Fiche Memo Python : Fonctions, Scope, Listes, *args/**kwargs, Lambda & Dictionnaires

1. Utilisation des fonctions

---

- Definition d'une fonction :

  Utilisez le mot-cle def pour declarer une fonction.

  Ex :

```
def ma_fonction(param1, param2):
    # instructions
    return resultat
```

- Appel de fonction (Call) :

  Appelez une fonction en utilisant son nom suivi de parentheses et des arguments.

  Ex :

```
resultat = ma_fonction(10, 20)
```

- Arguments nommes (Keyword arguments) :

  Permettent de preciser quels arguments correspondent a quels parametres.

  Ex :

```
ma_fonction(param2=20, param1=10)
```

- Valeurs par defaut (Default values) :

  Attribuez une valeur par defaut a un parametre.

  Ex :

```
def ma_fonction(param1, param2=5):
    return param1 + param2
```

2. Scope dans les fonctions

---

- Scope local : Les variables definies dans une fonction sont accessibles uniquement dans celle-ci.

- Scope global : Les variables definies a l'exterieur des fonctions sont globales.

- Declaration explicite :

  Utilisez le mot-cle global pour modifier une variable globale.

  Ex :

```
compteur = 0
def incremente():
    global compteur
    compteur += 1
```

3. Le mot-cle return

---

- Retourner une valeur :

  return met fin a l'execution de la fonction et renvoie une valeur.

  Ex :

```
def somme(a, b):
    return a + b
```

- Retour multiple :

  Retournez plusieurs valeurs sous forme de tuple.

  Ex :

```
def operations(a, b):
    return a + b, a * b
```

4. Passing list as function arguments & Modifying a list in a function

---

- Passage d'une liste :

  Les listes sont passees par reference et peuvent etre modifiees.

  Ex :

```
def ajoute_element(liste, element):
    liste.append(element)


ma_liste = [1, 2, 3]
ajoute_element(ma_liste, 4)
# ma_liste devient [1, 2, 3, 4]
```

- Reprendre une nouvelle liste :

  Pour eviter de modifier l'originale, copiez la liste.

  Ex :

```
def ajoute_element_copie(liste, element):
    nouvelle_liste = liste.copy()
    nouvelle_liste.append(element)
    return nouvelle_liste
```

5. *args, **kwargs et parametres variadiques

---

- *args :

  Permet de passer un nombre variable d'arguments positionnels.

  Ex :

```
def addition(*args):
    return sum(args)
```

```
print(addition(1, 2, 3))  # renvoie 6
```

- **kwargs :

  Permet de passer un nombre variable d'arguments nommes.

  Ex :

```
def afficher_info(**kwargs):
    for cle, valeur in kwargs.items():
        print(f"{cle} : {valeur}")
afficher_info(nom="Alice", age=30)
```

- Combinaison :

  Vous pouvez combiner *args et **kwargs.

  Ex :

```
def fonction_mixte(*args, **kwargs):
    print(args)
    print(kwargs)
```

6. Fonctions Lambda, Map, Reduce & Filter

---

- Fonctions lambda :

  Fonctions anonymes en une seule ligne.

  Ex :

```
carre = lambda x: x * x
print(carre(5))  # affiche 25
```

- map() :

  Applique une fonction a chaque element d'un iterateur.

Ex :

```
nombres = [1, 2, 3, 4]
carres = list(map(lambda x: x**2, nombres))
```

- filter() :

Filtre un iterateur selon une condition.

Ex :

```
pairs = list(filter(lambda x: x % 2 == 0, nombres))
```

- reduce() (depuis functools) :

Applique cumulativement une fonction aux elements d'un iterateur.

Ex :

```
from functools import reduce
produit = reduce(lambda x, y: x * y, nombres)
```

7. Les dictionnaires

---

- Creation d'un dictionnaire :

Stocke des paires cle/valeur.

Ex :

```
mon_dict = {"nom": "Alice", "age": 30, "ville": "Paris"}
```

- Acces et modification :

Ex :

```
print(mon_dict["nom"])  # affiche Alice
mon_dict["age"] = 31
```

- Methodes utiles :
  - keys() pour obtenir les cles.
  - values() pour obtenir les valeurs.
  - items() pour obtenir les paires cle/valeur.

  Ex :

```
for cle, valeur in mon_dict.items():
    print(cle, ":", valeur)
```