

Exercices de synthèse

Variables, types, conditions et boucles

NSI – Première – Séquence 1

Cette feuille mélange **toutes** les notions de la séquence 1 : séquences d'instructions, variables, types et conversions, structures conditionnelles, boucles bornées et non bornées. Les exercices sont nettement plus exigeants que les précédents : la plupart demandent de combiner au moins trois notions à la fois.

La partie A se traite **sans ordinateur**, la partie B **sur machine**, la partie C regroupe des **défis** à traiter en autonomie.

Remarque : Deux opérateurs à connaître

Plusieurs exercices utilisent les deux opérateurs suivants sur les entiers :

- `a // b` : le **quotient entier** de la division euclidienne de `a` par `b` (`17 // 5` vaut `3`);
- `a % b` : le **reste** de cette division (`17 % 5` vaut `2`).

En particulier, `n % 2 == 0` teste si `n` est pair, et, pour un entier positif `n`, `n % 10` donne son chiffre des unités tandis que `n // 10` donne le nombre privé de son chiffre des unités.

Partie A – Analyser un programme (sans ordinateur)

1 Types en cascade

Exercice 1: Types en cascade

On exécute le programme suivant :

```

1  a = "7"
2  b = 3
3  c = int(a) + b
4  d = a * b
5  e = (c > 10) + (c == 10)
6  f = str(c) + a
7  g = float(b) / 2

```

1. Recopier et compléter le tableau suivant, en donnant pour chaque variable sa **valeur** (écrite exactement comme Python l'afficherait) et son **type**.

Variable	a	b	c	d	e	f	g
Valeur							
Type							

2. L'instruction `d = a * b` ne provoque aucune erreur, alors que `a + b` en provoquerait une. Expliquer précisément pourquoi.

3. La variable `e` contient un nombre alors qu'elle est construite uniquement à partir de comparaisons. Expliquer.
4. Les variables `c` et `f` ont-elles la même valeur ? Le même type ? Justifier.
5. L'instruction `g = float(b) / 2` donne `1.5`. Qu'obtiendrait-on avec `b / 2` ? Avec `b // 2` ? Que peut-on en conclure sur le type du résultat de l'opérateur `/` ?

2 Deux boucles imbriquées

Exercice 2: Deux boucles imbriquées

On exécute le programme suivant :

```

1 total = 0
2 for i in range(1, 4):
3     for j in range(i, 4):
4         if (i + j) % 2 == 0:
5             total = total + i * j
6         else:
7             total = total - 1
8     print(i, total)
9 print("Fin", total)

```

1. Combien de fois, au total, le test `(i + j) % 2 == 0` est-il évalué ? Justifier sans exécuter le programme.
2. Recopier et compléter le tableau ci-dessous, avec une ligne par tour de la boucle intérieure.

i	j	(i+j) pair ?	total après ce tour
1	1		
...			

3. En déduire l'affichage complet du programme, ligne par ligne.
4. L'instruction `print(i, total)` est-elle dans la boucle intérieure, dans la boucle extérieure, ou en dehors des deux ? Qu'afficherait le programme si on l'indentait d'un niveau supplémentaire ?

3 Une boucle while à deux variables

Exercice 3: Une boucle while à deux variables

On exécute le programme suivant :

```

1  a = 42
2  b = 30
3  while a != b:
4      if a > b:
5          a = a - b
6      else:
7          b = b - a
8  print(a)

```

1. Tracer l'exécution : recopier et compléter un tableau donnant les valeurs de **a** et **b** après chaque tour de boucle. Combien de tours la boucle effectue-t-elle ? Qu'affiche le programme ?
2. Reprendre la question précédente avec **a = 18** et **b = 12**, puis avec **a = 20** et **b = 7**.
3. À partir de ces trois exemples, formuler une conjecture : que calcule ce programme ?
4. Que se passe-t-il si l'on exécute ce programme avec **a = 5** et **b = 5** ? Et avec **a = 5** et **b = 0** ? Le programme se termine-t-il toujours ? Justifier.
5. Proposer une modification du programme pour qu'il affiche aussi le **nombre de tours** effectués.

Indication

Pour la question 4, on pourra remarquer que la quantité **a + b** diminue strictement à chaque tour **tant que** les deux variables restent strictement positives.

4 Cinq erreurs à débusquer

Exercice 4: Cinq erreurs à débusquer

Un élève souhaite écrire un programme qui demande un entier **n** à l'utilisateur, puis affiche combien il y a de multiples de 3 parmi les entiers de 1 à **n** (bornes comprises). Il propose ceci :

```

1  n = input("Jusqu'à quel nombre ? ")
2  compteur = 0
3  i = 1
4  while i < n:
5      if i % 3 = 0:
6          compteur = compteur + 1
7  i = i + 1
8  print("Il y a " + compteur + " multiples de 3.")

