Variables, affectations

Chap. 3

Table des matières

1 Variables

Une variable est une zone de la mémoire repérée par un **identificateur**. Cet identificateur permet de *modifier ou de faire appel au contenu de cette zone de la mémoire* lors du déroulement du programme.

Remarque. Le langage Python est *sensible à la casse*. `ma_variable`, `maVariable` et `mavariable` sont donc *trois identificateurs différents* (préférer la première écriture).

Convention.

- Le premier caractère de l'identificateur doit être une lettre ou un blanc souligné (_).
- Les caractères suivant peut être alphanumériques ou des blanc soulignés (_).
- Les majuscules et les minuscules ne sont pas confondues.
- Le tableau 1, contient les mots-clés du langage Python. *Il est impossible de choisir un identificateur correspondant à l'un de ces mots-clés.*

and	as	assert	break	class	continue	def	del	elif
else	except	exec	finally	for	from	global	if	import
in	is	lambda	nonlocal	not	or	pass	print	raise
return	try	while	with	yield	None	True	False	

Tableau 1. Mots-clés réservés en Python

2 Déclaration et affectation des variables

2.1 Vocabulaire

La **déclaration** d'une variable consiste à réserver un emplacement dans la mémoire pour une utilisation ultérieure.

Note. La déclaration d'une variable doit donc précéder son utilisation.

L'**affectation** est une opération au cours de laquelle on copie dans une variable une *valeur littérale*, le *contenu d'une autre variable* ou le *résultat d'un calcul*.

2.2 Affectation d'une valeur à une variable

2.2.1 Affectation simple

En Python, il est impossible de déclarer une variable sans l'utiliser. *La variable est créée lorsqu'on lui affecte une donnée*. On utilise à cet effet l'opérateur `=`.

Python est un **langage à typage dynamique** : *il n'est pas nécessaire de déclarer le type de la valeur référencée par la variable*.

En Python

```
a = 3
```

En Java, C, C++, etc.

```
int a;
a = 3;
```

Remarque. On peut noter dès à présent qu'une *affectation en Python ne consiste pas au stockage d'une valeur dans une variable*, contrairement à ce qui se passe dans d'autres langages de programmation. En Python, les objets (entiers, flottants, chaînes de caractères, ...) sont **référencés** : *lors d'une affectation, c'est une référence à un objet (et non une valeur) qui est définie*, que l'objet vienne d'être créé ou qu'il s'agisse d'un objet préexistant.

Une variable est, en Python, une étiquette qui permet d'accéder à une valeur.

```
>>> un_entier = 12
>>> une_chaine = 'cartable'
>>> un_flottant = -2 + 3.12 + 17
```

2.2.2 Affectations multiples

```
>>> x = y = z = 1
>>> x
```

```
1
>>> y
1
>>> z
1
```

Remarque. Pour bien comprendre comment fonctionne l'affectation multiple, ci-dessus, **lire l'instruction de la droite vers la gauche**. Ainsi, on affecte à la variable `z` l'entier 1, à la variable `y` la valeur référencée par la variable `z`, à la variable `x` la valeur référencée par la variable `y`. L'instruction précédente est donc équivalente à l'enchaînement :

```
>>> z = 1
>>> y = z
>>> x = y
```

2.2.3 Affectation de « tuples»

```
>>> a, b, c = 1, 2, 3
>>> a
1
>>> b
2
>>> c
3
```

Remarque. Le fonctionnement de cette affectation sera détaillé dans un prochain chapitre.

2.3 Opérateurs d'affectation

Le langage Python possède un certain nombre de raccourcis permettant de très rapidement écrire la mise à jour de la valeur d'une variable à partir d'une expression faisant elle même référence au contenu de la variable. *Ici aussi, il est nécessaire de lire l'instruction de la droite vers la gauche.*

Opérateur	Expression	Description
`=`	`variable = expression`	
`+=`	`variable += expression`	`variable = variable + expression`
`-=`	`variable -= expression`	`variable = variable - expression`
`*=`	`variable *= expression`	`variable = variable * expression`
`/=`	`variable /= expression`	`variable = variable / expression`
`//=`	`variable //= expression`	`variable = variable // expression`
`%=`	`variable %= expression`	`variable = variable % expression`

Tableau 2. Opérateurs d'affectation

3 Formatage des chaînes de caractères

Comme nous l'avons vu dans le chapitre 2, une chaîne de caractère est un type pré-défini incontournable du langage Python.

```
>>> type("abcd")
<class 'str'>
```

```
>>> type('abcd')
<class 'str'>
>>> type("Python, c'est super !")
<class 'str'>
>>> type('Python, c'est super !')
    File "<pyshell>", line 1
       type('Python, c'est super !')
                       ^
     SyntaxError: invalid syntax
>>> type("2")
<class 'str'>
>>> type("a")
<class 'str'>
>>> type('-2.3')
<class 'str'>
```

Il faut donc être capable de manipuler ces chaînes de caractères, et en particulier d'y incorporer le résultat de l'évalution d'expressions[1].

La méthode `chaine.format()` permet d'insérer u*n littéral ou le résultat de l'évaluation d'une expression* dans une chaîne de caractères.

