01-variables

May 25, 2020

1 Variables

Para declarar una variable en Python, debemos crear el nombre y luego asignar su valor.

Ejemplo

```
variable_1 = 1
```

Si queremos mostrar el valor de la variable, debemos imprimir el dato con la función/método print.

```
print(variable_1)
```

En la consola nos mostrará *1*. Es el valor incicial que le asignamos a nuestra variables.

1.1 Ejemplo

[1]:
```
variable_1 = 2
print(variable_1)
```

```
2
```

[2]:
```
variable_2 = "Data Analytics"
print(variable_2)
```

```
Data Analytics
```

En python no es necesario declarar las variables con ningún tipo de datos en particular, inluso se puede cambiar el tipo de después de que se haya establecido.

[3]:
```
variable = 56
variable = 'Data Analytics'
print(variable)
```

```
Data Analytics
```

Las variables de cadena puede ser declaradas mediante comillas simples o dobles.

[4]:
```
variable = "Data Analytics"
print(variable)
variable = 'Tigo Data Analytics'
print(variable)
```

```
Data Analytics
Tigo Data Analytics
```

1.2 Nombre de variables

Una variable puede tener un nombre corto -como a y b- o un nombre más descriptivo -edad, nombre, apellido-

1.2.1 Condiciones para crear variables

- El nombre de una variable debe empezar con una letra o guion bajo
- El nombre de una variable no debe empezar con un número
- El nombre de una variable solo puede contener caracteres alfanuméricos y guion bajo (A-z, 0-9 y _)
- Los nombres de las variables distiguen entre mayúscula y minúscula

Nombre de variables correctas

```
mivariable = "Data Analytics"
mi_variable = "Data Analytics"
_mi_variable_ = "Data Analytics"
miVariable = "Data Analytics"
MIVARIABLE = "Data Analytics"
mivariable1 = "Data Analytics"
```

Nombre de variables no correctas

```
1mivariable = "Data Analytics"
mi-variable = "Data Analytics"
mi variable = "Data Analytics"
mi*variable = "Data Analytics"
```

1.3 Ejemplo

[5]:
```
mivariable = "Data Analytics"
mi_variable = "Data Analytics"
_mi_variable_ = "Data Analytics"
miVariable = "Data Analytics"
MIVARIABLE = "Data Analytics"
mivariable1 = "Data Analytics"
```

Cuando ejecutamos el fragmento de codigo anterior, lo compila exitosamente.

[6]:
```
1mivariable = "Data Analytics"
mi-variable = "Data Analytics"
mi variable = "Data Analytics"
mi*variable = "Data Analytics"
```

```
  File "<ipython-input-6-ff1e106c91a5>", line 1
```

```
    1mivariable = "Data Analytics"
               ^
SyntaxError: invalid syntax
```

Cuando ejecutamos el fragmento de codigo anterior, nos "arroja" un error, ya que los nombres de las variables no son correctos.

1.4 Asignar valor a múltiples variables

Python permite asignar múltiples valores a variable en una línea; también puede asignar el mismo valor a múltiples variables en una línea.

1.5 Ejemplo

[7]:
```
# asignar múltiples valores a varias variables en una sola linea
a, b, c = "Data Analytics", 2, "Tigo"
print(a)
print(b)
print(c)
```

```
Data Analytics
2
Tigo
```

[8]:
```
# asignar el mismo valor a múltiples variables en una sola linea
x = y = z = "Analytics"
print(x)
print(y)
print(z)
```

```
Analytics
Analytics
Analytics
```

02-numeros

May 25, 2020

1 Números

Existe tres (3) tipos de datos numéricos: - int - float - complex

Las variables de los tipos numéricos se crean cuando le asignamos el valor.

1.1 Ejemplo

[1]:
```
a = 45 # int
b = 30.7 # float
c = 56j # complex
print(a)
print(b)
print(c)
```

```
45
30.7
56j
```

Para verificar el tipo de cualquier objeto en python usamos la función ***type()***

1.2 Ejemplo

[2]:
```
a = 45 # int
b = 30.7 # float
c = 56j # complex
print(type(a))
print(type(b))
print(type(c))
```

```
<class 'int'>
<class 'float'>
<class 'complex'>
```

Se puede realizar operaciones matemáticas entre estos tipo de datos

1.3 Ejemplo

[3]:
```
a = 2 + 5.7
print(a)
```

```
7.7
```

[4]:
```
b = 58.5 / 2
print(b)
```

```
29.25
```

[5]:
```
c = 789 * 0.78
print(c)
```

```
615.4200000000001
```

[6]:
```
x = 10 / 4
print(x)
```

```
2.5
```

[7]:
```
y = 69 - 34
print(y)
```

```
35
```

[8]:
```
z = 67.9 - 23.6
print(z)
```

```
44.300000000000004
```

03-operaciones-con-numeros

May 25, 2020

1 Operaciones matemáticas

Los valores numéricos son el resultado de una serie de operaciones matemáticas.

