

TP (1h)

Petits programmes

NSI – Première – Variables, boucles et `input`

Dans ce TP, vous allez écrire **seuls**, sans code de départ, une série de petits programmes indépendants. Chacun demande des informations à l'utilisateur avec `input`, puis utilise des variables et une boucle pour produire un résultat.

Partie	Notions	Durée indicative
A – Parcourir une chaîne	<code>for lettre in mot</code>	20 min
B – Répéter n fois	<code>for i in range(...)</code>	20 min
C – Répéter tant que	<code>while</code>	20 min

Méthode de travail :

- un programme par fichier (`exo1.py`, `exo2.py`, etc.);
- pour chaque exercice, un **exemple d'exécution** vous montre ce que doit afficher votre programme : ce que tape l'utilisateur apparaît à la suite de chaque question ;
- testez chaque programme **plusieurs fois**, avec des valeurs différentes, avant de passer au suivant ;
- l'exercice bonus est réservé à ceux qui ont terminé.

Rappel : `input()` renvoie toujours une chaîne de caractères. Pour obtenir un nombre, il faut la convertir : `int(input("..."))` pour un entier, `float(input("..."))` pour un décimal.

1 Parcourir une chaîne de caractères

Exercice 1: Compter une lettre

Écrire un programme qui demande un mot, puis une lettre, et affiche le nombre de fois où cette lettre apparaît dans le mot.

```
Entrez un mot : banane
Quelle lettre compter ? a
La lettre a apparaît 2 fois dans banane
```

Indication

Utilisez une variable `compteur` qui vaut `0` au départ, et augmentez-la de 1 à chaque fois que la lettre parcourue est égale à la lettre cherchée.

Correction de l'exercice 1

```
1 mot = input("Entrez un mot : ")
2 lettre_cherchee = input("Quelle lettre compter ? ")
3 compteur = 0
4 for lettre in mot:
5     if lettre == lettre_cherchee:
6         compteur = compteur + 1
7 print("La lettre", lettre_cherchee, "apparaît", compteur, "fois dans", mot)
```

Le compteur doit être initialisé **avant** la boucle : s'il l'était à l'intérieur, il serait remis à 0 à chaque tour.

Exercice 2: Compter les voyelles

Écrire un programme qui demande une phrase (en minuscules) et affiche son nombre de voyelles (a, e, i, o, u, y).

```
Entrez une phrase : bonjour tout le monde
Nombre de voyelles : 8
```

Indication

Inutile d'écrire six tests `if` : l'opérateur `in` permet de tester si un caractère fait partie de la chaîne `"aeiouy"`.

Correction de l'exercice 2

```
1 phrase = input("Entrez une phrase : ")
2 nb_voyelles = 0
3 for caractere in phrase:
4     if caractere in "aeiouy":
5         nb_voyelles = nb_voyelles + 1
6 print("Nombre de voyelles :", nb_voyelles)
```

Même structure qu'à l'exercice 1 ; seule la condition change. Les espaces sont parcourus comme les autres caractères, mais ne sont pas comptés puisqu'ils ne sont pas dans `"aeiouy"`.

Exercice 3: Mot à l'envers

1. Écrire un programme qui demande un mot et l'affiche à l'envers.
2. Compléter le programme pour qu'il indique ensuite si le mot est un **palindrome**, c'est-à-dire un mot qui se lit de la même façon dans les deux sens (comme « radar » ou « kayak »).

```
Entrez un mot : python
À l'envers : nohtyp
Ce n'est pas un palindrome.
```

Indication

Construisez le mot à l'envers dans une variable `envers` qui vaut `""` au départ. À chaque tour, la question est : faut-il écrire `envers + lettre` ou `lettre + envers` ? Essayez les deux à la main sur le mot « nsi ».

Correction de l'exercice 3

```
1 mot = input("Entrez un mot : ")
2 envers = ""
3 for lettre in mot:
4     envers = lettre + envers
5 print("À l'envers :", envers)
6 if envers == mot:
7     print("C'est un palindrome !")
8 else:
9     print("Ce n'est pas un palindrome.")
```

En ajoutant chaque lettre **devant** ce qui a déjà été construit, la dernière lettre du mot se retrouve en première position. Sur « nsi » : `"n"`, puis `"sn"`, puis `"isn"`. Avec `envers + lettre`, on aurait simplement recopié le mot à l'identique.

