10_python_variables

November 27, 2023

1 Python Variables

En programmation les variables sont des espaces mémoires qui permettent de stocker des valeurs. Ces valeurs peuvent être de différents types: entiers, réels, chaînes de caractères, booléens, etc.

[4]:
```
number: int = 10
print(number)
```

```
10
```

Ici `number` est une variable de type entier qui contient la valeur 10

2 Assignation de variables

Pour assigner une valeur à une variable on utilise le symbole `=`. Par exemple:

[ ]:
```
# assign value to site_name variable
site_name = 'programiz.pro'

print(site_name)
n = 10
n *= 12 # <==> n = n * 12
print(n)
```

Dans l'exemple ci-dessus, on a assigné la valeur `programiz.pro` à la variable `site_name`. On peut ensuite utiliser cette variable dans notre code, ici on l'a affiché.

> **Note:** En Python, les variables sont sensibles à la casse, c'est-à-dire que `a` et `A` sont deux variables différentes.

> **Note:** En Python, le type des variables est déterminé dynamiquement, c'est-à-dire que le type de la variable est déterminé à l'exécution du programme. Par exemple, dans l'exemple ci-dessus, on a assigné la valeur `programiz.pro` à la variable `site_name`, donc le type de la variable `site_name` est `str` (chaîne de caractères).

3 Changement de valeur d'une variable Python

On peut changer la valeur d'une variable Python en lui assignant une nouvelle valeur. Par exemple:

[ ]:
```
site_name = 'facebook.com'
print(site_name)

# assigning a new value to site_name
site_name = 12
print(site_name)
```

Ici, on a changé la valeur de la variable `site_name` en lui assignant la valeur `facebook.com` et ensuite la valeure `apple.com`.

4 Assignation multiple

On peut assigner plusieurs variables en une seule ligne. Par exemple:

[ ]:
```
a, b, c = 5, 3.2, 'Hello'
a, b = b, a

print(a)
print(b)
print(c)
```

On peut aussi assigner une même valeur à plusieurs variables en une seule ligne. Par exemple:

[ ]:
```
site1 = site2 = 'apple.com'

print((site1))
print((site2))
```

5 Règles pour nommer les variables Python

- Les constantes et les variables en python doivent commencer par une lettre ou un underscore `_`.
- Le reste du nom de la variable peut contenir des lettres, des chiffres et des underscores `_`.

```
snake_case
MACRO_CASE
camelCase
CapWords
kebab-case
```

- Utilisez des nom de variables qui font sense, `voyelles` est un meilleur nom que `v`. Et essayer de coder en anglais.
- Si vous voulez utiliser une variables a 2 ou n mots utilisez des underscores pour séparer les mots

```
my_name
current_salary
```

- Evitez d'utiliser des `keywords` comme nom de variables

- Les noms de variables sont sensibles à la casse, c'est-à-dire que `a` et `A` sont deux variables différentes.

[ ]:
```
num = 5
Num = 55
print(num)
print(Num)
print(num)
```

6 Constantes en Python

Les constantes sont des variables dont la valeur ne peut pas être modifiée. En Python, on peut créer des constantes en utilisant des lettres majuscules. Par exemple:

[ ]:
```
# declare constants
PI = 3.14
GRAVITY = 9.8

print(PI, GRAVITY)
```

> **Note:** En Python, les constantes sont modifiables, non protégée comme dans d'autres langages de programmation. Par exemple, dans l'exemple ci-dessus, on peut changer la valeur de la constante `PI` en faisant `PI = 3.1415`. Le fait d'utiliser des lettres majuscules pour les constantes est une convention de codage pour indiquer aux autres programmeurs que c'est une constante. Donc, il est préférable de ne pas changer la valeur d'une constante.

7 Python literals

Les littéraux sont des données dont la valeur est fixe. En Python, il existe différents types de littéraux:

[ ]:
```
site_name = 'apple.com'
print(type(site_name))
site_name = "apple.com"
print(type(site_name))
```

Dans l'example ci-dessus `site_name` est la variable et `apple.com` est le littéral.

8 Python Numeric Literals

Les littéraux numériques sont des données dont la valeur est un nombre.