Operador	Nombre	Descripción	Ejemplo
+	Suma	Suma los valores de tipo de datos numéricos	`5 + 4 = 9`
-	Resta	Resta los valores de tipo de datos numéricos	`7 - 4 = 3`
*	Multiplicación	Multiplica los valores de tipo de datos numéricos	`6 * 5 = 30`
/	División	Devuelve el resultado de la división	`46 / 3 = 15.33333333`
//	División entera	Devuelve solo el resultado de la parte entera de la división	`46 // 3 = 15`
%	Módulo	No hace otra cosa que devolver el residuo de la división entre los dos operandos	`7 % 2 = 1`
**	Exponente	Calcula el exponente entre valores de tipo de datos numéricos	`3 ** 2 = 9`
-	Negación	Asigna un valor negativo a un tipo de datos numéricos	`-7`

1.1 Orden de precedencia

1. Exponente (**)
2. Negación (-)
3. Multiplicación (*) - División (/) - División entera (//) - Módulo (%)
4. Suma (+) - Resta (-)

1.2 Ejemplo

[1]:
```
# suma
5 + 4
```

[1]: 9

[2]:
```
# resta
7 - 4
```

[2]: 3

[3]:
```
# multiplicación
6 * 5
```

[3]: 30

[4]:
```
# división
46 / 3
```

[4]: 15.333333333333334

[5]:
```
# división entera
46 // 3
```

[5]: 15

[6]:
```
# módulo
7 % 2
```

[6]: 1

[7]:
```
# exponente
3 ** 2
```

[7]: 9

[8]:
```
# orden de precedencia
3 ** 3 / 56
```

[8]: 0.48214285714285715

[9]:
```
# orden de precedencia
(5 * 56 + 67) / (100 ** -1)
```

[9]: 34700.0

[10]:
```
# negación
-7
```

[10]: -7

04-casting

May 25, 2020

1 Casting

Puede existe que en ocasaciones se desee especificar el tipo en una variable. Esto lo podemos hacer con *casting*. Python es un lenguaje orientado a objeto y como tal, utiliza clases para difinir tipos de datos.

La conversión en python se reliza utuilizando funciones de constructor:

- **int()** Construye un número entero.
- **float()** Construye un número flotante.
- **str()** Construye una cadena a partir de una amplia variedad de tipos de datos, incluidas cadenas, enteros y flotantes.

1.1 Ejemplo

[1]:
```
# convertir a entero
a = int("67") # a será 67
b = int(134.6) # b será 134
c = int(5) # c será 5
# mostar variables
print(a)
print(b)
print(c)
```

```
67
134
5
```

[2]:
```
# convertir a float
a = float("67.78") # a será 67.78
b = float(134) # b será 134.0
c = float(5.6) # c será 5.6
# mostar variables
print(a)
print(b)
print(c)
```

```
67.78
134.0
```

```
5.6
```

[3]:
```python
# convertir a str
a = str("Data Analytics") # a será 'Data Analytics'
b = str(134) # b será '134'
c = str(5.6) # c será '5.6'
# mostar variables
print(a)
print(b)
print(c)
```

```
Data Analytics
134
5.6
```

05-string

May 25, 2020

1 String

Las cadenas en python están rodeados por comillas simples o comillas dobles.

```
'Hola' es lo mismo que "Hola"
```

Se puede mostar la cadena directamente con la función *print()*

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

1.1 Ejemplo

[1]:
```
# cadena de texto
a = "Hola Data Analytics"
print(a)
```

```
Hola Data Analytics
```

[2]:
```
# cadena de texto multilinea
a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.
"""
print(a)
```

```
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.
```

1.2 Concatenar cadena de texto

En python existe 3 forma de contatenar.

1.3 Ejemplo

[3]:
```
# concatenar con el simbolo +
nombre = "Yerson"
edad = 26
concatenacion = "Tu nombre es: " + nombre + " y tienes " + str(edad) + " año(s)"
concatenacion
```

[3]:
```
'Tu nombre es: Yerson y tienes 26 año(s)'
```

[4]:
```
# concatenar con el método format
concatenacion = "Tu nombre es: {0} y tienes: {1} año(s)".format(nombre, edad)
concatenacion
```

[4]:
```
'Tu nombre es: Yerson y tienes: 26 año(s)'
```

[5]:
```
# concatenar con la cadena F
concatenacion = f"Tu nombre es: {nombre} y tienes {edad} año(s)"
concatenacion
```

