Chapitre 3 : variable, instruction et programme

"Objectifs"

À la fin du chapitre, on doit savoir :

- représenter une variable comme une association entre un nom et une valeur
- distinguer instruction et expression
- tracer l'évolution de l'état d'un programme dans un tableau
- intepréter un message d'erreur `NameError`

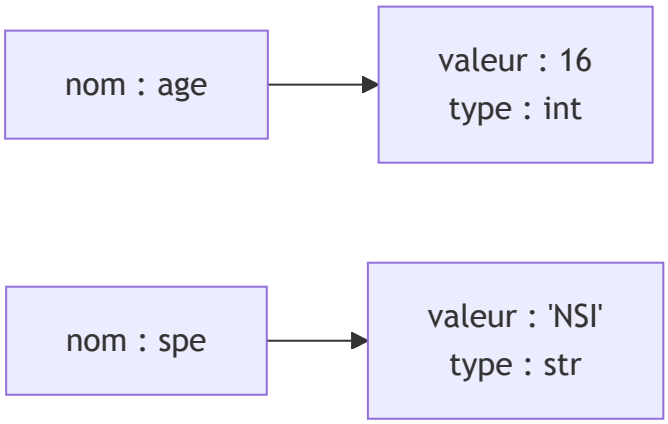
Variable, affectation

"Définition 1"

Une **variable** est un nom qui permet de désigner une valeur pendant l'exécution d'un programme.

La valeur désignée par une variable peut changer au cours de l'exécution.

Par exemple, on peut représenter ainsi l'association entre la variable `age` et la valeur 16 et la variable `spe` et la valeur `"NSI"`.



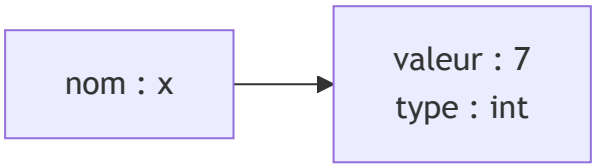
"Définition 2"

Une **affectation** est une instruction qui associe une valeur à un nom de variable.

En Python, l'affectation s'écrit avec le symbole `=`.

"Exemple 1"

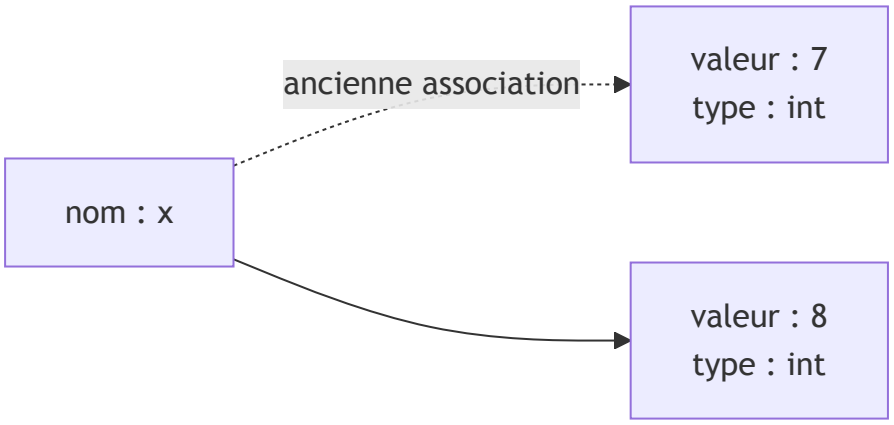
L'affectation `x = 7` signifie : le nom `x` désigne maintenant la valeur `7`.




```
x = x + 1
```

Dans l'instruction `x = x + 1`, Python évalue d'abord l'expression située à droite du signe `=`, puis associe le résultat au nom situé à gauche.

Si `x` vaut `7`, alors `x + 1` vaut `8`, puis `x` désigne la valeur `8`.



"Attention"

En Python, le symbole `=` ne signifie pas "est égal à" comme en mathématiques.

Il sert à affecter une valeur à une variable.

Pour tester l'égalité de deux valeurs, on utilise `==`.

```
x = 5       # affectation
x == 5      # comparaison, vaut True
```

Instruction, programme, état d'un programme

"Définition 3"

L'**état d'un programme** est l'ensemble des associations entre les noms de variables et les valeurs qu'ils désignent.

"Définition 4"

Une **expression** est un morceau de code que Python peut évaluer pour produire une valeur qui possède alors un **type**.

Une **instruction** est un ordre exécuté par Python. Elle peut modifier l'état du programme ou contrôler son exécution.

"Attention"

En Python, l'affectation `x = 3` est une instruction, pas une expression.

