Syntaxe de Python

Premiers pas en programmation

Résumé

Python est un langage de programmation inventé par Guido van Rossum en 1991. Sa syntaxe claire et lisible en fait un excellent premier langage. Ce cours présente les briques fondamentales : variables, affectation, types, affichage et saisie.

1 Variables et affectation

Définition 1 **(Variable)** Une **variable** est un nom qui désigne une donnée stockée en mémoire. On crée une variable en lui **affectant** une valeur avec l'opérateur `=`.

```
age = 17
nom = "Alice"
taille = 1.65
print(nom)
print(age)
```

Résultat : `Alice`, puis `17` sur la ligne suivante.

Le nom de la variable est à gauche du signe `=`, la valeur est à droite. Ce n'est pas une égalité mathématique : c'est une **instruction** qui associe un nom à une valeur. La variable n'est pas une boîte qui contient la donnée : c'est une **étiquette** collée sur un objet.

On distingue deux sortes de morceaux de code. Une **expression** est un calcul qui produit une valeur, par exemple `age + 1` ou `taille * 100`. Une **instruction** est un ordre que Python exécute, par exemple une affectation ou un affichage avec `print(...)`. Dans l'instruction `age = age + 1`, la partie droite `age + 1` est une expression.

Règles de nommage

Un nom de variable en Python doit respecter certaines règles :

— il ne contient que des lettres, des chiffres et le caractère `_` (underscore) ;

— il ne commence **jamais** par un chiffre ;

— il est sensible à la casse : `Age`, `age` et `AGE` sont trois variables différentes ;

— il ne doit pas être un mot réservé du langage (`if`, `for`, `while`, `def`, etc.) ; on évite aussi de réutiliser le nom d'une fonction intégrée comme `print`, `len` ou `input`, sous peine de la rendre inutilisable.

Par convention, on utilise le **snake_case** en Python : les mots sont séparés par des underscores et écrits en minuscules.

```python
nombre_de_joueurs = 4
temperature_max = 38.5
nom_de_famille = "Dupont"
```

Exercice 1 *Noms de variables* Parmi les noms suivants, lesquels sont des noms de variables valides en Python ? Pour ceux qui ne le sont pas, expliquez pourquoi.

1. `mon_age`
2. `2eme_note`
3. `MaVariable`
4. `for`
5. `_resultat`
6. `note-finale`

② La fonction `print()`

La fonction `print()` affiche à l'écran la valeur qu'on lui donne entre parenthèses : un texte, un nombre ou le contenu d'une variable.

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

Résultat : `Bonjour`, puis `Alice` sur la ligne suivante.

Les f-strings

Pour afficher dans une même phrase du texte et des variables, on utilise une **f-string** (chaîne formatée) : on écrit la lettre `f` juste avant les guillemets, puis on place chaque variable, ou chaque calcul, entre accolades `{ }`. À l'affichage, Python remplace chaque paire d'accolades par la valeur correspondante.

```python
prenom = "Alice"
age = 17
taille = 1.65
print(f"{prenom} a {age} ans et mesure {taille} m")
print(f"Dans 10 ans, {prenom} aura {age + 10} ans")
```

Résultat : `Alice a 17 ans et mesure 1.65 m`, puis `Dans 10 ans, Alice aura 27 ans`.

Exercice 2 *Affichage avec print et f-strings*

1. Créez trois variables : `ville` contenant `"Lyon"`, `departement` contenant `69`, `population` contenant `522969`.

2. Écrivez une instruction `print()` utilisant une f-string qui affiche : `Lyon est dans le département 69 et compte 522969 habitants.`

3. Écrivez une instruction qui affiche la population en milliers en utilisant une expression dans la f-string.

③ La fonction `input()`

La fonction `input()` permet de demander une saisie à l'utilisateur. Elle renvoie **toujours** une **chaîne de caractères** (`str`), même si l'utilisateur tape un nombre.

```
nom = input("Quel est votre nom ? ")
age = int(input("Quel est votre âge ? "))
print(f"Bonjour {nom}, vous avez {age} ans")
```

⚠ **Conversion obligatoire.** Si l'on veut travailler avec des nombres saisis par l'utilisateur, il faut convertir le résultat de `input()` avec `int()` (entier) ou `float()` (décimal).

Piège fréquent. Oublier la conversion provoque une erreur :

```
age = input("Quel est votre âge ? ")
print(age + 1)   # TypeError: can only concatenate str to str
```

Python ne sait pas additionner une chaîne et un entier. Il faut écrire `int(input(...))` pour obtenir un nombre.

Exercice 3 *Saisie utilisateur*

1. Écrivez un programme qui demande le prénom et l'année de naissance de l'utilisateur, puis affiche son âge approximatif.