```

Ce programme contient **cinq** erreurs.

1. Les repérer toutes, en indiquant pour chacune le numéro de ligne concerné.
2. Classer chaque erreur dans l'une des trois catégories suivantes :
 - **erreur de syntaxe** : le programme refuse de démarrer ;

- **erreur d'exécution** : le programme démarre puis s'interrompt sur un message d'erreur ;
 - **erreur de logique** : le programme s'exécute sans message d'erreur, mais ne fait pas ce qui était attendu.
3. Réécrire le programme corrigé.
 4. Vérifier mentalement le programme corrigé avec `n = 10` : quelle valeur doit-il afficher ?

Partie B – Écrire un programme (sur machine)

5 Racine numérique

Exercice 5: Racine numérique

La **racine numérique** d'un entier positif s'obtient en additionnant ses chiffres, puis en recommençant sur le résultat, jusqu'à n'obtenir qu'un seul chiffre. Par exemple, pour 9875 :

$$9875 \longrightarrow 9 + 8 + 7 + 5 = 29 \longrightarrow 2 + 9 = 11 \longrightarrow 1 + 1 = 2.$$

La racine numérique de 9875 est donc 2, obtenue en 3 étapes.

Dans un fichier `racine_numerique.py` :

1. Écrire un programme qui demande un entier positif `n` à l'utilisateur et affiche la **somme de ses chiffres**. On utilisera une boucle `while` ainsi que les opérateurs `%` et `//`, sans convertir `n` en chaîne de caractères.
2. Compléter ce programme pour qu'il calcule la **racine numérique** de `n`, en répétant l'opération précédente tant que le nombre obtenu comporte plus d'un chiffre. Le programme affichera la racine numérique **et** le nombre d'étapes effectuées.
3. Faire afficher au passage chaque étape intermédiaire, sous la forme `9875 -> 29`.
4. Tester le programme sur `9875`, `1`, `99999999` et `10`. Que remarque-t-on pour `1` et pour `10` ?
5. **Pour aller plus loin.** Écrire une seconde version de la question 1 qui, cette fois, convertit `n` en chaîne de caractères et parcourt ses caractères avec une boucle `for`. Laquelle des deux versions vous paraît la plus lisible ?

Indication

Question 1 : à chaque tour, `n % 10` donne le chiffre des unités, et `n // 10` donne le nombre débarrassé de ce chiffre. On s'arrête quand il ne reste plus rien, c'est-à-dire quand `n` vaut `0`. Attention : si l'on modifie `n` dans cette boucle, sa valeur initiale est perdue – travailler sur une copie si l'on en a encore besoin ensuite.

6 La conjecture de Syracuse

Exercice 6: La conjecture de Syracuse

On part d'un entier $u_0 \geq 1$ et on construit une suite de nombres selon la règle suivante :

- si le terme courant est **pair**, le suivant est sa moitié ;
- s'il est **impair**, le suivant vaut son triple augmenté de 1.

Par exemple, en partant de 6 : 6, 3, 10, 5, 16, 8, 4, 2, 1. La **conjecture de Syracuse** affirme que, quel que soit le point de départ, la suite finit toujours par atteindre 1 – personne n'a su le démontrer à ce jour.

Dans un fichier `syracuse.py` :

1. Écrire un programme qui demande u_0 et affiche tous les termes de la suite jusqu'à atteindre 1 (compris).
2. Faire afficher le **temps de vol** : le nombre d'étapes nécessaires pour atteindre 1.
3. Faire afficher l'**altitude maximale** : la plus grande valeur atteinte par la suite.
4. Faire afficher le **temps de vol en altitude** : le nombre d'étapes effectuées avant que la suite ne devienne, pour la première fois, strictement inférieure à u_0 .
5. Tester avec $u_0 = 15$, puis $u_0 = 27$. Commenter.
6. Pourquoi une boucle `while` est-elle ici indispensable ? Une boucle `for` conviendrait-elle ?

Indication

Question 3 : tenir à jour une variable `altitude_max`, initialisée à u_0 , et la mettre à jour à chaque nouveau terme s'il est plus grand.

Question 4 : le mot important est « pour la première fois ». Il faut donc un moyen de ne retenir que le premier passage sous u_0 – par exemple à l'aide d'un booléen indiquant si ce passage a déjà eu lieu.