2 Répéter un nombre de fois connu

Exercice 4: Table de multiplication

Écrire un programme qui demande un entier n et affiche sa table de multiplication, de 1 à 10.

```
Quelle table ? 7
7 x 1 = 7
7 x 2 = 14
...
7 x 10 = 70
```

Correction de l'exercice 4

```
1 n = int(input("Quelle table ? "))
2 for i in range(1, 11):
3     print(n, "x", i, "=", n * i)
```

`range(1, 11)` produit les entiers de 1 à 10 (la borne de fin est exclue). Sans la conversion `int(...)`, `n * i` répéterait la chaîne `"7"` au lieu de calculer un produit.

Exercice 5: Triangle d'étoiles

Écrire un programme qui demande une hauteur, puis dessine un triangle d'étoiles de cette hauteur.

```
Hauteur du triangle : 4
*
**
***
****
```

Indication

Utilisez une variable `ligne` qui vaut `""` au départ : à chaque tour de boucle, on lui ajoute une étoile, puis on l'affiche.

Correction de l'exercice 5

```
1 hauteur = int(input("Hauteur du triangle : "))
2 ligne = ""
3 for i in range(hauteur):
4     ligne = ligne + "*"
5     print(ligne)
```

Ici, la variable de boucle `i` ne sert qu'à compter les tours : on ne l'utilise pas dans le corps de la boucle. Le `print` doit être **dans** la boucle, pour afficher la ligne à chaque étape.

Exercice 6: Moyenne de notes

Écrire un programme qui demande combien de notes l'utilisateur veut saisir (au moins une), puis demande chacune de ces notes, et affiche enfin leur moyenne.

```
Combien de notes ? 3
Note : 12
Note : 15.5
Note : 9
Moyenne : 12.166666666666666
```

Indication

Le `input` qui demande une note doit se trouver **à l'intérieur** de la boucle. Additionnez les notes au fur et à mesure dans une variable `somme`. Les notes pouvant être décimales, convertissez-les avec `float`.

Correction de l'exercice 6

```
1 nb_notes = int(input("Combien de notes ? "))
2 somme = 0
3 for i in range(nb_notes):
4     note = float(input("Note : "))
5     somme = somme + note
6 print("Moyenne :", somme / nb_notes)
```

Le nombre de tours n'est pas écrit dans le programme : il est choisi par l'utilisateur, mais il est **connu avant** le début de la boucle, d'où le `for`. La division se fait **après** la boucle, une seule fois. Si l'utilisateur saisit `0`, on obtient une division par zéro : c'est pourquoi l'énoncé impose au moins une note.

3 Répéter tant qu'une condition est vraie

Exercice 7: Saisie contrôlée

Écrire un programme qui demande une note entre 0 et 20. Tant que la valeur saisie n'est pas valide, le programme affiche un message d'erreur et redemande la note.

```
Entrez une note entre 0 et 20 : 25
Note invalide !
Entrez une note entre 0 et 20 : -2
Note invalide !
Entrez une note entre 0 et 20 : 14
Note enregistrée : 14.0
```

Indication

Demandez une première fois la note **avant** la boucle. La condition du `while` doit être vraie quand la note est **invalide** : elle combine deux comparaisons avec `or`.

Correction de l'exercice 7

```
1 note = float(input("Entrez une note entre 0 et 20 : "))
2 while note < 0 or note > 20:
3     print("Note invalide !")
4     note = float(input("Entrez une note entre 0 et 20 : "))
5     print("Note enregistrée :", note)
```

On ne sait pas à l'avance combien de fois l'utilisateur se trompera : c'est une boucle non bornée. Une note est invalide si elle est trop petite **ou** trop grande, d'où le **or**. Le nouvel **input** dans la boucle met à jour la variable testée : sans lui, la boucle serait infinie.