Type	Example	Remarque
Décimal	5, 10, -68	Integers
Binaire	0b101, 0b11	Commence avec `0b`

Type	Example	Remarque
Octal	0o13	Commence avec `0o`
Hexadécimal	0x9, 0x11	Commence avec `0x`
Float	0.0, 15.20, -32.54	Nombre à virgule flottante
Complex	3+14j, 4+0j, -6+7j	Nombre sous la forme `a + bj` ou `a` est la partie réelle et `bla` partie imaginaire

[ ]:
```
a = 0b1010 #Binary Literals
b = 10 #Decimal Literal
c = 0o12 #Octal Literal
d = 0xA #Hexadecimal Literal

#Float Literal
float_1 = 10.5
float_2 = 1.5e2 # <==> 1.5 * 10^2 <==> 150

#Complex Literal
x = -3+14j

print(a, b, c, d)
print(float_1, float_2)
print(x, x.real, x.imag)
```

[ ]:
```
print(f"a = {a} | type de a={type(a)}")
print(f"b = {b} | type de b={type(b)}")
print(f"c = {c} | type de c={type(c)}")
print(f"d = {d} | type de d={type(d)}")
print(f"float_1 = {float_1} | type de float_1={type(float_1)}")
print(f"float_2 = {float_2} | type de float_2={type(float_2)}")
print(f"x = {x} | type de x={type(x)}")
print(f"x.real = {x.real} | type de x.real={type(x.real)}")
print(f"x.imag = {x.imag} | type de x.imag={type(x.imag)}")
```

9 Python Boolean Literals

Les littéraux booléens sont des données dont la valeur est soit `True` soit `False`. > **Note:** En Python, `True` et `False` sont des mots-clés et ils doivent être écrits en majuscules.

[ ]:
```
flag = (12 != 13)
```

```
print(flag)
if flag:
    print("flag is true")
```

Ici, `True` est un littéral booléen et est assigné à la variable `flag`.

10 String et char literals

Les caractères littéraux sont des données unicode dont la valeur est un caractère. En Python, on peut définir une chaîne de caractères en utilisant des guillemets simples ou doubles. Par exemple:

[ ]:
```
some_char = 'a"b'
print(type(some_char))

some_char2 = "a\"b"
print((some_char2))
```

Ici, `a` est un caractère littéral et est assigné au deux variables `some_char` et `some_char2` en utilisant une fois les simples guillemets et une fois les doubles guillemets.

De façon similaire on peut construire les strings comme étant des listes de caractères. Par exemple:

[ ]:
```
some_string = 'Python is fun'
print(some_string)

some_string2 = "Python is fun"
print(some_string2)

quote_in_strings = 'Python\'s syntax is easy to learn'
print(quote_in_strings)

quote_in_strings2 = "Python\\'s syntax is easy to learn"
print(quote_in_strings2)
```

Ici, `'Python is fun'` est un littéral de chaîne de caractères et est assigné à la variable `some_string`.

> **Note**: On peut mettre des `single quotes (')` à l'intérieur des `double quotes (")` et vice versa.

11 Python Special Literals

Python a un littéral spécial appelé `None`. On l'utilise souvent pour représenter la valeur nulle ou vide. Par exemple:

[ ]:
```
value = None

print(type(value))
```

Ici, nous recevons la valeur `None`, cela signifie que la variable `value` n'as pas de valeurs.

12 Literals Collections

Python a 4 types de collections littérales:

[ ]:
```
# list literal
fruits = ["apple", "mango", "orange", 1, 1.2, 0b101, 'salut', "apple"]
print(fruits)

# tuple literal
numbers = (1, 2, 3)
print(numbers)

# dictionary literal
utilisateur = {'prenom':'Antonio', 'nom':'Pisanello', 'age':28, "hobbies":
 ↪["programmer", "faire du sport"]}
utilisateur2 = {'prenom':'Lamine', 'nom':'Sakho', 'age':'28', "hobby":
 ↪["programmer", "faire du sport"]}
print(utilisateur)
print(utilisateur2)

# set literal
vowels = {'a', 'e', 'i' , 'o', 'u', 'e'}
print('e' in vowels)
print(vowels)
```

Dans l'example ci-dessus, nous avons crée une **liste** de `fruits`, un **tuple** de `chiffres`, un **Dictionnaire** d'`alphabet` et un **set** de `voyelles`.

> **Note:** Les listes et les tuples sont des collections ordonnées, donc les éléments sont accessibles par leur index, tandis que les dictionnaires et les sets sont des collections non ordonnées, donc les éléments ne sont pas accessibles par leur index. (Nous verrons plus loin ce que sont les listes, les tuples, les dictionnaires et les sets.)