Variables, types, structures conditionnelles et boucles

NSI – Première

Séquence 1 : les bases du langage Python

Table des matières

1 Un programme : une séquence d'instructions

Définition 1.1: Séquence d'instructions

Un programme est une **séquence d'instructions** : une suite d'instructions exécutées les unes après les autres, dans l'ordre où elles sont écrites, de haut en bas. C'est la construction la plus élémentaire de tout langage de programmation.

Exemple 1.2: L'ordre des instructions a de l'importance

```
1 print("Bonjour")
2 print("Comment vas-tu ?")
```

Ce programme affiche d'abord `Bonjour`, puis `Comment vas-tu ?`. Si on inverse l'ordre des deux lignes, les deux messages s'affichent toujours, mais dans l'autre sens :

```
1 print("Comment vas-tu ?")
2 print("Bonjour")
```

Le résultat n'a alors plus vraiment de sens : l'ordre des instructions change le déroulement, et donc le résultat, du programme.

Remarque : Lecture séquentielle

Python lit et exécute un programme ligne par ligne, dans l'ordre du texte. Chaque instruction peut modifier l'**état du programme** (par exemple, ce qui a déjà été affiché à l'écran), et cet état est celui que « voit » l'instruction suivante au moment où elle s'exécute. Les structures que nous verrons ensuite (conditionnelles, boucles) permettent de s'écarter de cette lecture purement linéaire, en répétant ou en sautant certains blocs d'instructions. Nous verrons bientôt que les **variables** sont un moyen de mémoriser une partie de cet état pour la réutiliser plus tard dans le programme.

2 Les variables informatiques

Définition 2.1: Variable

Une **variable** est un espace de la mémoire de l'ordinateur, désigné par un **nom** (ou *identificateur*), dans lequel est stockée une **valeur**. Cette valeur peut être modifiée au cours de l'exécution du programme.

En Python, on crée une variable et on lui donne une valeur grâce à l'opérateur `=`, appelé **opérateur d'affectation**.

Exemple 2.2: Affectation d'une variable

```
1 age = 16
2 prenom = "Alice"
3 print(age)
4 print(prenom)
```

Ici, la variable `age` contient la valeur `16` et la variable `prenom` contient la valeur `"Alice"`.

Remarque : Affectation et égalité

En Python, le symbole `=` **n'est pas** le symbole d'égalité mathématique : il signifie « la variable de gauche reçoit la valeur de droite ». C'est pourquoi une instruction comme

```
1 compteur = compteur + 1
```

a un sens en programmation (on remplace l'ancienne valeur de `compteur` par l'ancienne valeur augmentée de 1), alors qu'elle n'aurait aucun sens comme égalité mathématique. Pour tester une égalité en Python, on utilise l'opérateur `==` (voir section 4).

Méthode 2.3: Nommer une variable

Un nom de variable :

- ne peut contenir que des lettres, des chiffres et le caractère `_` (underscore) ;
- ne peut pas commencer par un chiffre ;
- est sensible à la casse (`age` et `Age` sont deux variables différentes) ;
- doit, par convention, être écrit en minuscules et être **explicite** (préférer `score_joueur` à `s`).

Exercice 2.4: Prise en main

1. Créer une variable `nom` contenant votre prénom, puis l'afficher avec `print`.
2. Créer une variable `a` valant `5`, puis une variable `b` valant `3`. Afficher la somme `a + b`.
3. Sans relancer le programme depuis le début, ajouter une instruction qui remplace la valeur de `a` par `a + 10`, puis afficher la nouvelle valeur de `a`.

3 Les types de base

Définition 3.1: Type d'une variable

Le **type** d'une variable précise la nature de la valeur qu'elle contient, et donc les opérations que l'on peut effectuer avec elle. En Python, le type se déduit automatiquement de la valeur affectée : on parle de **typage dynamique**.

Propriété 3.2: Types de base en Python

- `int` : un nombre entier (ex : `7`, `-3`) ;
- `float` : un nombre décimal (ex : `3.14`, `-0.5`) ;
- `str` : une chaîne de caractères, délimitée par des guillemets (ex : `"bonjour"`) ;
- `bool` : un booléen, qui ne prend que deux valeurs : `True` ou `False`.

