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`.

```python
# n >= 0
# Renvoie 1 + 2 + ... + n
def somme(n):
    """(n: int) -> int"""
    s = 0
    for i in range(?, ?):
        s = s + i
    return s
```

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

```python
def x_in_L(L,x):
    """(L: list[int],x: int) -> bool"""
    for e in L:
        if e == x:
            return True
    return False
```

```python
def somme_liste(L):
    """(L: list[int]) -> int"""
    s = 0
    for e in L:
        s += e
    return s
```

```python
# Teste si n >= 2 est premier
def est_1er(n):
    """(n: int) -> bool"""
    for d in range(?,?):
        if n % d == 0:
            return False
    return True
```

```python
# n >= 0
# Compte les nombres premiers <= n
def cpt_1er(n):
    """(n: int) -> int"""
    cpt = 0
    for i in range(?,?):
        if est_1er(i):
            cpt += 1
    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`

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

```
def somme_liste2(L):
    """(L: list[int]) -> int"""
    s = 0
    i = 0
    while i < ?:
        s = s + L[i]
        i = i+1
    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 :

```
def swap(L, i, j):
    """
    L: list[int]; i,j: int
    Return: NoneType
    """
    tmp = L[i]
    L[i] = L[j]
    L[j] = tmp
```

```
def miroir(L):
    """(L:list[int]) -> NoneType"""
    for i in range(?, ?):
        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").

```
def mini(L, i0):
    """
    L: list[int]; i0: int
    Return: int
    """
    m = i0
    i = i0 + 1
    while i < len(L):
        if L[m] > L[i]:
            m = i
        i = i+1
    return m
```

```
def swap(L, i, j):
    """
    L: list[int]; i,j: int
    Return: NoneType
    """
    tmp = L[i]
    L[i] = L[j]
    L[j] = tmp
def tri(L):
    """tri(L: list[int]) -> NoneType"""
    for i0 in range(len(L)):
        m = mini(L, i0)
        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 22/04/2025

Faites ces exercices sur feuille et apportez les lors de la séance de TD du 22/04/2025. 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`.

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