"Définition 5"

Un **programme** est une **séquence d'instructions**.

"Tableau d'évolution d'un programme"

On peut tracer l'évolution de l'état d'un programme dans un tableau.

On considère le programme suivant :

```
x = 3
y = 4
z = x
x = y
y = z
```

Tableau d'évolution des variables :

Instruction exécutée	Valeur de `x`	Valeur de `y`	Valeur de `z`
départ	non définie	non définie	non définie
`x = 3`	`3`	non définie	non définie
`y = 4`	`3`	`4`	non définie
`z = x`	`3`	`4`	`3`
`x = y`	`4`	`4`	`3`
`y = z`	`4`	`3`	`3`

Ce programme échange les valeurs de `x` et `y` en utilisant la variable intermédiaire `z`.

"Exercice 1"

On considère le programme suivant (à compléter en numérotant les lignes de code)

```
x = 3
y = 4
x = x + y
y = x - y
x = x - y
```

Compléter le tableau d'évolution des variables.

Instruction exécutée	Valeur de `x`	Valeur de `y`
départ	non définie	non définie
`x = 3`	_____	_____
`y = 4`	_____	_____
`x = x + y`	_____	_____
`y = x - y`	_____	_____
`x = x - y`	_____	_____

Instructions d'entrée et de sortie

"Entrée avec `input`"

Dans un programme, l'instruction `var = input("Message de prompt")` interrompt le flot d'exécution et attend la saisie d'une valeur au clavier par l'utilisateur. Cette valeur de type `str` est alors affectée à la variable `var`. On peut changer le message de prompt et le nom de la variable.

"Conversion d'entrée saisie avec `input`"

La valeur renvoyée par `input` est de type `str`. Si on attend la saisie d'un autre type, il faut penser à la convertir dans le type cible. Par exemple si on attend la saisie d'un âge de type `int`, on écrira :

```
age = int(input("Saisissez votre âge : "))
```

"Sortie avec `print`"

La fonction `print` permet de construire une instruction qui affiche une expression sur la sortie standard (une zone de l'IDE) avec `print(expression)`.

Exemple :

```
age = 16
print(age)
print("Vous avez", age, "ans")
```

Pour découvrir toutes les options possibles de `print`, saisir `help(print)` dans une console Python.

"Exemple de programme avec entrée et sortie"

Le programme suivant demande un taux de TVA et un prix hors taxes, puis affiche le prix toutes taxes comprises.

```
# Entrées
taux_tva = float(input("Taux de TVA en pourcentage ? "))
prix_ht = float(input("Prix hors taxes ? "))

# Traitement
coefficient = 1 + taux_tva / 100
prix_ttc = prix_ht * coefficient

# Sortie
print("Le prix TTC est :", prix_ttc)
```

Un programme est souvent organisé en trois parties :

- les **entrées** : les données nécessaires au calcul ;
- le **traitement** : les calculs effectués ;
- les **sorties** : les résultats affichés ou renvoyés.

"Définition"

Un **commentaire** est un texte écrit dans le programme mais ignoré par Python lors de l'exécution.
En Python, un commentaire commence par le caractère `#`.

Exemple :

```
# Ceci est un commentaire
x = 3  # ce commentaire explique l'instruction
```

"Exercice 3"

Écrire un programme qui demande à l'utilisateur de saisir son prénom, son année de naissance et l'année en cours et qui affiche un message personnalisé du type "[Prénom], en [année en cours] vous avez [âge] ans" en adaptant les valeurs entre crochets

Erreurs

"Erreur de type `NameError`"

L'erreur `NameError` apparaît lorsqu'un programme utilise un nom que Python ne connaît pas. Le cas le plus fréquent est l'utilisation d'une variable qui n'a pas encore été définie.

Exemple :

```
>>> x + 1
NameError: name 'x' is not defined
```

Ici, Python ne peut pas évaluer `x + 1`, car aucune valeur n'a encore été associée au nom `x`.

Autre exemple :

```
>>> age = 16
>>> agge + 1
NameError: name 'agge' is not defined
```

Ici, l'erreur vient d'une faute de frappe dans le nom de variable.

"Exercice 4"

Pour chaque programme, dire s'il provoque une erreur `NameError`. Si oui, expliquer pourquoi.

Programme A

```
x = 3
y = x + 2
print(y)
```

Programme B

```
x = 3
y = z + 2
print(y)
```

Programme C

```
prenom = "Alice"
print(prenom)
```

Programme D

```
prenom = "Alice"
print(prénom)
```

Programme E

```
age = 16
print(Age)
```