On obtient le type d'une variable avec la fonction `type()`.

Exemple 3.3: Identifier des types

```
a = 12
b = 3.5
c = "NSI"
d = True

print(type(a))   # <class 'int'>
print(type(b))   # <class 'float'>
print(type(c))   # <class 'str'>
print(type(d))   # <class 'bool'>
```

Remarque : Conversion de type

On peut convertir explicitement une valeur d'un type vers un autre grâce aux fonctions `int()`, `float()`, `str()`, `bool()`. C'est en particulier indispensable avec `input()`, qui renvoie toujours une chaîne de caractères (`str`), même si l'utilisateur tape un nombre.

```
age_str = input("Quel est ton âge ? ")
age = int(age_str)
print(age + 1)
```

Exercice 3.4: Types et conversions

1. Prédire, puis vérifier avec `type()`, le type des valeurs suivantes : `10`, `10.0`, `"10"`, `10 == 10`.
2. Que se passe-t-il si on essaie de calculer `"5" + 3` ? Proposer une correction pour que le programme fonctionne.

4 Les structures conditionnelles

Définition 4.1: Structure conditionnelle

Une **structure conditionnelle** permet d'exécuter un bloc d'instructions uniquement si une **condition** (une expression qui vaut `True` ou `False`) est vérifiée.

Notation 4.2: Opérateurs de comparaison

Opérateur	Signification
`==`	égal à
`!=`	différent de
`<` / `>`	strictement inférieur / supérieur
`<=` / `>=`	inférieur ou égal / supérieur ou égal

On peut combiner plusieurs conditions avec `and` (ET), `or` (OU) et `not` (NON).

Syntaxe : Le `if`

```
1 if condition_1:
2     # instructions si condition_1 est vraie
3 elif condition_2:
4     # instructions si condition_1 est fausse ET condition_2 est vraie
5 else:
6     # instructions si aucune condition n'est vraie
```

Les blocs `elif` et `else` sont facultatifs. L'indentation (les espaces en début de ligne) n'est pas esthétique : c'est elle qui délimite les blocs d'instructions en Python.

Exemple 4.3: Majorité

```
1 age = int(input("Quel âge as-tu ? "))
2
3 if age >= 18:
4     print("Tu es majeur.")
5 else:
6     print("Tu es mineur.")
```

Remarque : Erreur fréquente

Oublier les deux points `:` après la condition, ou mal indenter le bloc qui suit, provoque une erreur (`IndentationError` ou `SyntaxError`). En Python, contrairement à d'autres langages, l'indentation fait partie de la syntaxe : elle n'est pas facultative.

Exercice 4.4: Notes et mentions

Écrire un programme qui demande une note sur 20 à l'utilisateur et affiche :
- « Insuffisant » si la note est strictement inférieure à 10 ;
- « Passable » si la note est entre 10 et 12 (exclu) ;
- « Assez bien » si la note est entre 12 et 14 (exclu) ;
- « Bien » si la note est entre 14 et 16 (exclu) ;
- « Très bien » si la note est supérieure ou égale à 16.

5 La boucle bornée : `for`

Définition 5.1: Boucle bornée

Une **boucle bornée** permet de répéter un bloc d'instructions un nombre de fois **connu à l'avance**. En Python, on utilise pour cela l'instruction `for ... in ...`. On parle d'**itération** pour chaque répétition du bloc.

Syntaxe : Le `for` avec `range`

```
for i in range(n):
    # instructions répétées, utilisant éventuellement i
```

La variable `i` est appelée **variable de boucle** : elle prend successivement les valeurs `0, 1, 2, ..., n-1`.

Propriété 5.2: La fonction `range`

`range(debut, fin, pas)` engendre la suite arithmétique de premier terme `debut`, de raison `pas`, s'arrêtant strictement avant `fin`.

- `range(n)` équivaut à `range(0, n, 1)` : les entiers de `0` à `n-1` ;
- `range(a, b)` équivaut à `range(a, b, 1)` : les entiers de `a` à `b-1`.

