

Exercice 1. Fonctions avec boucles for

1. Remplacer les ? dans la fonction `somme`.
2. Montrer la terminaison de la fonction `somme` en précisant le nombre exact de tours de boucle.
3. Montrer la correction de la fonction `somme`.

```

1 # n >= 0
2 # Renvoie 1 + 2 + ... + n
3 def somme(n):
4     """(n: int) -> int"""
5     s = 0
6     for i in range(?, ?):
7         s = s + i
8     return s

```

```

1 # n >= 0
2 # Renvoie f_n où f_0 = 0, f_1 = 1
3 # et f_{n+2} = f_{n+1} + f_n
4 def fibo2(n):
5     """(n: int) -> int"""
6     L = [0,1]
7     for i in range(?):
8         L.append(L[-1] + L[-2])
9     return L[n]

```

```

1 def x_in_L(L,x):
2     """(L: list[int],x: int) -> bool"""
3     for e in L:
4         if e == x:
5             return True
6     return False

```

```

1 def somme_liste(L):
2     """(L: list[int]) -> int"""
3     s = 0
4     for e in L:
5         s += e
6     return s

```

```

1 # Teste si n >= 2 est premier
2 def est_1er(n):
3     """(n: int) -> bool"""
4     for d in range(?,?):
5         if n % d == 0:
6             return False
7     return True

```

```

1 # n >= 0
2 # Compte les nombres premiers <= n
3 def cpt_1er(n):
4     """(n: int) -> int"""
5     cpt = 0
6     for i in range(?,?):
7         if est_1er(i):
8             cpt += 1
9     return cpt

```

4. Remplacer les ? dans les fonctions ci-dessus.
5. Pour chacune des fonctions, préciser le nombre exact ou le nombre maximal de tours de boucle.
6. Pour chacune des fonctions, donner un invariant de boucle. Ici, on ne demande pas de faire la preuve complète de la correction, on demande seulement de donner l'invariant, d'admettre que c'est bien un invariant (sans faire la récurrence) puis d'en déduire la correction de la fonction.

Exercice 2. Fonctions avec boucles while

```
1 # Renvoie le plus petit k tq L[k] == x
2 # ou None si k n'existe pas.
3 def ind(L,x):
4     """(L: list[int],x: int) -> int"""
5     k = 0
6     while k < len(L) and L[k] != x:
7         k = k + 1
8     if k == len(L):
9         return None
10    else:
11        return k
```

```
1 def somme_liste2(L):
2     """(L: list[int]) -> int"""
3     s = 0
4     i = 0
5     while i < ?:
6         s = s + L[i]
7         i = i+1
8     return s
```

1. Montrer la terminaison de la fonction `ind`.
2. Montrer la correction de la fonction `ind`.
3. Remplacer le ? dans la fonction `somme_liste2`.
4. Trouver un variant de boucle pour la fonction `somme_liste2`. Ici, on ne demande pas de faire la preuve de la terminaison, on demande seulement le variant.
5. Donner un invariant de boucle pour la fonction `somme_liste2`. Ici, on ne demande pas de faire la preuve de la correction, on demande seulement l'invariant.

Exercice 3. Fonctions récursives

Le but de cet exercice est de calculer le nombre de chiffres qui composent un entier positif ou nul. Par exemple, l'entier 147 contient 3 chiffres, les entiers 1 et 2 contiennent 1 chiffre, et par convention l'entier 0 contient 0 chiffre. Afin de compter le nombre de chiffres dans un entier n , on peut effectuer la division entière de n par 10 jusqu'à obtenir 0 et compter le nombre de divisions effectuées.

1. Écrire une fonction récursive qui prend en argument un entier n et renvoie le nombre de chiffres de n à l'aide de la méthode décrite ci-dessus.
2. Montrer la terminaison et la correction de votre fonction.

Exercice 4. Miroir d'une liste

Dans cet exercice, on étudie la fonction `miroir` qui inverse l'ordre des éléments d'une liste :

```
1 def swap(L, i, j):
2     """
3     L: list[int]; i,j: int
4     Return: NoneType
5     """
6     tmp = L[i]
7     L[i] = L[j]
8     L[j] = tmp
```

```
1 def miroir(L):
2     """(L:list[int]) -> NoneType"""
3     for i in range(?, ?):
4         swap(L, i, len(L)-1-i)
```

1. Remplacer les ? pour que la fonction `miroir` fasse un minimum de tours de boucle.
2. Quel est le nombre de tours de boucle dans la fonction `miroir`? En déduire que la fonction `miroir` termine.
3. Montrer la correction de la fonction `miroir`.

Exercice 5. Tri par sélection

Dans cet exercice, on étudie le tri par sélection pour trier une liste d'entiers. Notez que l'algorithme utilisé modifie directement la liste donnée en argument au lieu de créer une nouvelle liste (un tel algorithme est appelé un "tri en place").

```
1 def mini(L, i0):
2     """
3     L: list[int]; i0: int
4     Return: int
5     """
6     m = i0
7     i = i0 + 1
8     while i < len(L):
9         if L[m] > L[i]:
10            m = i
11            i = i+1
12    return m

1 def swap(L, i, j):
2     """
3     L: list[int]; i,j: int
4     Return: NoneType
5     """
6     tmp = L[i]
7     L[i] = L[j]
8     L[j] = tmp

1 def tri(L):
2     """tri(L: list[int]) -> NoneType"""
3     for i0 in range(len(L)):
4         m = mini(L, i0)
5         swap(L, m, i0)
```

1. Soit L une variable globale qui vaut $[5,6,7,0,2]$. Expliquer pourquoi `print(tri(L))` ne permet pas de tester la fonction. Comment tester la fonction ?
2. Que fait la fonction `mini` ? Prouver sa terminaison et sa correction.
3. Prouver la terminaison et la correction de la fonction `tri`.

