Feuille d'exercices – Les fonctions

Feuille débranchée (sans ordinateur)

NSI – Première – Séquence 3 : Les fonctions

Cette feuille reprend, sans ordinateur, les notions vues dans le cours de cette séquence : elle vous permet de vérifier que vous les maîtrisez avant de passer aux TP. Tous les exercices se résolvent avec un stylo et un peu de raisonnement : aucun code ne doit être exécuté.

Partie	Objectif	Durée indicative
A – Fonctions, print et return	Vocabulaire, tracer, corriger	20 min
B – Portée des variables	Prédire et corriger une erreur	15 min
C – Décomposer, spécifier, tester	Concevoir des fonctions	20 min
D – Bibliothèques et combinaisons	Corriger un import, combiner	15 min

1 Fonctions, print et return

Exercice 1: Vrai ou faux ?

Pour chacune des affirmations suivantes, dire si elle est **vraie** ou **fausse**, et justifier en une phrase.

1. Une fonction peut avoir plusieurs paramètres.
2. Une fonction doit obligatoirement contenir un `return`.
3. `print(x)` et `return x` produisent exactement le même effet.
4. Le mot **paramètre** désigne le nom utilisé dans la définition de la fonction, et le mot **argument** la valeur donnée lors de l'appel.
5. Si une fonction ne contient aucun `return`, elle renvoie automatiquement la valeur `None`.

Correction de l'exercice 1

1. **Vrai** : une fonction peut prendre autant de paramètres que nécessaire (ou aucun).
2. **Faux** : une fonction peut très bien ne contenir aucun `return` ; elle renvoie alors `None`.
3. **Faux** : `print` affiche une valeur à l'écran sans la transmettre, alors que `return` la transmet réellement pour qu'elle soit réutilisable.
4. **Vrai**, c'est exactement la distinction vue en cours.
5. **Vrai**, `None` est la valeur par défaut renvoyée par une fonction sans `return`.

Exercice 2: Tracer l'exécution

On considère la fonction suivante :

```
1  def f(a, b):
2      total = a + b
3      diff = a - b
4      return total * diff
5
6  print(f(5, 2))
7  print(f(3, 3))
```

Donner la valeur renvoyée par chacun des deux appels, en détaillant le calcul.

Correction de l'exercice 2

Pour `f(5, 2)` : `total = 5 + 2 = 7` et `diff = 5 - 2 = 3`, donc la fonction renvoie $7 \times 3 = \mathbf{21}$.
Pour `f(3, 3)` : `total = 3 + 3 = 6` et `diff = 3 - 3 = 0`, donc la fonction renvoie $6 \times 0 = \mathbf{0}$.

Exercice 3: Corriger un piège classique

Le code suivant doit calculer `4 * 4 - 1 = 15`, puis y ajouter `10`, mais il provoque une erreur :

```
1 def carre_moins_un(x):
2     print(x * x - 1)
3
4 y = carre_moins_un(4) + 10
5 print(y)
```

1. Expliquer précisément pourquoi ce code ne fonctionne pas comme prévu.
2. Corriger la fonction `carre_moins_un` pour que le programme affiche bien `25`.

Correction de l'exercice 3

1. `carre_moins_un(4)` affiche `15` à l'écran, mais ne le **renvoie** pas (aucun `return`) : elle renvoie donc `None`. La ligne `y = carre_moins_un(4) + 10` tente alors de calculer `None + 10`, ce qui provoque une erreur (`TypeError`).
2. Il suffit de remplacer `print` par `return` :

```
1 def carre_moins_un(x):
2     return x * x - 1
3
4 y = carre_moins_un(4) + 10
5 print(y)   # affiche 25
```

2 Portée des variables

Exercice 4: Vrai ou faux ?

Pour chacune des affirmations suivantes, dire si elle est **vraie** ou **fausse**, et justifier en une phrase.