Exercice 8: Le plus grand

Écrire un programme qui demande des nombres entiers positifs, les uns après les autres, jusqu'à ce que l'utilisateur tape **-1**. Le programme affiche alors combien de nombres ont été saisis (sans compter le **-1**) et le plus grand d'entre eux. Si le premier nombre saisi est **-1**, le programme affiche « Aucun nombre saisi. »

```
Entrez un nombre (-1 pour arrêter) : 14
Entrez un nombre (-1 pour arrêter) : 42
Entrez un nombre (-1 pour arrêter) : 17
Entrez un nombre (-1 pour arrêter) : -1
Vous avez saisi 3 nombre(s).
Le plus grand est 42
```

Indication

Même schéma qu'à l'exercice précédent : une première saisie avant la boucle, une nouvelle saisie à la fin de chaque tour. Pour le plus grand, utilisez une variable **plus_grand** qui prend comme valeur de départ le **premier** nombre saisi ; à chaque tour, si le nombre saisi la dépasse, elle est remplacée.

Correction de l'exercice 8

```
1 nombre = int(input("Entrez un nombre (-1 pour arrêter) : "))
2 plus_grand = nombre
3 compteur = 0
4 while nombre != -1:
5     compteur = compteur + 1
6     if nombre > plus_grand:
7         plus_grand = nombre
8     nombre = int(input("Entrez un nombre (-1 pour arrêter) : "))
9 if compteur == 0:
10     print("Aucun nombre saisi.")
11 else:
12     print("Vous avez saisi", compteur, "nombre(s).")
13     print("Le plus grand est", plus_grand)
```

Points clés :

- la nouvelle saisie est la **dernière** instruction de la boucle : ainsi, le **-1** est testé par la condition du **while** avant d'être compté ou comparé ;
- initialiser **plus_grand** avec le premier nombre saisi (plutôt qu'avec **0**) garantit que la valeur de départ est bien l'un des nombres de l'utilisateur ;
- le test **compteur == 0** après la boucle traite le cas où l'utilisateur tape **-1** dès le départ.

Exercice 9: Bonus – Le nombre mystère

L'ordinateur choisit un nombre entier au hasard entre 1 et 100. L'utilisateur fait des propositions ; après chacune, le programme répond « C'est plus ! » ou « C'est moins ! », jusqu'à ce que le nombre soit trouvé. Il affiche alors le nombre d'essais utilisés.

Pour tirer le nombre au hasard, commencez votre programme par ces deux lignes :

```
1 from random import randint
2 mystere = randint(1, 100)
```

```
Votre proposition : 50
C'est plus !
Votre proposition : 75
C'est moins !
Votre proposition : 62
Bravo ! Trouvé en 3 essais.
```

Correction de l'exercice 9

```
1 from random import randint
2 mystere = randint(1, 100)
3
4 essais = 0
5 proposition = 0
6 while proposition != mystere:
7     proposition = int(input("Votre proposition : "))
8     essais = essais + 1
9     if proposition < mystere:
10         print("C'est plus !")
11     elif proposition > mystere:
12         print("C'est moins !")
13 print("Bravo ! Trouvé en", essais, "essais.")
```

Autre façon d'organiser la boucle : ici, **proposition** est initialisée à **0** (une valeur qui ne peut pas être le nombre mystère) pour que la condition du **while** soit vraie au premier tour, et la saisie est placée au **début** de la boucle. Le cas « égal » n'a pas besoin de branche : c'est la condition du **while** qui arrête la boucle.

Remarque : Ce qu'il faut retenir de ce TP

Presque tous ces programmes reposent sur les mêmes schémas :

- une variable **initialisée avant** la boucle (`compteur = 0`, `somme = 0`, `envers = ""`) et **mise à jour à chaque tour** ;
- `for` quand le nombre de tours est connu avant de commencer (longueur d'une chaîne, valeur saisie) ; `while` quand il dépend de ce qui se passe pendant la boucle ;
- avec `while`, une valeur qui change à chaque tour, pour que la boucle finisse par s'arrêter.