Exemple 5.3: Répéter et parcourir

```
# Répéter 5 fois un affichage
for i in range(5):
    print("Bonjour")

# Parcourir les caractères d'une chaîne
for lettre in "NSI":
    print(lettre)
```

Remarque : Ne pas modifier la variable de boucle

Il est possible d'utiliser la variable de boucle dans les instructions répétées, mais il ne faut pas la modifier à l'intérieur de la boucle : ses valeurs successives sont entièrement gérées par `range()`.

Exercice 5.4: Somme et compteur

1. Écrire un programme qui calcule la somme des entiers de `1` à `100` à l'aide d'une boucle `for` et d'une variable accumulateur.
2. Écrire un programme qui affiche la table de multiplication par 7, de `7 x 1` à `7 x 10`.

6 La boucle non bornée : `while`

Définition 6.1: Boucle non bornée

Une **boucle non bornée** permet de répéter un bloc d'instructions **tant qu'une condition** reste vraie, sans connaître à l'avance le nombre de répétitions. En Python, on utilise pour cela l'instruction `while`.

Syntaxe : Le `while`

```
while condition:
    # instructions répétées tant que condition est vraie
```

Méthode 6.2: Construire une boucle non bornée

Écrire correctement une boucle non bornée suppose en général trois étapes :

1. **initialisation** de la (ou des) variable(s) utilisée(s) dans la condition ;
2. **condition d'arrêt**, testée avant chaque itération ;
3. **mise à jour** de la variable, à l'intérieur de la boucle, pour que la condition finisse par devenir fausse.

Exemple 6.3: Compter jusqu'à un seuil

```
1 i = 1                  # initialisation
2 while i < 7:           # condition d'arrêt
3     print(i)
4     i = i + 2           # mise à jour
```

Remarque : Le danger de la boucle infinie

Si la condition d'une boucle `while` reste toujours vraie (par exemple si on oublie de mettre à jour la variable de contrôle), le programme ne s'arrête jamais : on parle de **boucle infinie**. Il faut alors interrompre l'exécution manuellement. Pensez à sauvegarder régulièrement votre travail avant de tester une boucle `while`.

Exemple 6.4: Boucle non bornée par nature

Certains problèmes imposent une boucle `while` car le nombre d'itérations n'est pas connu à l'avance : par exemple, demander un mot de passe jusqu'à ce qu'il soit correct.

```
1 mot_de_passe = ""
2 while mot_de_passe != "python2026":
3     mot_de_passe = input("Mot de passe : ")
4 print("Accès autorisé.")
```

Exercice 6.5: `for` ou `while` ?

1. Pour chacune des situations suivantes, indiquer si une boucle `for` ou une boucle `while` est la plus adaptée, puis écrire le programme correspondant :
   - afficher les 10 premiers multiples de 3 ;
   - diviser un nombre par 2 tant que le résultat est supérieur à 1, et compter le nombre de divisions effectuées.
2. Réécrire la boucle `for i in range(5): print(i)` à l'aide d'une boucle `while` produisant exactement le même affichage.

7 Bilan

Bilan

- Une **variable** associe un nom à une valeur stockée en mémoire ; l'**affectation** (`=`) n'est pas une égalité mathématique.
- Le **type** (`int` , `float` , `str` , `bool`) détermine la nature d'une valeur et les opérations possibles ; Python le déduit automatiquement.
- Une **structure conditionnelle** (`if` / `elif` / `else`) exécute un bloc selon la valeur d'une condition booléenne.
- Une boucle **bornée** (`for`) répète un nombre connu à l'avance de fois ; une boucle **non bornée** (`while`) répète tant qu'une condition reste vraie, au risque d'une boucle

infinie si elle n'évolue jamais vers `False`.

Conclusion

Ces quatre notions – variables, types, conditions, boucles – forment le socle de tout programme Python. Elles seront réutilisées et combinées dans les séquences suivantes, notamment avec les **fonctions**, qui permettront de regrouper ces instructions pour les réutiliser facilement.