

Fonctions

Fiche de synthèse

① DÉFINIR ET APPELER

Définition 1 (Fonction) Une **fonction** regroupe des instructions sous un nom. On la **définit** une fois avec `def`, puis on l'**appelle** autant de fois qu'on veut en écrivant son nom suivi de parenthèses.

Les fonctions évitent de recopier du code et découpent un problème en sous-problèmes. La définition n'exécute rien : le corps de la fonction ne s'exécute qu'à l'appel.

```
def saluer(prenom):
    print(f"Bonjour {prenom} !")

saluer("Alice")      # Bonjour Alice !
saluer("Bob")        # Bonjour Bob !
```

Les **paramètres** sont les noms écrits dans la définition, ici `prenom`; les **arguments** sont les valeurs données à l'appel, ici `"Alice"`. Plusieurs paramètres se séparent par des virgules.

② RENVoyer UNE VALEUR : RETURN

```
def carre(n):
    return n ** 2

resultat = carre(5)      # 25
print(carre(3) + 1)      # 10

def valeur_absolue(n):
    if n >= 0:
        return n
    return -n
```

- `return` renvoie une valeur et **termine** la fonction : la suite du corps est ignorée.
- Une fonction sans `return` renvoie la valeur `None`.
- Une fonction qui **affiche** avec `print` ne renvoie rien d'utilisable; une fonction qui **renvoie** une valeur permet de la réutiliser dans un calcul. On préfère presque toujours renvoyer.

③ VALEURS PAR DÉFAUT ET ARGUMENTS NOMMÉS

```
def saluer(prenom, formule="Bonjour"):
    print(f"{formule} {prenom} !")

saluer("Alice")          # Bonjour Alice !
saluer("Alice", "Bonsoir") # Bonsoir Alice !
saluer(formule="Salut", prenom="Bob")
```

Un paramètre qui a une **valeur par défaut** peut être omis; il se place après les paramètres obligatoires. À l'appel, un **argument nommé** précise le paramètre et permet de changer l'ordre.

④ PORTÉE DES VARIABLES

- Une variable créée dans une fonction est **locale** : elle n'existe que pendant l'appel. L'utiliser ensuite provoque l'erreur `NameError`.
- Une variable créée hors de toute fonction est **globale** : une fonction peut la lire.
- Une variable locale qui porte le nom d'une variable globale la masque sans la modifier.

```
x = 100
def modifier():
    x = 0          # x locale
modifier()
print(x)          # 100 : x global inchangé
```

⑤ DOCUMENTER ET TESTER

```
def moyenne(a: float, b: float) -> float:
    """Renvoie la moyenne de a et de b.

    >>> moyenne(12, 15)
    13.5
    """
    return (a + b) / 2

assert moyenne(12, 15) == 13.5
assert moyenne(0, 0) == 0
```

- La **docstring**, chaîne placée juste après la ligne `def`, décrit la fonction; `help(moyenne)` l'affiche.
- Les **annotations** `a: float` et `-> float` indiquent les types attendus; Python ne les vérifie pas.
- `assert condition` ne fait rien si la condition est vraie, et arrête le programme sur une erreur `AssertionError` sinon. On teste ainsi la fonction sur des exemples dont on connaît le résultat.

📌 À retenir.

- `def` définit une fonction, l'appel l'exécute.
- `return` renvoie une valeur et termine la fonction; sans `return`, la fonction renvoie `None`.
- Les variables créées dans une fonction sont locales.
- On documente avec une docstring et on teste avec `assert`.