[5]:
```
'Tu nombre es: Yerson y tienes 26 año(s)'
```

1.4 Métodos en string

Python tiene el comando `dir` que nos ayuda a saber qué método tenemos disponible para `x` tipo de variables. Por ejemplo, queremos saber los método que tiene string usuamos `dir("hola")`

[6]:
```
# mostrar los métodos de un string
dir("Hola")
```

[6]:
```
['__add__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__getnewargs__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
```

```
'__len__',
'__lt__',
'__mod__',
'__mul__',
'__ne__',
'__new__',
'__reduce__',
'__reduce_ex__',
'__repr__',
'__rmod__',
'__rmul__',
'__setattr__',
'__sizeof__',
'__str__',
'__subclasshook__',
'capitalize',
'casefold',
'center',
'count',
'encode',
'endswith',
'expandtabs',
'find',
'format',
'format_map',
'index',
'isalnum',
'isalpha',
'isascii',
'isdecimal',
'isdigit',
'isidentifier',
'islower',
'isnumeric',
'isprintable',
'isspace',
'istitle',
'isupper',
'join',
'ljust',
'lower',
'lstrip',
'maketrans',
'partition',
'replace',
'rfind',
'rindex',
```

```
 'rjust',
 'rpartition',
 'rsplit',
 'rstrip',
 'split',
 'splitlines',
 'startswith',
 'strip',
 'swapcase',
 'title',
 'translate',
 'upper',
 'zfill']
```

¡Podemos observar que existen muchos! Vamos a explicar algunos métodos, "los más usuados"

- **upper()** Convierte el texto a mayúscula
- **lower()** Convierte el texto a minúscula
- **strip()** Elimina cualquier espacio en blanco desde el principio o el final
- **split()** Divide la cadena en subcadenas si encuentra instancias del separador
- **replace()** Reemplaza una cadena con otra cadena.

1.5 Ejemplo

[7]:
```
texto = " Tigo Analytics "
# convertir el texto todo en minúscula
print(f"Cadena en minúscula: {texto.lower()}")
# convertir el texto todo en mayúscula
print(f"Cadena en minúscula: {texto.upper()}")
# eliminar espacio en blanco al principio y/o al final
texto = texto.strip()
print(f"Cadena en minúscula: {texto.strip()}")
# reemplazar la palabra Tigo por Data
texto = texto.replace('Tigo', 'Data')
print(f"Cadena en minúscula: {texto}")
```

```
Cadena en minúscula:  tigo analytics
Cadena en minúscula:  TIGO ANALYTICS
Cadena en minúscula: Tigo Analytics
Cadena en minúscula: Data Analytics
```

[8]:
```
# separar texto por espacios
texto_separado = texto.split(" ")
texto_separado
```

[8]:
```
['Data', 'Analytics']
```

Aquí podemos ver todos las descripciones de los método en un string

11-operadores-asignacion

May 25, 2020

1 Operadores de asignacion

Lo operadores de asignación se utilizan para asignar valores a las variables.

Operador	Descripción	Ejemplo
=	Es el más simple de todos... asigna a la variable del lado izquierdo cualquier variable o resultado del lado derecho	`x = 9`
+=	Suma a la variable del lado izquierdo el valor del lado derecho	`x = 9 x += 10 x ya equivale a 19`
-=	Resta a la variable del lado izquierdo el valor del lado derecho	`x = 10 x -= 5 x ya equivale a 5`
*=	Multiplica a la variable del lado izquierdo el valor del lado derecho	`x = 7 x *= 3 x ya equivale a 21`
/=	Divide a la variable del lado izquierdo el valor del lado derecho	`x = 46 x /= 3 x ya equivale a 15.33333333`
%=	Devuelve el residuo de la división a la variable del lado izquierdo el valor del lado derecho	`x = 7 x %= 3 x ya equivale a 1`
//=	Calcula la división entera a la variable del lado izquierdo el valor del lado derecho.	`x = 46 x //= 3 x ya equivale a 15`
**=	Calcula el exponente a la variable del lado izquierdo el valor del lado derecho.	`x = 3 x **= 3 x ya equivale a 27`

1.1 Ejemplo

[1]:
```
# = igual
x = 9
x
```

[1]: 9

[2]:
```
# += suma
x = 9
x += 10
x
```

[2]: 19

[3]:
```
# -= resta
x = 10
x -= 5
x
```

[3]: 5

[4]:
```
# *= multiplicación
x = 7
x *= 3
x
```

[4]: 21

[5]:
```
# /= división
x = 46
x /= 3
x
```

[5]: 15.333333333333334

[6]:
```
# %= módulo
x = 7
x %= 3
x
```

[6]: 1

[7]:
```
# //= división entera
x = 46
x //= 3
x
```

[7]: 15

[8]:
```
# **= exponente
x = 3
x **= 3
x
```

```
[8]: 27
```

12-operadores-comparacion

May 25, 2020

1 Operadores de Comparación

Los operadores de comparación se utilizan para comparar dos valores.