1. Une variable créée à l'intérieur d'une fonction reste accessible depuis le programme principal, après l'appel de la fonction.
2. Deux fonctions différentes peuvent utiliser un paramètre du même nom sans se perturber l'une l'autre.
3. Une fonction peut lire une variable globale.
4. Modifier une variable globale depuis l'intérieur d'une fonction, à l'aide du mot-clé `global`, est une pratique recommandée.

Correction de l'exercice 4

1. **Faux** : une variable créée à l'intérieur d'une fonction est **locale** ; elle disparaît dès que l'exécution de la fonction se termine.
2. **Vrai** : chaque fonction a ses propres variables locales, indépendantes des variables locales des autres fonctions.
3. **Vrai** : la **lecture** d'une variable globale depuis une fonction est autorisée et ne pose pas de problème.
4. **Faux** : c'est une pratique à **éviter**, car elle rend le comportement de la fonction dépendant de variables invisibles dans sa définition. On préfère toujours passer par les paramètres et le `return`.

Exercice 5: Prédire et corriger une erreur de portée

On considère le code suivant :

```
1  score = 0
2
3  def ajouter_point():
4      score = score + 1
5      return score
6
7  print(ajouter_point())
```

1. Que se passe-t-il à l'exécution de ce programme ? Justifier précisément en utilisant le vocabulaire de **portée** vu en cours.
2. Proposer une version corrigée de `ajouter_point` qui renvoie bien `score + 1`, en utilisant un **paramètre**.

Correction de l'exercice 5

1. Ce programme provoque une erreur (`UnboundLocalError`). La ligne `score = score + 1` crée `score` comme variable **locale** à `ajouter_point` (car on lui affecte une valeur à l'intérieur de la fonction) ; mais au moment de calculer `score + 1`, Python cherche la valeur de cette variable locale `score`, qui n'a pas encore été définie. La variable globale `score = 0` n'est donc pas utilisée ici.
2. On utilise un paramètre, ce qui évite toute ambiguïté :

```
1  score = 0
2
3  def ajouter_point(score):
4      return score + 1
5
6  score = ajouter_point(score)
7  print(score)   # affiche 1
```

3 Décomposer, spécifier, tester

Exercice 6: Repérer une duplication et concevoir une fonction

On considère le code suivant :

```python
cote1 = 3
volume1 = cote1 * cote1 * cote1
print("Volume du cube 1 :", volume1)

cote2 = 5
volume2 = cote2 * cote2 * cote2
print("Volume du cube 2 :", volume2)

cote3 = 2
volume3 = cote3 * cote3 * cote3
print("Volume du cube 3 :", volume3)
```

1. Quel bloc de code est répété, presque à l'identique, trois fois ?
2. **Sans écrire de code Python**, proposer une fonction qui éviterait cette répétition : donner son nom, la liste de ses paramètres, et une phrase décrivant ce qu'elle renvoie.
3. Réécrire le programme en remplaçant les trois blocs dupliqués par des appels à votre fonction (vous pouvez écrire ces appels en vrai Python, ou sous forme d'appels « `nom_fonction(...)` » si vous préférez).

Correction de l'exercice 6

1. Le calcul `cote * cote * cote` et l'affichage « Volume du cube... » sont répétés pour les trois cubes ; seul le côté change à chaque fois.
2. Par exemple : une fonction `volume_cube(cote)`, qui prend en paramètre le côté du cube (un nombre), et qui renvoie le volume de ce cube (`cote * cote * cote`).
3.

```python
def volume_cube(cote):
    return cote * cote * cote

print("Volume du cube 1 :", volume_cube(3))
print("Volume du cube 2 :", volume_cube(5))
print("Volume du cube 3 :", volume_cube(2))
```

Exercice 7: Associer un prototype à sa docstring

Voici trois prototypes de fonctions (A, B, C) et trois docstrings (1, 2, 3), dans le désordre. Associer chaque prototype à la docstring qui lui correspond.

