Programmation fonctionnelle

Exercice 1 : appliquer une fonction à une liste

Écrire une fonction `apply` qui

- en entrée prend une fonction `f(x : float) -> float` et `l` une liste de `float` ;
- renvoie la liste dont les éléments sont les images des éléments de `l` par `f`, dans le même ordre.

Exemple d'utilisation :

```
def f(x):
    return 2 * x + 1


print(apply(f, [1, 2, 3]))  # [3, 5, 7]
```

Exercice 2 : vérification d'une propriété

On considère une fonction `p` qui

- en entrée prend un `int x` ;
- en sortie renvoie `True` ou `False` selon que `x` vérifie ou non une certaine propriété (être pair, être premier, être compris entre 1 et 100, *et cætera).*)

Écrire une fonction `verify` qui

- en entrée prend `p` une fonction comme spécifié plus haut et `l` une liste d'`int` ;
- en sortie renvoie
  - le premier élément `x` de `l` tel que `p(x)` vaut `True` s'il y en a un ;
  - `None` s'il n'y en a pas.

Exemple d'utilisation :

```
def p(x: int) -> bool:
    return x % 10 == 2


print(verify(p, [1, 3, 293, 202, 14]))
```

Exercice 3 : équation réduite de droite

Écrire une fonction `affine_function2` qui

- en entrée prend quatre `float xA, yA, xB, yB` qui sont les coordonnées de deux points A et B dans un repère du plan ;
- en sortie renvoie
  - `None` si `xA` et `xB` sont égaux ;
  - la fonction affine dont la représentation graphique est la droite (AB) sinon.

Exemple d'utilisation :

```
f = affine_function2(0, 2, 1, 5)  # points sur y = 3 * x + 2
print(f(4))  # 14
```

Exercice 4 : composée de deux fonctions

Écrire une fonction `compose` qui

- en entrée prend deux fonctions `f` et `g` qui représentent deux fonctions de `R` dans `R` ;
- en sortie renvoie la fonction qui correspond à l'enchaînement de `g` puis de `f`, c'est à dire à la fonction qui à tout `float x` associe `f(g(x))`.

Exemple d'utilisation :

```
def u(x):
    return x + 1


def v(x):
    return 2 * x


w = compose(u, v)  # w(x) = u(v(x)) = u(2x) = 2x + 1

print(w(4))  # 2*4 + 1 = 9
```

Exercice 5

Écrire une fonction `evaluate_with_delay` qui

- en entrée prend fonction `f(x : float )-> float`, un `int n` et un `float d`;
- en sortie affiche les résultats `f(0)`, `f(1)`, ..., `f(n-1)` en faisant une pause de `d` millisecondes entre chaque affichage.

On pourra utiliser la fonction `sleep` du module `time` de PYTHON : `sleep(t : float)` met le programme en pause pendant `t` secondes.
Exemple d'utilisation :

```
evaluate_with_delay(lambda x: x * 2, 10, 1000)
# affiche 0 2 4 6 8 10 12 14 16 18  seconde par seconde
```

Exercice 6 : mesurer le temps d'exécution d'une fonction

Écrire une fonction `get_execution_time` qui

- en entrée prend une fonction `f(x : int)` et un `int x`;
- en sortie renvoie le temps nécessaire à l'exécution (évaluation) de `f(x)` en millisecondes.

On pourra utiliser la la fonction `perf_counter` du module `time` qui indique combien de millisecondes se sont écoulées depuis le lancement d'un programme PYTHON.

Exemple d'utilisation :

```
duration = get_execution_time(sum_first_int, 10**8)
print(f"Duration : {duration} milliseconds.")
# affiche Duration : 5.8542724999999995 milliseconds.
```