = 5+3j

>>> complejo.real

5.0

>>> complejo.imag

3.0
```

Para tener acceso a los equivalentes complejos del módulo math, se debe usar cmath.

Aritmética de los tipos numéricos

Para todos los tipos numéricos se pueden aplicar las operaciones: suma, resta, producto o división. Para exponentes se usa ** y para la división entera se usa //.

Se permite realizar una operación aritmética con números de distinto tipo. En este caso, el tipo numérico “más pequeño” se convierte al del tipo “más grande”, de forma que el tipo del resultado siempre es el del tipo mayor.

El tipo int es menor que el tipo float, el tipo float es menor que el tipo complex.

Si vamos a sumar un int y un float, el resultado es un float.

Si vamos a sumar un int y un complex, el resultado es un complex.

```
# Ejemplo
```

```
>>> x = 2

>>> a = x**3 # a es 8 elevado a la 3

>>>print(a)

8

>>> b = 31

>>> c = b//4 # c es la parte entera de dividir b entre 4

>>> print(c)

7

>>> g = 31.0

>>> h = d/4 # h es la parte entera de dividir g entre 4

>>>print(h)

7.0

>>> 1 + 2.0

3.0

>>> 2+3j + 5.7

(7.7+3j)
```

B

Datos numéricos

En Python se representan los valores booleanos con ***bool***. Esta clase solo se puede instanciar con dos valores: True para verdadero y False para falso.

Una particularidad del lenguaje es que cualquier variable puede ser usada en un contexto donde se requiera comprobar si algo es verdadero o falso.

Los siguientes objetos/instancias son consideradas falsas:

None

False

El valor cero de cualquier tipo númerico: 0, 0.0, 0j

Secuencias y colecciones vacías: '', (), [], {}, set(), range(0)

```
# Ejemplo
>>> a = False
>>> b = True
>>> Type(a)
<class ' bool ' >
>>> print(a)
false
>>> print(b)
True
>>> c = None
>>> print(c)
None
>>> type(c)
<class ' NoneType ' >
```

C

Datos tipo cadena de caracteres

Salazar (2019) denomina este tipo de dato como *string*. Para crear un *string*, se deben encerrar entre comillas simples o dobles una secuencia de caracteres.

En Python las cadenas de caracteres se representan con ***str***.

Se puede usar comillas simples o dobles. Si en la cadena de caracteres se necesita usar una comilla simple, existen dos opciones: usar comillas dobles para encerrar el *string*, o bien, usar comillas simples, pero anteponer el carácter \ a la comilla simple del interior de la cadena.

```
# Ejemplo

>>> saludo1 = 'Hola "María"'

>>> type(saludo1)

<class ' str ' >

>>> saludo2 = 'Hola \'María\' '

>>> saludo3 = "Hola 'María' "

>>> print(saludo1)

Hola "María"

>>> print(saludo3)

Hola 'María'
```

A diferencia de otros lenguajes, en Python no existe el tipo «caracter». Pero se puede simular con un *string* de un solo caracter:

```
# Ejemplo

>>> caracter1 = 'z'

>>> print(caracter1)

z
```

D

Otros tipos de datos

Adicional a los tipos básicos, se encuentran otros tipos fundamentales de Python denominados secuencias *(list y tuple)*, los conjuntos *(set)* y los mapas *(dict)*.

Pérez (2016) aclara que todos ellos son tipos de datos compuestos y se utilizan para agrupar valores del mismo o diferente tipo.

Las listas son arreglos unidimensionales de elementos donde podemos ingresar cualquier tipo de dato, para acceder a estos datos debemos usar un índice. La posición inicial es la posición 0.

```
# Ejemplo

>>> lista = [3, 4.2, 'SENA', [8,9] ,5] # lista contiene int, real, cadena, list, int

>>> print lista[0] # la posición 0 de la lista contiene el valor 3

>>> print lista[1] # la posición 1 de la lista contiene el valor 4.2

>>> print lista[2] # la posición 2 de la lista contiene la cadena 'SENA'

>>> print lista[3] # la posición 3 de la lista contiene la lista [8,9]

>>> print lista[4] # la posición 4 de la lista contiene el valor 5

>>> print lista[3][0] # la posición 3,0 de la lista contiene 8

>>> print lista[3][1] # la posición 3,1 de la lista contiene 9

>>> print lista[1:3] # las posiciones de la 1 a la 3 contienen [4.2, 'SENA']

>>> print lista[1:4] # las posiciones de la 1 a la 4 contienen [4.2, 'SENA', [8, 9]

>>> print lista[1:5] # las posiciones de la 1 a la 5 contienen [4.2, 'SENA', [8, 9 ],5 ]
```

Las tuplas se representan escribiendo los elementos entre paréntesis y separados por comas. La función len() devuelve el número de elementos de una tupla. Una tupla puede no contener ningún elemento, es decir, puede ser una tupla vacía. Una tupla puede incluir un único elemento seguido de una coma.

