Session 2 : Programmation Fonctionnelle et Gestion des Erreurs

1. Programmation Fonctionnelle en Python

La programmation fonctionnelle est un paradigme qui repose sur l'utilisation de fonctions comme entités de première classe. Python propose plusieurs outils pour la programmation fonctionnelle, tels que les fonctions lambda, `map()`, `filter()` et `reduce()`.

1.1 Fonctions Lambda

Les fonctions lambda permettent de créer des fonctions anonymes en une seule ligne.

```
# Fonction lambda pour calculer le carré d'un nombre
carre = lambda x: x ** 2
print(carre(5))  # 25
```

1.2 `map()` : Appliquer une Fonction à une Séquence

La fonction `map()` applique une fonction à tous les éléments d'une séquence.

```
nombres = [1, 2, 3, 4, 5]
carres = list(map(lambda x: x ** 2, nombres))
print(carres)  # [1, 4, 9, 16, 25]
```

1.3 `filter()` : Filtrer une Séquence

La fonction `filter()` permet de sélectionner certains éléments d'une séquence selon un critère.

```
# Filtrer les nombres pairs
pairs = list(filter(lambda x: x % 2 == 0, nombres))
print(pairs)  # [2, 4]
```

1.4 `reduce()` : Réduction d'une Séquence

La fonction `reduce()` de `functools` applique une fonction cumulativement à une séquence.

```
from functools import reduce

# Somme de tous les éléments de la liste
```

```
somme = reduce(lambda x, y: x + y, nombres)
print(somme)  # 15
```

2. Gestion des Erreurs et Exceptions en Python

La gestion des erreurs en Python repose sur l'utilisation des blocs `try`, `except`, `finally` et `raise`.

2.1 Gestion des Exceptions

```
try:
    x = int("abc")  # Cela génère une ValueError
except ValueError as e:
    print(f"Erreur : {e}")
```

2.2 `finally` : Exécution de Code Peu Importe l'Exception

```
try:
    fichier = open("exemple.txt", "r")
    contenu = fichier.read()
except FileNotFoundError:
    print("Le fichier n'existe pas.")
finally:
    print("Exécution terminée.")
```

2.3 Lever une Exception avec `raise`

```
def diviser(a, b):
    if b == 0:
        raise ValueError("La division par zéro n'est pas autorisée.")
    return a / b

try:
    print(diviser(10, 0))
except ValueError as e:
    print(f"Erreur : {e}")
```

3. Débogage et Logging

Le module `logging` permet d'enregistrer des messages pour le débogage.

3.1 Utilisation de `logging`

```
```

```
import logging

logging.basicConfig(level=logging.INFO)

logging.info("Ceci est un message d'information.")
logging.warning("Ceci est un avertissement.")
logging.error("Ceci est une erreur.")
```

4. Exercices Pratiques

Exercice 1 : Manipulation de Fonctions Lambda et `map()`

Écrivez un programme qui utilise `map()` pour convertir une liste de températures en Celsius en Fahrenheit.

```
temperatures_celsius = [0, 10, 20, 30, 40]
temperatures_fahrenheit = list(map(lambda c: (c * 9/5) + 32,
temperatures_celsius))
print(temperatures_fahrenheit)
```

Exercice 2 : Filtrage avec `filter()`

Écrivez un programme qui utilise `filter()` pour extraire les mots ayant plus de 5 lettres d'une liste de mots.

```
mots = ["chat", "éléphant", "chien", "hippopotame"]
longs_mots = list(filter(lambda mot: len(mot) > 5, mots))
print(longs_mots)  # ["éléphant", "hippopotame"]
```

Exercice 3 : Gestion des Erreurs

Modifiez la fonction `diviser()` pour capturer l'exception de division par zéro et afficher un message personnalisé.

```
def diviser(a, b):
    try:
        return a / b
    except ZeroDivisionError:
        return "Erreur : division par zéro !"

print(diviser(10, 0))
```

Exercice 4 : Ajout de Logging

Ajoutez des logs à un programme qui ouvre un fichier et lit son contenu.

```
import logging

logging.basicConfig(level=logging.INFO)

def lire_fichier(nom_fichier):
    try:
        with open(nom_fichier, "r") as f:
            return f.read()
    except FileNotFoundError:
        logging.error("Le fichier n'existe pas.")

print(lire_fichier("inexistant.txt"))
```