Les fonctions en Python

Résumé

Imaginons qu'il faille calculer l'aire d'un rectangle à dix endroits différents dans un programme. Sans fonction, il faudrait recopier la même formule dix fois, avec le risque de se tromper à chaque copie. Une **fonction** est un bloc de code réutilisable : on lui donne un nom, des entrées (les paramètres) et elle renvoie un résultat. C'est le moyen fondamental d'organiser un programme, d'éviter les répétitions et de rendre le code lisible. En NSI, presque tout repose sur les fonctions : les algorithmes de tri, de recherche, les structures de données sont tous écrits sous forme de fonctions.

Prérequis : Les types de base.

① Mémo

Définir une fonction

```
def nom_de_la_fonction(parametre1, parametre2):
    instructions
    return resultat
```

- Le mot-clé `def` introduit la définition.
- Les **paramètres** (ou paramètres formels) sont les entrées de la fonction.
- Le mot-clé `return` renvoie le résultat à l'appelant.
- Le corps de la fonction est indenté (quatre espaces).

Appeler une fonction

```
resultat = nom_de_la_fonction(valeur1, valeur2)
```

- Les valeurs passées lors de l'appel sont les **paramètres effectifs**.
- La valeur renvoyée par `return` peut être stockée dans une variable.
- Si la fonction ne contient pas de `return`, elle renvoie `None`.

Définition 1 (Différence entre `return` et `print`)

- `return` renvoie une valeur : on peut la stocker, la réutiliser dans un calcul.
- `print` affiche à l'écran mais ne renvoie rien (`None`).

Exemple. `print(abs(-5))` fonctionne car `abs` *renvoie* 5. Si `abs` ne faisait qu'afficher, on ne pourrait pas utiliser le résultat.

Propriété 1 (Variables locales et portée)

- Une variable créée dans une fonction est **locale** : elle n'existe que pendant l'exécution de la fonction.
- Une variable définie en dehors de toute fonction est **globale** : elle est accessible partout, mais il est déconseillé de la modifier dans une fonction.
- Deux fonctions peuvent utiliser le même nom de variable sans conflit.

Fonctions prédéfinies utiles

`print()` Affiche à l'écran.
`input()` Demande une saisie (renvoie toujours un `str`).
`type()` Renvoie le type d'une valeur.
`len()` Renvoie la longueur d'une chaîne ou d'une séquence.
`int()`, `float()`, `str()` Conversions de type.
`abs()` Valeur absolue.
`round(x, n)` Arrondi de `x` à `n` décimales.
`max()`, `min()` Plus grande et plus petite valeur.

Pièges fréquents

- Oublier `return` : la fonction renvoie alors `None`.
- Confondre `return` et `print` : écrire `print` là où il faudrait `return` empêche de réutiliser la valeur.
- Oublier les parenthèses à l'appel : `ma_fonction` désigne la fonction elle-même, `ma_fonction()` l'exécute.
- Placer du code après `return` : il ne sera jamais exécuté.

Erreurs classiques

Code erroné	Code correct	Explication

`def f(L=[]):`
`L.append(1)`
`return L`
	`def f(L=None):`
	`if L is None:`
	`L = []`
		Un paramètre par défaut mutable (liste, dictionnaire) est partagé entre tous les appels : il faut utiliser `None`.

`f = carre` (sans parenthèses)	`f = carre(5)`	Sans parenthèses, on obtient l'objet fonction, pas son résultat.
`print(carre(5))` au lieu de `return`	`return x * x`	`print` affiche mais ne renvoie rien : la fonction retourne `None` et le résultat ne peut pas être réutilisé.
`carre(5, 3)` avec `def carre(x):`	`carre(5)`	Trop de paramètres effectifs provoque un `TypeError` : respecter le nombre de paramètres.

② Exemples

Programme 1 · Aire d'un disque

```python
import math

def aire_disque(rayon):
    return math.pi * rayon ** 2

print(aire_disque(5))     # 78.539...
print(aire_disque(1))     # 3.14159...
```

Programme 2 · Conversion Celsius-Fahrenheit

```python
def celsius_vers_fahrenheit(c):
    return c * 9 / 5 + 32

def fahrenheit_vers_celsius(f):
    return (f - 32) * 5 / 9

print(celsius_vers_fahrenheit(100))   # 212.0
print(fahrenheit_vers_celsius(68))    # 20.0
```

Programme 3 · Fonction à plusieurs paramètres

```python
def prix_ttc(prix_ht, taux_tva):
    return round(prix_ht * (1 + taux_tva / 100), 2)

print(prix_ttc(100, 20))    # 120.0
print(prix_ttc(49.99, 5.5)) # 52.74
```

Programme 4 · Différence return vs print

```python
def double_return(n):
    return n * 2

def double_print(n):
    print(n * 2)

a = double_return(5)   # a vaut 10
b = double_print(5)    # affiche 10, mais b vaut None

print(a + 1)            # 11 (on peut calculer avec a)
# print(b + 1)          # ERREUR : None + 1 impossible
```

Programme 5 · Variables locales

```
def perimetre_rectangle(longueur, largeur):
    p = 2 * (longueur + largeur)    # p est locale
    return p

print(perimetre_rectangle(5, 3))    # 16
# print(p)   # ERREUR : p n'existe pas en dehors de la fonction
```