Prototypes :

A. `aire_disque(rayon: float) -> float`

B. `est_positif(n: int) -> bool`

C. `initiale(prenom: str) -> str`

Docstrings :

1. « Renvoie `True` si `n` est strictement positif, `False` sinon. »
2. « Renvoie l'aire d'un disque de rayon donné (formule : `pi * rayon**2`). »
3. « Renvoie la première lettre de `prenom` , en majuscule. »

Correction de l'exercice 7

A – 2 (une aire de disque est un nombre décimal, comme le type renvoyé `float`) ; B – 1 (un test renvoie un booléen, `bool`) ; C – 3 (une lettre est une chaîne de caractères, `str`).

Exercice 8: Un jeu de tests incomplet

On considère la fonction `est_majeur(age)`, qui renvoie `True` si `age` est supérieur ou égal à 18, et `False` sinon. Un élève propose le jeu de tests suivant :

```
1 assert est_majeur(20) == True
2 assert est_majeur(15) == False
3 assert est_majeur(20) == True
```

1. Ce jeu de tests contient un test **inutile**. Lequel, et pourquoi ?
2. Un cas important, à la limite entre les deux réponses possibles, n'a pas été testé. Lequel ? Que doit-il renvoyer ?

Correction de l'exercice 8

1. Le test `assert est_majeur(20) == True` est écrit **deux fois** : le second n'apporte aucune information supplémentaire par rapport au premier.
2. Il manque le **cas limite** `age = 18` : d'après l'énoncé (« supérieur ou égal à 18 »), `est_majeur(18)` doit renvoyer `True`. C'est exactement ce genre de valeur limite qu'il faut penser à tester en priorité.

4 Bibliothèques et combinaisons

Exercice 9: Corriger une erreur d'import

Le code suivant provoque une erreur :

```
1 import math
2
3 rayon = 5
4 resultat = math.pi * rayon ** 2
5 print(sqrt(resultat))
```

1. Identifier précisément l'erreur.
2. Corriger ce code de **deux façons différentes**.

Correction de l'exercice 9

1. Avec `import math`, la fonction `sqrt` n'est **pas** importée directement dans l'espace de noms principal : elle n'est accessible que via `math.sqrt`. L'appel `sqrt(resultat)` provoque donc une erreur (`NameError`), `sqrt` seul n'existant pas.
2. Deux corrections possibles :

```
1 print(math.sqrt(resultat))
```

ou bien, en changeant l'import tout en haut du fichier :

```python
from math import sqrt
# ...
print(sqrt(resultat))
```

Exercice 10: Combiner des fonctions déjà écrites (sans coder)

On donne trois fonctions déjà écrites et déjà testées, sans connaître leur code – seulement ce qu'elles font :

- `majuscules(texte)` : transforme `texte` en le même texte, tout en majuscules.
- `compter_lettres(texte)` : transforme `texte` en le nombre de caractères qu'il contient (un entier).
- `message_longueur(n)` : transforme un entier `n` en le texte « Cela fait n caractère(s). »

En **combinant deux** de ces trois fonctions (sans en écrire de nouvelle, et sans utiliser la troisième), quel texte obtient-on à partir de l'entrée `"nsi"` ? Détailler les étapes.

Correction de l'exercice 10

On combine `compter_lettres` puis `message_longueur` : `compter_lettres("nsi")` renvoie `3` (trois caractères), puis `message_longueur(3)` renvoie « Cela fait 3 caractère(s). »
La fonction `majuscules` n'est pas utile ici : elle transforme un texte en un autre texte, ce qui ne permet pas d'obtenir un nombre de caractères. Savoir repérer, parmi plusieurs fonctions disponibles, celles qui sont réellement utiles pour une combinaison donnée est une compétence tout aussi importante que savoir les combiner – vous la retrouverez en construisant la bibliothèque du TP final.

Remarque : Ce qu'il faut retenir de cette feuille

Cette feuille a couvert l'ensemble du cours : la différence entre `print` et `return`, la portée des variables, la décomposition d'un programme en fonctions, la spécification (prototype, docstring), les jeux de tests, et l'usage de bibliothèques. Si certains exercices vous ont posé des difficultés, relisez la section correspondante du cours avant de commencer le premier TP.