Le type du littéral ou du résultat de l'évaluation est déterminé automatiquement ; la conversion vers le type `str` est alors effectuée.

```
>>> '{}, {}, {}'.format('a', 'b', 'c')
'a, b, c'
>>> '{0}, {1}, {2}'.format('a', 'b', 'c')
'a, b, c'
>>> "La somme de 1 + 2 est {}".format(1 + 2)
'La somme de 1 + 2 est 3'
>>> "La somme de {1} + {2} est {0}".format(2 + 3, 2, 3)
'La somme de 2 + 3 est 5'
>>> '{0}{1}{0}'.format('abra', 'cad')
'abracadabra'
>>> '{:f}; {:f}'.format(3.14, -3.14) # :f est un marqueur
'3.140000; -3.140000'
```

Certaines mises en forme nécessitent de préciser le type de la donnée et la façon de l'afficher. Le **marqueur de formatage :** permet d'introduire des **drapeaux** :

- `s` indique que la donnée à insérer est une chaîne. *C'est le drapeau implicite et il est inutile de le préciser* ;
- le drapeau `d` attend un nombre et le convertit en entier ;
- les drapeaux `f`, `g` et `e` attendent des flottants. `g` est le plus général, il correspond à `f` si le nombre n'est pas trop grand, à la notation avec puissances de 10 `e`, si l'écriture du nombre prend beaucoup de place ;
- ...

1. Nous en apprendrons bien plus sur les chaînes de caractères dans un futur chapitre.

Pour finir, il est possible d'indiquer *le nombre minimal de chiffres à utiliser (ils n'ont pas besoin d'être visibles, seule la place nécessaire pour le affichage compte) et le nombre de chiffres après la virgule à afficher.*

```
>>> "avant{0:2f}".format(3333.3333) # au moins 2 chiffres avant la virgule
'avant3333.333300'                  # notation décimale
>>> "avant{0:20f}".format(3333.3333) # au moins 20 chiffres avant la virgule
'avant 3333.333300'
>>> "avant{0:2.2f}".format(3333.3333) # au moins 2 chiffres avant et 2 chiffres après
'avant3333.33'
>>> "avant{0:2.2f}".format(3333.3383) # au moins 2 chiffres avant et 2 chiffres après
'avant3333.34'
>>> "avant{0:2.2e}".format(3333.3333) # au moins 2 chiffres avant et 2 chiffres après
'avant3.33e+03'                       # notation scientifique
```

4 Exercices du chapitre

Exercice 1. Variables

Quelles sont les valeurs des nombres contenus dans les variables `prix`, `tva` et `total` après exécution de chacune des instructions suivantes ?

```
>>> prix = 20
>>> tva = 18.6
>>> total = prix + prix * tva /100
```

Exercice 2. Variables

Quelles sont les valeurs des nombres contenus dans les variables `prix`, `tva` après exécution de chacune des instructions suivantes ?

```
>>> prix = 20
>>> tva = 18.6
>>> prix = prix + prix * tva /100
```

Exercice 3.

Dans chacun des cas, quelles sont les valeurs des variables `a` et `b` après l'exécution de chacune des instructions suivantes ?

```
>>> a = 5
>>> b = 7
>>> b = a
>>> a = b
```

```
>>> a = 5
>>> b = 7
>>> a = b
>>> b = a
```

Exercice 4. Échange de valeurs

Laquelle des instructions suivantes permet d'échanger les valeurs des deux variables `a` et `b` ?

```
>>> a = b
>>> b = a
```

```
>>> t = a
>>> a = b
>>> b = t
```

```
>>> t = a
>>> b = a
>>> t = b
```

Exercice 5. Échange de valeurs

Soit trois variables `a`, `b` et `c`. Écrire les instructions permutant les valeurs, de sorte que la valeur de `a` passe dans `b`, celle de `b` dans `c` et celle de `c` dans `a`. N'utiliser qu'une (et une seule) variable entière supplémentaire, nommée `tmp`.

Exercice 6. Conversion francs euros

Écrire une fonction qui convertit des francs en euros (1 euro correspond à 6,55957 francs).

La spécification de la fonction est :

```
def conversion_franc_euro(francs: float) -> str:
    """ Conversion d'une somme en francs en euros.
    Le message retourné donne les deux valeurs.

    >>> conversion_franc_euro(100)
    '100 francs correspondent à 15.244901723741037 euros.'
    """
```

Exercice 7. Message d'accueil

Écrire une fonction qui, à partir d'un nom, d'un prénom et d'une date de naissance retourne un message d'accueil incluant l'age de la personne.

La spécification de la fonction est :

```
def salutation(nom: str, prenom: str, naissance: int) -> str:
    """ À partir des nom, prénom et date de naissance, retourne un
    message d'accueil indiquant l'âge.

    >>> salutation("Dupond", "Patrick", 1961)
    'Bonjour Patrick Dupond, vous avez 58 ans.'
    """
```

Exercice 8. Nombre aléatoire

Écrire une fonction qui, à partir de deux ages limites, retourne une chaîne de caractères indiquant l'age de la personne qui appelle cette fonction, choisi aléatoirement.

La spécification de la fonction est :

```
def age_aleatoire(age_min: int, age_max: int) -> str:
    """ À partir de deux ages limites, retourne un message contenant
    un age déterminé aléatoirement.

    >>> age_aleatoire(20, 100)
    'Votre age est : 85 ans !'
    """
```