TD9 : Prog.

Les fonctions suivantes (sans précondition) sont codées dans l'intention de réaliser la spécification ci-dessus.

2. Pour chaque fonction :

- indiquer si son code est conforme aux bonnes pratiques de l'UE
- en réfléchissant au code ou en déroulant à la main l'exécution sur les données de test, indiquer si le code est correct (= la fonction renvoie un résultat conforme à sa spécification pour tout paramètre satisfaisant sa précondition)
- si le code n'est pas correct : donner un ou plusieurs tests qui mettent en évidence le problème, ainsi qu'un ou plusieurs tests qui **ne mettent pas** en évidence le problème et peuvent laisser penser que la fonction est correctement codée.

```
def foo1(liste:list[int]) -> bool:
    for elem in liste:
      if elem >= 0:
         return True
      else:
         return False

def foo2(liste:list[int]) -> bool:
    for elem in liste:
      if elem >= 0:
         res = True
      else:
         res = False
    return res
```

```
def foo3(liste:list[int]) -> bool:
    i = 0
    while i < len(liste):
       if liste[i] >= 0:
          return True
       i = i + 1
    return False

def foo4(liste:list[int]) -> bool:
    i = 0
    while liste[i] < 0 and i < len(liste):
       i = i + 1
    return False
```

Vous devez répondre à la question suivante en réfléchissant, sans le support d'une exécution Python.

On souhaite écrire une fonction qui renvoie `True` ssi la liste d'entiers passée en paramètre contient au moins un élément positif ou nul.

2a. return liste[0] >= 0

pas correct.

$$$ foo1([42, -5])

True

$$$ foo1([-1, 2024])

True

Faux

b. for elem in liste: res = elem >= 0

pas correct.

$$$ foo2([-27, 30])

True

$$$ foo2([30, -27])

True.

$$$ foo2([1, -12, 30])

True

$$$ foo2([-2, -4, -6])

False

Faux

---

c. Semble conforme ici si return True.

correct

$$$ foo2([-27, 30])

True

$$$ foo2([30, -27])

True.

$$$ foo2([1, -12, 30])

True

$$$ foo2([-2, -4, -6])

False

d. semble conforme
pas correct

$$$ foo4([-2,-3,-7])
False
$$$ foo4([4,7,2])
True
$$$ foo4([4,-7,2])
True

FAUX

```
def foo5(x : str, u : str) -> list[str]:
   t = []
   for i in x:
      if i != u:
         t.append(i)
   return t
```

```
def foo5(chaine : str, charistr) -> list[str]:
    Res = []
    for caractere in chaine:
        if caractere != char:
            Res.append(caractere)
    Return Res
```

```
def foo6(a:list[int]) -> bool:
    x = 0
    while x < len(a) and a[x] != -1:
        x = x + 1
    return x < len(a)
```

```
def foo6(liste : list[int]) -> bool:
    i = 0
    while i < len(liste) and liste[i] != -1:
        i += 1
    Return i < len(liste).
```

4. Écrire un prédicat `est_premier_naif` qui prend en paramètre un entier `p` positif et renvoie `True` ssi `p` est premier.

```
def est_premier_naif(p:int) -> bool:
    i=2
    premier=True.                    ∀x∈P
    while i*i<n and premier:
        if p%i==0:
            premier=False
    return premier
```

5. Écrire un prédicat `au_moins_un_premier` qui prend en paramètre un intervalle et renvoie `True` ssi l'intervalle contient au moins un nombre premier. Par exemple l'intervalle, `range(12)` contient 2, qui est premier, alors qu'aucun des trois éléments de l'intervalle `range(8, 11)` (qui sont 8, 9 et 10) n'est premier.

```
def au_moins_un_premier(l:Range )-> bool
    i=0
    while i<len(l) and not est_premier_naif(l[i]):
        i+=1.
    return i!=len(l) # si l ne contient pas de premiers on a i==len(l).
```

6. Écrire une fonction `genere_binaire` qui prend en paramètre un entier `n` positif et génère aléatoirement une suite de `'0'` et de `'1'` qui contient au moins `n` occurrences de `'0'` et `n` occurrences de `'1'`.