variables

January 28, 2020

1 Variables

1.1 Asignacion de Variables

Pensemos que una variable es un nombre asignado a un objeto en particular.

Una variable se crea asignando un valor al mismo, a fin de utilizarlo mas adelante. La asignacion se hace con el symbolo =.

[1]:
```
# Guardemos una variable
frase = 'Hola Mundo'
print(frase)
```

```
Hola Mundo
```

[2]:
```
# En una sesion REPL, podemos utilizar la variable sin el print()
frase
```

[2]:
```
'Hola Mundo'
```

[3]:
```
# Podemos cambiar el valor de la variable y volver a utilizarla
frase = 'Hola Chiriqui!'
print(frase)
```

```
Hola Chiriqui!
```

[4]:
```
# Tambien podemos realizar una asignacion en cadena (el mismo valor a muchas␣
 ↪variables)
a = b = c = 100
print(a, b, c)
```

```
100 100 100
```

[5]:
```
# Podemos reasignar una variable de un numero a un string
a = 'hola'
print(a)
```

```
hola
```

1.2 Referencias a Objetos

Que sucede cuando creamos una asignacion a una variable?

Toda cosa o dato en Python es un objeto de un **tipo** o **clase**

```
# Por ejemplo
print(300)
```

El interpretador hace lo siguiente: 1. Crea un objeto de tipo `integer` 2. Le da un valor de `300` 3. Muestra el valor en la consola

[6]:
```
# Podemos ver que es de tipo integer utilizando el metodo incorporado␣
↪(built-in) type()
type(300)
```

[6]:
```
int
```

Una variable en Python es nombre que sirve de referencia o puntero a un objeto. Cuando un objeto es asignado a una variable, podemos hacer referencia al mismo utilizando el nombre, no obstante, el dato o valor sigue contenido dentro del objeto.

Por ejemplo:

```
n = 300
```

Esta asignacion crea un objeto de tipo entero o `integer` con un valor de `300` y asigna la variable `n` a que apunte a ese objeto.

[7]:
```
n = 300
print(n, type(n))
```

```
300 <class 'int'>
```

[8]:
```
# Consideremos la siguiente declaracion
m = n
```

Que sucede cuando ejecutamos esa declaracion?

Python no crea otro objeto, simplemente crea un nombre simbolico o referencia denominada `m` que apunta al mismo objeto al que apunta `n`

[9]:
```
# ahora, hagamos esto
m = 400
```

Python crea un nuevo objeto de tipo `integer` con un valor de `400` y `m` se convierte en su referencia.

[10]:
```
# supongamos esta declaracion se ejecuta despues
n = 'foo'
```

Python crea un objeto de tipo `string` con un valor `foo` y `n` se convierte en su referencia.

Observemos que ya no existe una referencia al objeto de tipo `integer` con el valor `300`. Es huerfano, toda vez que no hay manera de acceder al mismo. Esto trae a colacion el **tiempo de vida de un**

objeto. La vida de un objeto empieza cuando es creado y donde por lo menos existe una referencia al mismo. Durante el tiempo de vida de un objeto, pueden existir referencias adicionales al mismo, como vimos anteriormente, y esas referencias tambien pueden desaparecer. Es importante destacar que un objeto se mantiene vivo, siempre y cuando exista una referencia que apunte al mismo.

No es posible acceder a un objeto cuando el numero de referencias al mismo es `0`. Esto es de suma importancia, toda vez que Python automaticamente reclama el espacio de memoria asignado al mismo para que pueda ser utilizado por alguna otra cosa. Este proceso se llama `garbage collection` o `recoleccion de basura`.

1.3 Identidad de un objeto

En Python, cada vez que un objeto es creado, se le da un numero que lo identifique. No existen dos objetos con el mismo numero de identificacion durante el tiempo de vida de ambos. En cierto sentido, es el equivalente de nuestra cedula. No existen (por lo menos no deberian existir) distintas personas con el mismo numero de cedula.

Cuando el numero de referencias de un objeto es `0`, se activa el recolector de basura, y su numero de identificacion se torna disponible para ser usado nuevamente.

