Programmation 2 Spring, 2026

TD 4: fonctions

1. Écrivez une fonction appelée `est_permier()` qui prend un entier positif comme argument et renvoie `True` s'il s'agit d'un nombre premier et `False` sinon.

   Utilisez-la pour imprimer tous les nombres de 1 à 100 qui sont premiers.

2. Écrivez une fonction qui prend un entier n comme argument et renvoie les n premiers nombres premiers.

3. Quel sera le résultat du code suivant ?

```
def Fun1(mylist):
    for i in range(len(mylist)):
        if mylist[i]%2==0:
            mylist[i]/=2
        else:
            mylist[i]*=2

list1 =[21,20,6,7,9,18,100,50,13]
Fun1(list1)
print(list1)
```

4. Écrivez une fonction `f(x)` qui évalue le polynôme $3x^2 - x + 2$.

5. Écrivez une fonction `my_max(x,y)` qui renvoie le maximum de x et y. N'utilisez pas la fonction `max`, mais utilisez plutôt if des deux manières suivantes :

   (a) Utilisez une instruction if et une instruction else.

   (b) Utilisez une instruction if mais pas d'instruction else ni elif.

6. Un *mot de requête* correspond à un modèle donné `s` s'il peut être obtenu en insérant un nombre quelconque de caractères dans `s`.

   L'insertion peut se faire à n'importe quelle position, y compris au début et à la fin du modèle, et l'insertion de zéro caractère est autorisée (ce qui signifie que `s` lui-même est toujours un mot de requête valide).

   Écrivez une fonction `patternMatching(A, s)` qui prend en entrée une liste `A` de requêtes (une liste de chaînes de caractères) et un modèle `s` (une chaîne de caractères), et renvoie une liste de booléens `B` de même longueur que `A`, telle que `B[i]` vaut **True** si et seulement si `A[i]` correspond au modèle `s`.

   Example: Pour `A = ["FooBar","FooBarTest","Football","FrameBuffer","ForceLack"]` et `s = "FB"`, la fonction renvoie `B = [True,True,False,True,False]`.

7. Nous avons une collection de roches, chaque roche a un poids entier positif.

   À chaque tour, nous choisissons les deux pierres les plus lourdes et les cassons l'une contre l'autre. Supposons que les pierres aient des poids `x` et `y`, avec `x <= y`. Le résultat de ce cas est :

Si `x == y`: les deux pierres sont totalement détruites

Si `x != y`: la pierre de poids `x` est totalement détruite et la pierre de poids `y` a un nouveau poids `y-x`.

À la fin, il ne reste plus qu'une pierre.

Écrivez une fonction `lastStoneWeight` qui prend une liste `A` d'entiers, chacun représentant le poids d'une pierre. Il renvoie le poids de la seule roche restante (ou 0 s'il ne reste plus de pierres).

Exemple: Pour `A = [2,7,4,1,8,1]`, la fonction renvoie `1`:

cassons 7 et 8 → `A = [2,4,1,1,1]`

cassons 2 et 4 → `A = [2,1,1,1]`

cassons 1 et 2 → `A = [1,1,1]`

cassons 1 et 1 → `A = [1]`

8. (a) Démontrer qu'à part 2 et 3, tous les nombres premiers sont de la forme $6k \pm 1$ (bien que tous les nombres de la forme $6k \pm 1$ ne soient pas premiers).
   (b) En utilisant cela, nous pouvons améliorer le temps de calcul d'un facteur 3 pour vérifier si un nombre est un nombre premier. Écrivez le `Python` code correspondant.

9. **(Difficile)** La suite 'Look-and-Say' est une suite d'entiers dont les cinq premiers termes sont les suivants :

   (a) 1
   (b) 11
   (c) 21
   (d) 1211
   (e) 111221

   1 se lit comme 'un 1' ou '11'.
   11 se lit comme 'deux 1' ou '21'.
   21 se lit comme 'un 2, puis un 1' ou '1211'.

   Écrivez une fonction appelée `saySequence()` qui prend un entier positif comme argument et renvoie les n premiers éléments de cette séquence sous forme de `strings`.

10. **(Difficile)** Soit $I_1, I_2, \ldots, I_n$ une famille d'intervalles fermés de $\mathbb{R}$ telle que pour tout $i, j \in \{1, \ldots, n\}$, on ait
$$I_i \cap I_j \neq \emptyset.$$
Montrer qu'il existe un point appartenant à tous les intervalles. Donner un algorithme simple pour trouver un tel point.

11. (**TP question.**) On considère une piste circulaire comportant n stations-service disposées tout autour.

    Chaque station i, où $i \in [0, n-1]$, dispose d'une certaine quantité d'essence, notée `gas[i]`.

    Cette quantité peut varier d'une station à l'autre—certaines peuvent avoir très peu, voire pas d'essence du tout, tandis que d'autres peuvent en avoir un large excédent.

    Soit `cost[i]` la quantité d'essence nécessaire pour se rendre de la station i à la station $(i+1) \bmod n$.

La quantité totale d'essence disponible, toutes stations confondues, est exactement suffisante pour effectuer un tour complet de la piste. Autrement dit,

$$\sum_{i=0}^{n-1} \texttt{gas}[i] = \sum_{i=0}^{n-1} \texttt{cost}[i].$$

Vous commencez votre trajet avec le réservoir vide. À chaque fois que vous atteignez une station, vous pouvez récupérer toute l'essence qui s'y trouve et l'ajouter à votre réservoir.

> Votre objectif est de choisir une station de départ telle qu'en suivant la piste dans l'ordre, vous ne tombiez jamais en panne d'essence et puissiez ainsi effectuer un tour complet (c'est-à-dire revenir à votre point de départ).

Montrer qu'il existe toujours au moins une station de départ permettant d'effectuer le circuit complet sans jamais tomber en panne d'essence.