4. Fonctions.

1. Définition des fonctions.

Une fonction est un bloc de code réutilisable qui effectue une tâche spécifique. Elle permet de structurer un programme en le divisant en sections logiques.

Syntaxe de base :

```
def nom_de_fonction(): # Instructions de la fonction pass
```

Exemple :

```
def saluer(): print("Bonjour !") saluer()
```

Sortie :

```
Bonjour !
```

---

2. Paramètres.

Les paramètres permettent de transmettre des données à une fonction. Ce sont des variables définies dans la signature de la fonction.

Exemple :

```
def saluer_utilisateur(nom): print(f"Bonjour, {nom} !") saluer_utilisateur("Alice")
```

Sortie :

```
Bonjour, Alice !
```

---

3. Retour de valeurs.

Une fonction peut renvoyer une valeur à l'aide de l'instruction `return`.

Exemple :

```
def addition(a, b): return a + b resultat = addition(3, 5) print(resultat)
```

Sortie :

```
8
```

---

4. Utilisation de plusieurs paramètres.

Une fonction peut accepter plusieurs paramètres, séparés par des virgules.

Exemple :

```
def multiplier(a, b, c): return a * b * c print(multiplier(2, 3, 4))
```

Sortie :

```
24
```

---

5. Nommer les fonctions.

- Les noms de fonctions doivent être **clairs et significatifs**.
- Utiliser des caractères minuscules et le séparateur `_` pour les noms composés (convention snake_case).
- **Exemple** : `calculer_somme`, `afficher_message`.

---

6. Portée (globale et locale).

- **Portée locale** : Les variables définies dans une fonction sont accessibles uniquement dans cette fonction.
- **Portée globale** : Une variable définie hors de toute fonction est accessible partout.

Exemple :

```
def fonction_locale(): local = 10 # Variable locale print(local) global_var = 20 # Varia
ble globale def fonction_globale(): print(global_var) fonction_locale() fonction_globale
()
```

Sortie :

```
10 20
```

---

7. Fonction avec une condition.

Une fonction peut contenir des instructions conditionnelles comme `if`, `else`, et `elif`.

Exemple :

```
def verifier_parite(nombre): if nombre % 2 == 0: return "Pair" else: return "Impair" pri
nt(verifier_parite(4)) print(verifier_parite(7))
```

Sortie :

```
Pair Impair
```

---

8. Fonction avec une boucle.

Une fonction peut inclure des boucles pour effectuer des tâches répétitives.

Exemple :

```
def afficher_nombres(maximum): for i in range(1, maximum + 1): print(i) afficher_nombres
(5)
```

Sortie :

```
1 2 3 4 5
```

---

9. Fonction avec une liste en argument.

Les fonctions peuvent prendre des listes en argument et les manipuler.

Exemple :

```
def somme_liste(nombres): return sum(nombres) ma_liste = [1, 2, 3, 4, 5] print(somme_lis
te(ma_liste))
```

Sortie :

```
15
```

Exercices pratiques.

1. **Exercice 1** : Écrivez une fonction `carre` qui prend un nombre comme argument et retourne son carré.
   - Exemple : `carre(4)` doit retourner `16`.

```
def carre(n): return n*n n = carre(5) print(n)
```

2. **Exercice 2** : Créez une fonction `max_de_trois` qui prend trois nombres en paramètre et retourne le plus grand.

```
def max_de_trois(a, b, c): return max(a, b, c) n = max_de_trois(1062, 702, 2025)
print(n)
```

1. **Exercice 3** : Écrivez une fonction `filtrer_pairs` qui prend une liste de nombres en argument et retourne une nouvelle liste contenant uniquement les nombres pairs.

```
bus = [210, 325, 56, 46, 124, 318, 115, 118] def filter_pairs(bus): pairs = [n for n in
liste if n % 2 == 0] return pairs nombre_de_pairs = filter_pairs(bus) print(f"Voici les
nombres pairs : {nombre_de_pairs}")
```

1. **Exercice 4** : Développez une fonction `compter_occurrences` qui prend une liste et un élément comme arguments, et retourne le nombre de fois que cet élément apparaît dans la liste.

Résumé.

Concept	Description	Exemple
Définition	Blocs de code réutilisables pour effectuer une tâche spécifique.	`def saluer(): print("Bonjour !") saluer()`
Paramètres	Variables passées à une fonction pour personnaliser son exécution.	`def saluer_utilisateur(nom): print(f"Bonjour, {nom} !")`
Retour de valeurs	Une fonction peut renvoyer une valeur avec `return`.	`def addition(a, b): return a + b`
Utilisation de plusieurs paramètres	Une fonction peut accepter plusieurs arguments séparés par des virgules.	`def multiplier(a, b, c): return a * b * c`
Nommer les fonctions	Utiliser des noms significatifs, en minuscules, avec `_` pour séparer les mots (snake_case).	`def calculer_somme(): pass`
Portée locale	Les variables déclarées dans une fonction sont locales et accessibles uniquement dans celle-ci.	`def f(): local = 10 print(local)`
Portée globale	Les variables déclarées hors d'une fonction sont accessibles dans tout le programme.	`global_var = 20 def f(): print(global_var)`
Fonction avec une condition	Inclut des `if`, `else`, ou `elif` pour prendre des décisions dans la fonction.	`def verifier_parite(n): return "Pair" if n % 2 == 0 else "Impair"`
Fonction avec une boucle	Peut inclure une boucle `for` ou `while` pour effectuer des tâches répétitives.	`def afficher_nombres(n): for i in range(1, n+1): print(i)`
Liste en argument	Une fonction peut accepter une liste et la manip… directement	`def somme_liste(lst): return`