```
# Ejemplo

>>> tupla= (8, "b", 4.91)

>>> tupla

(8, 'b', 4.91)

>>> len(tupla)
```

```
3

>>> len(())

0

>>> (3,)

(3,)

>>> len((3,))

1
```

Los conjuntos son una colección no ordenada y sin elementos repetidos. Se definen con la palabra set, seguida de llaves que contienen los elementos separados por comas. Si se desea remover un elemento de un conjunto, se puede usar el método *remove()*.

```
# Ejemplo

>>> frutas = set([ 'mango', 'pera', 'manzana', 'limón' ])

>>> frutas

{'mango', 'manzana', 'pera', 'limón'}

>>> frutas.remove('manzana')

>>> frutas

{'mango', 'pera', 'limón'}
```

Los diccionarios son un tipo de estructuras de datos que permiten guardar un conjunto no ordenado de pares clave-valor, existiendo las claves únicas dentro de un mismo diccionario (es decir, que no pueden tener dos elementos con una misma clave). El diccionario se declara entre los caracteres '{ }' y los elementos se separan por comas (',').

Los diccionarios denominados ***dict*** para Python, son estructuras de datos muy extendidos en otros lenguajes de programación, aunque en otros lenguajes como java se les denominan con distintos nombres como *"Map"*.

```
# Ejemplo

# Defino la variable 'futbolistas' como un diccionario.

futbolistas = dict()

futbolistas = {

13 : "Mina", 21 : "Lucumi",

17 : "Fabra", 11 : "Cuadrado",

9 : "Falcao", 19 : "Muriel",

15 : "Uribe", 10 : "James Rodriguez",

16 : "Lerma", 5 : "Wilmar Barrios",

3 : "Murillo"

}

>>> futbolistas

{13: 'Mina', 21: 'Lucumi', 17: 'Fabra', 11: 'Cuadrado', 9: 'Falcao', 19: 'Muriel', 15: 'Uribe', 10: 'James Rodriguez', 16: 'Lerma', 5: 'Wilmar Barrios', 3: 'Murillo'}

>>> futbolistas[9]

'Falcao'
```

Ejemplo completo de listas, tupla, conjunto y diccionario:

```
# Ejemplo

>>> lista = [1, 2, 3, 8, 9]

>>> tupla = (1, 4, 8, 0, 5)

>>> n=len(tupla)

>>> conjunto = set([1, 3, 1, 4])

>>> diccionario = {'a': 1, 'b': 3, 'z': 8}

>>> print(lista)
```

```
[1, 2, 3, 8, 9]

>>> print(tupla)

(1, 4, 8, 0, 5)

>>> print(“Longitud de la tupla= “,n)

Longitud de la tupla= 5

>>> print(conjunto)

{1, 3, 4}

>>> print(diccionario)

{'a': 1, 'b': 3, 'z': 8}
```

2.2 Identificar el tipo de variable

Guzdial (2013) afirma que existen dos funciones en Python para determinar el tipo de dato que contiene una variable: type() e isinstance():

type()

Recibe como parámetro un objeto y devuelve el tipo del mismo.

isinstance()

Recibe dos parámetros: un objeto y un tipo. Devuelve True si el objeto es del tipo que se pasa como parámetro y False en caso contrario.

```
# Ejemplo

>>> type(5)

<class 'int'>

>>> type(3.14)

<class 'float'>

>>> type('Hola mundo')

<class 'str'>

>>> isinstance(7, float)
```

```
False

>>> isinstance(8, int)

True

>>> isinstance(2, bool)

False

>>> isinstance(False, bool)

True
```

2.3 Conversión de tipos de datos

En algunos casos se requiere convertir el tipo de datos a otro que sea más adecuado. Por ejemplo, si una cadena contiene el valor “10” para poderlo sumar a otra variable tipo entero, se debe convertir la cadena en un dato tipo entero.

Según Cuevas (2017), Python ofrece las siguientes funciones:

str(): devuelve la representación en cadena de caracteres del objeto que se pasa como parámetro.

int(): devuelve un int a partir de un número o secuencia de caracteres.

float(): devuelve un *float* a partir de un número o secuencia de caracteres.

NOTA

Si a las funciones anteriores se les pasa como parámetro un valor inválido, el intérprete mostrará un error.

```
# Ejemplo

>>> edad=”25”

>>> edad = int(edad) + 10 # Convierte edad a int
```

```
>>> edad # edad es un int

35

>>> edad = str(edad) # Convierte edad a str

>>> edad # edad es un str (se muestran las '')

'35'

>>> loat('18.66') # Convierte un str a float

18.66

>>> float('hola') # Convierte un str a float (pero no es válido)

Traceback (most recent call last):

File “<input>” , line 1, in <module>

ValueError: could not convert string to float: 'hola'
```