

Rappels sur les fonctions en Python

Fiche de synthèse

① DÉFINIR ET APPELER UNE FONCTION

Définition 1 (Fonction) Une **fonction** est un bloc d'instructions nommé, défini avec le mot-clé `def`, que l'on exécute en l'appelant par son nom suivi de parenthèses. Elle peut recevoir des paramètres et renvoyer une valeur avec l'instruction `return`.

```
def carre(x):
    return x * x

print(carre(5)) # affiche 25
```

Une valeur **renvoyée** par `return` peut être stockée dans une variable ou réutilisée dans un calcul; un **affichage** par `print` ne fait que l'écrire à l'écran. Une fonction qui se termine sans `return` renvoie la valeur spéciale `None`, qui signifie « aucune valeur ».

② PORTÉE DES VARIABLES

Définition 2 (Variable locale) Une variable affectée dans le corps d'une fonction est **locale** : elle n'existe que pendant l'exécution de la fonction et n'est pas visible en dehors.

```
x = 10
y = 10

def une_fonction():
    x = 5 # x locale, x global intact
    global y
    y = 5 # réaffecte le y global

une_fonction()
print(x) # affiche 10
print(y) # affiche 5
```

Seul le mot-clé `global` permet de réaffecter une variable globale; on l'évite autant que possible. En revanche, une liste ou un dictionnaire global peut être modifié en place sans `global`, par exemple avec `L.append(3)` ou `d[cle] = valeur`.

③ PARAMÈTRES

```
def presentation(prenom, age=15, ville="Lyon"):
    print(f"{prenom}, {age} ans, {ville}")

presentation("Gaudric") # Gaudric, 15 ans, Lyon
presentation("Justine", 21, "Paris")
```

```
presentation("Audrey", ville="Toulouse")
```

- Les **paramètres positionnels** reçoivent les valeurs dans l'ordre de la définition : "Justine", puis 21, puis "Paris".
- Un **argument nommé** s'écrit `nom=valeur` à l'appel : on peut alors donner les paramètres dans n'importe quel ordre.
- Un **paramètre par défaut** a une valeur fixée dans la définition (`age=15`) : on peut l'omettre à l'appel.

④ FONCTIONS EN PARAMÈTRE

On peut passer une fonction en paramètre à une autre fonction, en écrivant son nom sans parenthèses.

```
def appliquer(f, mots):
    return [f(mot) for mot in mots]

print(appliquer(len, ["Les", "mots"]))
# affiche [3, 4]
```

⑤ PRÉCONDITIONS ET DOCUMENTATION

Définition 3 (Assertion) L'instruction `assert condition, "message"` vérifie une condition; si elle est fausse, le programme s'arrête sur une erreur `AssertionError`, avec le message s'il est fourni.

```
def est_une_voyelle(lettre):
    """Renvoie True si lettre, chaîne de
    longueur 1, est une voyelle, False sinon."""
    assert isinstance(lettre, str)
    assert len(lettre) == 1, "une seule lettre"
    return lettre in "aeiouy"
```

`isinstance(v, T)` renvoie `True` si la valeur `v` est de type `T`. La **docstring**, chaîne placée en première ligne du corps, décrit le rôle de la fonction, ses paramètres et sa valeur renvoyée; `help(est_une_voyelle)` l'affiche.

📌 À retenir.

- `return` renvoie une valeur réutilisable; `print` ne fait qu'afficher.
- Une variable affectée dans une fonction est locale; `global` est à éviter.
- Paramètres positionnels, arguments nommés et paramètres par défaut se combinent à l'appel.
- `assert` vérifie les préconditions; la docstring documente la fonction.