Programme 6 · Composition de fonctions

```
def distance(x1, y1, x2, y2):
    return ((x2 - x1) ** 2 + (y2 - y1) ** 2) ** 0.5

def milieu(x1, y1, x2, y2):
    return ((x1 + x2) / 2, (y1 + y2) / 2)

print(distance(0, 0, 3, 4))   # 5.0
print(milieu(0, 0, 6, 8))     # (3.0, 4.0)
```

③ Exercices

Exercice 1 *Premières fonctions* Écrire les fonctions suivantes :

1. `cube(n)` qui renvoie n^3.
2. `initiales(prenom, nom)` qui renvoie les initiales en majuscules. Par exemple, `initiales("ada", "lovelace")` renvoie `"A.L."`.

Exercice 2 *Conversions d'unités*

1. Écrire une fonction `km_vers_miles(km)` sachant que 1 mile ≈ 1,609 km.
2. Écrire une fonction `convertir_duree(secondes)` qui renvoie un tuple `(heures, minutes, secondes)`.
3. Tester les fonctions : vérifier que `km_vers_miles(1.609)` renvoie environ `1.0` et que `convertir_duree(3725)` renvoie `(1, 2, 5)`.

Exercice 3 *Return ou print* Sans exécuter le code, prédire ce qui est affiché par chaque programme :

```
# Programme A
def mystere(x):
    x * 2 + 1

print(mystere(5))

# Programme B
def calcul(a, b):
    return a + b
    print("Terminé")

resultat = calcul(3, 4)
print(resultat)

# Programme C
def afficher_double(n):
    print(n * 2)

x = afficher_double(6)
print(x)
```

Exercice 4 *Géométrie*

1. Écrire une fonction `aire_triangle(base, hauteur)` qui renvoie l'aire d'un triangle.
2. Écrire une fonction `distance(x1, y1, x2, y2)` qui renvoie la distance entre deux points du plan.
3. Écrire une fonction `perimetre_triangle(x1, y1, x2, y2, x3, y3)` qui utilise la fonction `distance` pour calculer le périmètre d'un triangle défini par ses trois sommets.

Solutions des exercices

Corrigé de l'exercice 1.

```python
def cube(n):
    return n ** 3

def initiales(prenom, nom):
    return prenom[0].upper() + "." + nom[0].upper() + "."
```

Vérification :

- `cube(3)` renvoie `27` car $3^3 = 27$;
- `cube(-2)` renvoie `-8` car $(-2)^3 = -8$;
- `initiales("ada", "lovelace")` renvoie `"A.L."`.

Corrigé de l'exercice 2.

```python
def km_vers_miles(km):
    return round(km / 1.609, 2)

def convertir_duree(secondes):
    h = secondes // 3600
    m = (secondes % 3600) // 60
    s = secondes % 60
    return (h, m, s)
```

Vérification :

- `km_vers_miles(1.609)` renvoie `1.0` ;
- `km_vers_miles(42.195)` renvoie `26.22` (un marathon) ;
- `convertir_duree(3725)` : 3725//3600 = 1, reste 125, 125//60 = 2, reste 5, soit `(1, 2, 5)`.

On vérifie : $1 \times 3600 + 2 \times 60 + 5 = 3725$. ✓

Corrigé de l'exercice 3.

- **Programme A :** affiche `None`. La ligne `x * 2 + 1` calcule le résultat mais ne le renvoie pas (`return` oublié).
- **Programme B :** affiche `7`. Le `return` interrompt la fonction immédiatement : la ligne `print("Terminé")` n'est jamais exécutée.
- **Programme C :** affiche d'abord `12` (le `print` dans la fonction), puis `None` (car la fonction ne contient pas de `return`, donc `x` vaut `None`).

Trace d'exécution du programme C :

Étape	Action	Affichage
1	Appel `afficher_double(6)`	
2	`print(6 * 2)` s'exécute dans la fonction	`12`
3	Pas de `return`, la fonction renvoie `None`	
4	`x = None`	
5	`print(x)`	`None`

Corrigé de l'exercice 4.

```python
def aire_triangle(base, hauteur):
    return base * hauteur / 2

def distance(x1, y1, x2, y2):
    return ((x2 - x1) ** 2 + (y2 - y1) ** 2) ** 0.5

def perimetre_triangle(x1, y1, x2, y2, x3, y3):
    d1 = distance(x1, y1, x2, y2)
    d2 = distance(x2, y2, x3, y3)
    d3 = distance(x3, y3, x1, y1)
    return d1 + d2 + d3
```

Vérification :

- `aire_triangle(6, 4)` renvoie `12.0` car $\frac{6 \times 4}{2} = 12$;
- `distance(0, 0, 3, 4)` renvoie `5.0` car $\sqrt{9+16} = 5$;
- `perimetre_triangle(0, 0, 3, 0, 0, 4)` renvoie `12.0` car $3 + 4 + 5 = 12$ (triangle rectangle 3-4-5).

Principe de composition : la fonction `perimetre_triangle` appelle trois fois `distance` au lieu de dupliquer le calcul. C'est un avantage majeur des fonctions : on écrit la formule une seule fois.