2. Que se passe-t-il si l'on écrit `age = input("Âge ? ")` puis `age + 1` ? Pourquoi ?

④ Réaffectation

Une variable peut changer de valeur au cours du programme :

```
score = 0
print(score)     # 0

score = score + 10
print(score)     # 10

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

L'instruction `score = score + 10` n'est pas une équation mathématique (qui serait absurde). Elle signifie : « calculer la valeur de `score + 10`, puis affecter le résultat à `score` ».

Exercice 4 *Trace d'exécution* Donnez la valeur de chaque variable après chaque ligne :

```
a = 5
b = 3
c = a + b
a = a * 2
b = c - a
```

Complétez le tableau :

Après la ligne	a	b	c
`a = 5`			
`b = 3`			
`c = a + b`			
`a = a * 2`			
`b = c - a`			

⑤ Exécution pas à pas : la trace

Python exécute les instructions **dans l'ordre où elles sont écrites**, une par ligne, de haut en bas. Chaque instruction utilise la valeur que les variables ont **à ce moment-là** : une valeur calculée plus tôt ne se recalcule pas toute seule.

Pour comprendre un programme sans l'exécuter, on le suit ligne par ligne en notant, après chaque instruction, la valeur de chaque variable. Ce tableau s'appelle une **trace d'exécution** (ou **table de trace**).

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

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

Une case vide signifie que la variable n'existe pas encore. À la ligne 4, `b - a` se calcule avec les valeurs de la ligne 3 : 5 − 10, soit −5.

La trace sert aussi à trouver une erreur : quand un programme n'affiche pas ce que l'on attendait, la première ligne où une valeur s'écarte de ce que l'on prévoyait désigne l'instruction fautive.

6 Le type d'une donnée

Chaque donnée possède un **type** qui détermine ce qu'on peut en faire. La fonction `type()` permet de connaître le type :

```
print(type(42))          # <class 'int'>    (entier)
print(type(3.14))        # <class 'float'>  (flottant)
print(type("Bonjour"))   # <class 'str'>    (chaîne)
print(type(True))        # <class 'bool'>   (booléen)
```

En Python, c'est la **donnée** qui est typée, pas la variable. Une variable peut donc changer de type :

```
x = 42
print(type(x))  # <class 'int'>

x = "quarante-deux"
print(type(x))  # <class 'str'>
```

⚠ **Piège classique.** `42` est un entier, mais `"42"` est une chaîne de caractères. Les guillemets font toute la différence.

Exercice 5 *Types de données* Sans exécuter le code, donnez le type de chaque expression :

1. `42`
2. `"42"`
3. `3.0`
4. `True`

7 Les commentaires

Un **commentaire** commence par le caractère #. Tout ce qui suit sur la même ligne est ignoré par Python. Les commentaires servent à expliquer le code :

```
# Calcul de la moyenne de trois notes
note1 = 15
note2 = 12
note3 = 18
moyenne = (note1 + note2 + note3) / 3  # division par 3
print(f"La moyenne est {moyenne}")
```

8 Échange de deux variables

Comment échanger les valeurs de deux variables ? La première idée ne fonctionne pas :

```
a = 5
b = 3
a = b   # a vaut maintenant 3, mais on a perdu le 5 !
b = a   # b vaut 3 aussi... raté !
```

Il faut utiliser une variable **temporaire** :

```
a = 5
b = 3
temp = a   # on sauvegarde a
a = b      # a prend la valeur de b
b = temp   # b récupère l'ancienne valeur de a
```

En Python, il existe un raccourci élégant grâce à l'**affectation multiple** :

```
a, b = b, a
```

Exercice 6 *Échange et affectation multiple*

1. Qu'affiche le programme suivant ?

```
x = 10
y = 20
z = 30
x, y, z = z, x, y
print(f"{x} {y} {z}")
```

2. Écrivez un programme qui échange les valeurs de trois variables en rotation : `a` prend la valeur de `b`, `b` prend celle de `c`, `c` prend celle de `a`. Testez avec $a = 1$, $b = 2$, $c = 3$.

Synthèse

Concept	Syntaxe	Exemple
Affectation	`variable = valeur`	`age = 17`
Affichage	`print(...)`	`print("Bonjour")`
f-string	`f"...{var}..."`	`f"J'ai {age} ans"`
Saisie	`input(...)`	`nom = input("Nom ?")`
Conversion	`int()`, `float()`	`int(input("Âge ?"))`
Type	`type(...)`	`type(42)` → `int`
Commentaire	`# texte`	`# ceci est ignoré`
Échange	`a, b = b, a`	échange les valeurs