Exercices à faire pour la séance du 12/03/2024

Faites ces exercices sur feuille et apportez les lors de la séance de TP du 12/03/2024. Je ramasserai certaines copies au hasard.

Exercice 6. Moyenne des éléments d'une liste d'entiers

1. À l'aide d'une boucle `for`, écrire une fonction qui calcule la moyenne des éléments de la liste d'entiers donnée en argument. On pourra supposer que la liste est non vide.
2. Montrer la terminaison de votre fonction.
3. Montrer la correction de votre fonction.

Exercice 7. Calcul de la factorielle

1. Remplacer le ? dans la fonction `fact`.
2. Montrer la terminaison de la fonction `fact`.
3. Montrer la correction de la fonction `fact`.

```
1 # n >= 0
2 def fact(n):
3     """(n: int) -> int"""
4     i = 2; res = 1
5     while i <= ?:
6         res = res * i
7         i = i+1
8     return res
```

Exercices à rendre au plus tard le 10/03/2024 à 20h

Envoyez moi cet exercice en utilisant l'adresse mail habituelle.

Exercice 8. Tri de crêpes (DS de l'année 2021-2022)

Un forain présent lors de la foire de Nancy a préparé une pile de crêpes et veut les trier en fonction de leurs diamètres (la plus grande en dessous et la plus petite au dessus). Par exemple en partant de la figure 1 ou de la figure 2, il souhaite obtenir la figure 3 :

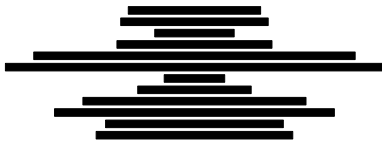


FIGURE 1



FIGURE 2

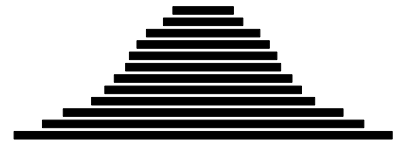


FIGURE 3

Soit $n \in \mathbb{N}$ le nombre de crêpes. On indice les crêpes de 0 à $n - 1$, celle d'indice 0 étant celle se trouvant en dessous. Pour effectuer son tri, le forain dispose de deux opérations :

- ★ Opération 1. À partir d'un indice $i \in \llbracket 0, n - 1 \rrbracket$, il peut trouver l'indice $j \geq i$ correspondant à une crêpe de diamètre maximal. Avec l'exemple de la figure 1 :

i	0	1	2	3	4	5	6	7	8	9	10	11
j	6	6	6	6	6	6	6	7	8	10	10	11

Pour cela, il suffit de repérer la crêpe qui dépasse par rapport aux autres.

- ★ Opération 2. Étant donné un indice $i \in \llbracket 0, n - 1 \rrbracket$, le forain peut placer sa spatule sous la crêpe d'indice i et retourner la pile se trouvant au dessus de la spatule. Par exemple avec $i = 4$ et la pile de la figure 1, on obtient la figure 2.

En Python, une pile de crêpes sera représentée par la liste des diamètres en commençant par la crêpe d'indice 0. Ainsi, la figure 1 correspond à :

[5.2, 4.7, 7.4, 5.9, 3, 1.6, 10, 8.5, 4.1, 2.1, 3.9, 3.5]

1. Dans cette question, la liste « L: list[float] » représente la pile de crêpes.
 - (a) Écrire une fonction « maxi(L: list[float], i: int) -> int » correspondant à l'opération 1.
 - (b) Écrire une fonction « spatule(L: list[float], i: int) -> NoneType » correspondant à l'opération 2. La fonction doit modifier L directement, sans créer de liste intermédiaire et ne doit rien renvoyer. Pour tester cette fonction on écrira :


```

                    L = [...]
                    spatule(L, ...)
                    print(L)
                    
```
2.
 - (a) Trouver une procédure pour trier la pile de crêpes en utilisant uniquement les opérations 1 et 2. On attend une explication en français, pas un programme Python.
 - (b) Donner en le démontrant un majorant sur le nombre d'utilisations des opérations 1 et 2 dans la procédure de la question 2a.
3.
 - (a) Écrire une fonction « tri_crepes(L: list[float]) -> NoneType » correspondant à la procédure décrite dans la question 2a. Cette fonction doit modifier L et ne rien renvoyer.
 - (b) Quelle est la complexité de la fonction tri_crepes ?
4. Écrire une fonction « tri_crepes_bis(L: list[float]) -> list[float] » qui effectue le tri crêpes. La différence avec la fonction de la question 3a est qu'elle doit renvoyer une nouvelle liste et ne pas modifier L.

Question facultative. Le forain n'est pas complètement satisfait de la fonction tri_crepes car il pense avoir utilisé l'opération 2 plus de fois que nécessaire. Pour toute liste de crêpes P , on note $op_2(P)$ le nombre minimal d'utilisations de l'opération 2 pour trier P . Pour tout entier n , on note M_n le maximum des $op_2(P)$ lorsque P parcourt toutes les piles de crêpes de taille n . Par exemple :

n	0	1	2	3	4	5	6	7	8	9	10
M_n	0	0	1	3	4	5	7	8	9	10	11

Ainsi, n'importe quelle pile composée de 6 crêpes peut être triée en utilisant au plus 7 fois l'opération 2. De plus, il existe une pile composée de 6 crêpes qui ne peut pas être triée en utilisant moins de 7 fois l'opération 2.

5. Écrire une fonction `nb_max_op2` qui prend en entrée $n \in \mathbb{N}$ et renvoie M_n .