Variables, affectation et trace d'exécution

Résumé

Un programme est une suite d'instructions que Python exécute une par une, de la première ligne à la dernière. Une **variable** est un nom associé à une valeur, et l'**affectation** (le signe =) associe une valeur à ce nom. Pour comprendre ce que fait un programme sans l'exécuter, on suit ses instructions dans l'ordre et on note la valeur de chaque variable après chaque ligne : c'est le **tableau d'état**, appelé aussi trace d'exécution. Savoir le remplir sans erreur est la base de tout le reste : conditions, boucles et fonctions se comprennent de la même façon.

① Mémo

Définition 1 (Séquence d'instructions)

Python exécute les instructions **dans l'ordre où elles sont écrites**, une par ligne, de haut en bas. Une instruction ne connaît que ce qui a été fait avant elle : une variable doit avoir reçu une valeur avant d'être utilisée.

Définition 2 (Variable et affectation)

L'instruction `x = 5` se lit « `x` reçoit 5 ». Python procède en deux temps :

1. il calcule la valeur de l'expression à droite du signe = ;
2. il associe cette valeur au nom écrit à gauche.

Le signe = n'est donc pas une égalité mathématique : à gauche, il y a toujours un nom de variable.

Réaffectation

Une variable peut recevoir une nouvelle valeur ; l'ancienne est alors perdue.

- `x = x + 1` : Python calcule `x + 1` avec l'ancienne valeur de `x`, puis range le résultat dans `x`.
- `x += 1` est une écriture abrégée de la même instruction.

Le tableau d'état

Pour suivre un programme à la main :

1. une colonne par variable, une ligne par instruction ;
2. sur chaque ligne, on écrit la valeur de chaque variable **après** l'instruction ;
3. une case vide signifie que la variable n'existe pas encore.

Échanger deux valeurs

- Avec une variable temporaire : `temp = a`, puis `a = b`, puis `b = temp`.
- Avec l'affectation multiple : `a, b = b, a`. Python calcule d'abord les deux valeurs de droite, puis les range.

Pièges fréquents

- `b = a` copie la valeur de `a` dans `b` : modifier `a` ensuite ne change pas `b` (pour un nombre ou une chaîne).
- Écrire `x + 1` seul calcule une valeur mais ne modifie pas `x`.
- Deux instructions échangées dans l'ordre donnent en général un autre résultat.

Erreurs classiques

Code erroné	Code correct	Explication
`5 = x` → `SyntaxError`	`x = 5`	Le nom de la variable s'écrit à gauche du signe =, la valeur à droite.
`print(total)` `total = 0` → `NameError`	`total = 0` `print(total)`	Une instruction ne voit que ce qui précède : la variable doit exister avant d'être lue.
`a = b` `b = a`	`a, b = b, a`	Après la première ligne, l'ancienne valeur de `a` est perdue : les deux variables valent `b`.
`x + 1`	`x = x + 1`	Le calcul seul ne change pas la variable : il faut ranger le résultat.

② Exemples

Programme 1 · Un programme de quatre lignes

```
a = 3
b = a + 2
a = b * 2
b = b - a
print(a, b)    # 10 -5
```

Tableau d'état :

Ligne	Instruction	a	b
1	`a = 3`	3	
2	`b = a + 2`	3	5
3	`a = b * 2`	10	5
4	`b = b - a`	10	−5

Programme 2 · Un score qui augmente

```
score = 0
score = score + 10
score = score + 5
print(score)    # 15
```

Programme 3 · Échange avec une variable temporaire

```
a = 5
b = 3
temp = a    # temp vaut 5
a = b       # a vaut 3
b = temp    # b vaut 5
print(a, b)    # 3 5
```

Programme 4 · Des noms qui disent ce qu'ils contiennent

```
prix_ht = 40
taux_tva = 0.2
prix_ttc = prix_ht * (1 + taux_tva)
print(prix_ttc)    # 48.0
```

③ Exercices

Exercice 1 *Prédire un affichage* Sans exécuter le programme, dire ce qu'il affiche, puis vérifier en Python.

```
x = 4
y = x * 3
x = x + 1
print(x, y)
```

Exercice 2 *Remplir un tableau d'état* Remplir le tableau d'état du programme suivant, puis donner l'affichage final.

```
n = 7
m = 2
n = n - m
m = m * n
n = n + m
m = n % m
print(n, m)
```

Exercice 3 *Rotation de trois valeurs* Les variables `a`, `b` et `c` valent respectivement 1, 2 et 3. Écrire des instructions pour qu'à la fin `a` vaille 2, `b` vaille 3 et `c` vaille 1 :

1. en utilisant une seule variable temporaire ;
2. en une seule ligne, avec l'affectation multiple.

Exercice 4 *Retrouver le programme* Écrire un programme de quatre affectations, sans autre nombre que 2, qui produit exactement ce tableau d'état.

Ligne	x	y
1	2	
2	2	4
3	8	4
4	8	12

Solutions des exercices

Corrigé de l'exercice 1. Le programme affiche `5 12`. À la ligne 2, `y` reçoit $4 \times 3 = 12$. La ligne 3 change `x` en 5, mais `y` garde la valeur 12 calculée avant : elle ne se recalcule pas.

Corrigé de l'exercice 2.

Ligne	Instruction	`n`	`m`
1	`n = 7`	7	
2	`m = 2`	7	2
3	`n = n - m`	5	2
4	`m = m * n`	5	10
5	`n = n + m`	15	10
6	`m = n % m`	15	5

Le programme affiche `15 5`. À la ligne 6, `n % m` est le reste de la division de 15 par 10.

Corrigé de l'exercice 3.

1. On sauvegarde la valeur de `a`, qui sera écrasée la première :

```
a, b, c = 1, 2, 3
temp = a    # temp vaut 1
a = b       # a vaut 2
b = c       # b vaut 3
c = temp    # c vaut 1
print(a, b, c)    # 2 3 1
```

2. `a, b, c = b, c, a` : les trois valeurs de droite sont calculées avant toute affectation.

Corrigé de l'exercice 4.

```
x = 2
y = x * 2     # 4
x = x * y     # 2 × 4 = 8
y = x + y     # 8 + 4 = 12
print(x, y)    # 8 12
```

Méthode : à chaque ligne, une seule variable change ; on cherche comment obtenir sa nouvelle valeur à partir des valeurs de la ligne précédente. D'autres réponses sont possibles, par exemple `y = x + x` à la ligne 2.