7 Anatomie d'un mot

Exercice 7: Anatomie d'un mot

Dans un fichier `mot.py`, écrire un programme qui demande un mot à l'utilisateur (en minuscules, sans accent) et affiche successivement :

1. le nombre de **voyelles** et le nombre de **consonnes** qu'il contient ;
2. le mot **à l'envers**, sans utiliser aucune fonction toute faite ;
3. si le mot est un **palindrome** (il se lit de la même façon dans les deux sens) ;
4. si le mot contient **deux lettres identiques consécutives** (comme le double `n` de `"bonne"`), et, le cas échéant, laquelle est la première ;
5. la plus longue **suite de voyelles consécutives** qu'il contient (par exemple 3 pour `"oiseau"`, qui contient `eau`).

Tester sur `radar`, `bonne`, `oiseau`, `rythme` et `a`.

Indication

Question 2 : construire une chaîne `envers` initialement vide, puis, pour chaque lettre du mot, placer cette lettre **devant** ce qui a déjà été construit.

Questions 4 et 5 : il faut se souvenir de la lettre précédente. On utilisera une variable `precedente`, initialisée à la chaîne vide et mise à jour à la fin de chaque tour de boucle.

Partie C – Défis

8 Défi 1 : la pyramide

Exercice 8: La pyramide

Écrire un programme qui demande un entier `n` compris entre 1 et 9, et affiche une pyramide de chiffres de hauteur `n`. Pour `n = 5`, on doit obtenir exactement :

```

1      1
2     121
3    12321
4   1234321
5  123454321
  
```

1. Avant d'écrire la moindre ligne de code, décrire précisément la ligne numéro `i` : combien d'espaces au début ? quels chiffres ensuite, et dans quel ordre ? combien de caractères en tout ?
2. Écrire le programme. On construira chaque ligne dans une variable de type `str` avant de l'afficher.
3. Pourquoi l'énoncé impose-t-il $n \leq 9$? Que se passerait-il pour `n = 12` ?

Indication

La partie descendante de la ligne peut s'obtenir avec `range(i - 1, 0, -1)`, qui engendre les entiers de `i-1` jusqu'à `1` en décroissant.

9 Défi 2 : l'ordinateur devine votre nombre

Exercice 9: L'ordinateur devine votre nombre

On inverse les rôles du jeu du « plus petit / plus grand » : c'est **vous** qui choisissez mentalement un nombre entier entre 1 et 100, et c'est le **programme** qui doit le deviner. À chaque tour, le programme propose un nombre, et vous répondez en tapant :

- `+` si le nombre cherché est plus grand que la proposition ;
- `-` s'il est plus petit ;
- `=` si c'est le bon.

1. Écrire un programme qui procède ainsi, en proposant à chaque tour le **milieu** de l'intervalle des valeurs encore possibles. Le programme s'arrête dès que vous répondez **=**, et affiche le nombre de propositions qu'il lui a fallu.
2. Tester le programme en pensant à 1, puis à 100, puis à 37. Quel est le nombre **maximal** de propositions nécessaires ? Comparer à un programme qui essaierait **1**, puis **2**, puis **3**, etc.
3. Que se passe-t-il si vous trichez, c'est-à-dire si vos réponses sont incohérentes ? Modifier le programme pour qu'il détecte cette situation et affiche « Vous avez triché ! ».

Indication

Le programme doit garder en mémoire deux bornes, **mini** et **maxi**, entre lesquelles le nombre cherché se trouve encore. Une réponse **+** signifie que la proposition est trop petite : toutes les valeurs jusqu'à la proposition comprise sont éliminées.

10 Défi 3 : le programme mystère

Exercice 10: Le programme mystère

On considère le programme suivant :

```
1 a = 13
2 b = 11
3 resultat = 0
4 while a > 0:
5     if a % 2 == 1:
6         resultat = resultat + b
7     a = a // 2
8     b = b * 2
9 print(resultat)
```

1. Tracer l'exécution complète dans un tableau à quatre colonnes : **a**, **b**, **resultat**, et le test **a % 2 == 1**. Qu'affiche le programme ?
2. Refaire la trace avec **a = 6** et **b = 5**, puis avec **a = 4** et **b = 7**. Que constate-t-on ?
3. Formuler une conjecture : que calcule ce programme ?
4. Ce programme n'utilise jamais l'opérateur ***** entre **a** et **b**. Expliquer, en observant la colonne **b** de votre trace, comment il parvient malgré tout à ce résultat.
5. Combien de tours de boucle sont effectués pour **a = 13** ? Pour **a = 1000** ? Comparer avec un programme qui calculerait le même résultat en ajoutant **b** à lui-même **a** fois.