Operador	Nombre	Descripción	Ejemplo
==	Igual	Evalua que los valores sean iguales para varios tipos de datos.	`5 == 6 False`
!=	No es igual	Evalua si los valores son distintos.	`5 != 6 True`
>	Mayor qué	Evalua si el valor del lado izquierdo es mayor que el valor del lado derecho.	`5 > 6 False`
<	Menor qué	Evalua si el valor del lado izquierdo es menor que el valor del lado derecho.	`5 < 6 True`
>=	Mayor ó igual a	Evalua si el valor del lado izquierdo es mayor o igual que el valor del lado derecho.	`5 >= 6 False`
<=	Menor ó igual a	Evalua si el valor del lado izquierdo es menor o igual que el valor del lado derecho.	`5 <= 6 True`

1.1 Ejemplo

[1]:
```
# igual ==
5 == 6
```

[1]:
```
False
```

[2]:
```
# no es igual
5 != 6
```

[2]:
```
True
```

[3]:
```
# mayor qué
5 > 6
```

[3]: False

[4]:
```
# menor qué
5 < 6
```

[4]: True

[5]:
```
# mayor ó igual a
5 >= 6
```

[5]: False

[6]:
```
# menor ó igual a
5 <= 6
```

[6]: True

13-operadores-logicos

May 25, 2020

1 Operadores lógicos

Los operadores lógicos se utilizan para combinar declaraciones condicionales.

Operador	Descripción	Ejemplo
and	Devuelve True si ambas afirmaciones son verdaderas	`5 < 6 and 7 > 1 True`
or	Devuelve True si una de las declaraciones es verdadera	`5 < 6 and 7 < 1 True`
not	Invierte el resultado, devuelve False si el resultado es verdadero	`not(5 < 6 and 5 == 5) False`

1.1 Ejemplo

[1]:
```
# and
5 < 6 and 7 > 1
```

[1]: True

[2]:
```
# or
5 < 6 or 7 < 1
```

[2]: True

[3]:
```
# not
not(5 < 6 and 5 == 5)
```

[3]: False

[4]:
```
# not
not(5 > 6 and 5 != 5)
```

[4]: True

14-operadores-identidad

May 25, 2020

1 Operadores de Identidad

Los operadores de identidad se utilizan para comparar los objetos, no si son iguales, sino si en realidad son el mismo objeto, con la misma ubicación de memoria.

Operador	Descripción	Ejemplo
is	Devuelve True si ambas variables son el mismo objeto	`x = 5 y = 5 x is y True`
is not	Devuelve True si ambas variables no son el mismo objeto	`x = 5 y = 5 x is not y True`

1.1 Ejemplo

[1]:
```
# is
x = 5
y = 5
x is y
```

[1]:
```
True
```

[2]:
```
# is
x = 5
y = 5.9
x is y
```

[2]:
```
False
```

[3]:
```
# is not
x = 5
y = 5
x is not y
```

[3]:
```
False
```

[4]:
```
# is not
x = 5
y = 5.9
```

```
x is not y
```

[4]: True

[5]:
```
lista_1 = ["manzana", "banano"]
lista_2 = ["manzana", "banano"]
lista_3 = lista_1
```

[6]:
```
# is
lista_1 is lista_2
```

[6]: False

Devuelve `False` porque la `lista_1` y `lista_2` ocupan espacio de memoria diferente.

[7]:
```
lista_3 is lista_1
```

[7]: True

Devuelve `True` porque la `lista_1` y `lista_3` ocupan espacio de memoria igual.

15-operadores-membresia

May 25, 2020

1 Operadores de Membresia

Los operadores de membresía se utilizan para probar si una secuencia se presenta en un objeto.

Operador	Descripción	Ejemplo
in	Devuelve True si una secuencia con el valor especificado está presente en el objeto	`x = [1, 2, 3] 3 in x True`
in not	Devuelve True si una secuencia con el valor especificado no está presente en el objeto	`x = [1, 2, 3] 2 in not x False`

1.1 Ejemplo

[1]:
```
x = [1, 2, 3]
```

[2]:
```
# in
3 in x
```

[2]:
```
True
```

[3]:
```
# not in
2 not in x
```

[3]:
```
False
```

[4]:
```
lista = ["manzana", "banano", "sandia"]
```

[5]:
```
# in
"mango" in lista
```

[5]:
```
False
```

[6]:
```
# not in
"mango" not in lista
```

[6]:
```
True
```