Python tiene un metodo incorporado denominado `id()` que retorna el numero de identificacion de un objeto. Utilizando este metodo, podemos verificar que dos variables apuntan al mismo objeto.

[11]:
```
n = m = 300
print(id(n), id(m))
```

```
139935005885200 139935005885200
```

[12]:
```
# cambiemos el valor de uno
m = 400
print(id(n), id(m))
```

```
139935005885200 139935005885104
```

1.4 Nombres de Variables

Oficialmente, los nombres de variables en Python pueden ser de cualquier numero de caracteres y pueden consistir de letras mayusculas, minusculas, digitos e incluso el guion bajo `_`. Sin embargo, el primer caracter de una variable no puede ser un digito.

[13]:
```
# todo los siguientes nombres son validos
nombre = 'juan'
edad = 55
tiene_cedula = True
print(nombre, edad, tiene_cedula)
```

```
juan 55 True
```

[14]:
```
# este nombre no es valido
123 = 'no es valido'
```

```
print(123)
```

```
  File "<ipython-input-14-39be306f3ddc>", line 2
    123 = 'no es valido'
    ^
SyntaxError: cannot assign to literal
```

[15]:
```
# mayusculas y minusculas no son lo mismo
edad = 50
Edad = 60
print(edad, Edad)
```

```
50 60
```

Existen 3 principales convenciones de escribir nombrse de variables con mas de una palabra:

1. Camello : `primeroSegundo`
2. Pascal: `PrimeroSegundo`
3. Culebra `primero_segundo`

Es importante destacar que la guia oficial de estilo de Python, conocida como **PEP 8**, contiene Convenciones de Nombres y hace las siguientes recomendaciones:

1. La convencion Culebra debe ser utilizado para funciones / metodos y nombre de variables
2. La convencion Pascal debe ser utilizado para nombre de Clases (lo veremos mas adelante)

1.5 Palabras Reservadas (Keywords)

Existen varias palabras reservadas que no pueden ser utilizadas como nombres

[16]:
```
# en Python 3.8 existen 34 palabras reservadas
import keyword
for index, key in enumerate(keyword.kwlist):
    print(index, key)
```

```
0 False
1 None
2 True
3 and
4 as
5 assert
6 async
7 await
8 break
9 class
10 continue
11 def
```

```
12 del
13 elif
14 else
15 except
16 finally
17 for
18 from
19 global
20 if
21 import
22 in
23 is
24 lambda
25 nonlocal
26 not
27 or
28 pass
29 raise
30 return
31 try
32 while
33 with
34 yield
```

1.6 Prueba de conocimiento

Cual de las siguientes declaraciones le asigna un valor `100` a la variable `x`? 1. `x <- 100` 2. `x = 100` 3. `x << 100` 4. `let x = 100` 5. `x := 100`

En Python, una variable puede ser asignada un valor de un tipo, y luego ser asignada a un valor de otro tipo?

```
n = 'hola'
n = 1
```

Considera la proxima declaracion:

```
n = 300
m = n
```

Cuantos objetos y cuantas referencias son creadas?

1. Un objeto, una referencia
2. Un objeto, dos referencias
3. Dos objetos, dos referencias
4. Dos objetos, una referencia

Cual es el metodo que retorna el numero de identificacion asignado a un objeto?

1. `identity()`
2. `ref()`
3. `id()`

4. `refnum()`

Cual de las siguientes constituyen nombres validos en Python? 1. `return` 2. `direccion_casa` 3. `calle13` 4. `Edad` 5. `4ruedas` 6. `version1.0`

Te encuentras las siguientes declaraciones:

```
empleadonumero = 4398
    .
    .
    .
EmpleadoNumero = 4398
    .
    .
    .
empleadoNumero = 4398
```

Estas declaraciones hacen referencia a la misma variable o a variables distintas? 1. Mismas variables 2. Distintas variables

Cuales de los siguientes estilos es recomendado para variables de mas de un nombre? 1. `distanciaABoquete` 2. `DistanciaABoquete` 3. `distancia_a_boquete`

Cuales de las proximas palabras estan reservadas? 1. `None` 2. `class` 3. `and` 4. `default` 5. `goto`