Spécialité NSI Terminale
Correction du contrôle n° 2
Lundi 10 octobre 2022

Exercice 1

1. La première valeur prise par la variable `i` dans la boucle `for` est 0. L'instruction

```
total = total + 1/i
```

génère donc alors une division par 0.
On peut corriger le problème *et obtenir le résultat attendu* en écrivant le code suivant :

```
def somme(n) :
    total = 0
        for i in range(1, n+1) :
            total = total + 1/i
    return total
```

2. a. Lorsque la variable `indice` est égale à la longueur de la liste `L`, l'expression `L[indice]` génère le message d'erreur "`IndexError : list index out of range`". On peut corriger ce problème en écrivant la boucle de la façon suivante :

```
    while indice < len(L) :
        ...
```

b. La fonction renvoie `0` car ici tous les éléments de la liste sont négatifs donc inférieurs à `0`. Pour corriger le problème il faut changer l'initialisation de la variable `maximum` en lui donnant la première valeur de la liste de la façon suivante :

```
maximum = L[0]
```

3. La variable `i` est de type `int` alors que `'Joueur '` est un objet de type `str`. L'expression `'Joueur '+i` ne peut pas donc pas être exécutée comme une concaténation de chaînes de caractères. Pour résoudre ce problème il faut effectuer une conversion de type sur la variable `i` en modifiant l'instruction de la façon suivante :

```
        L.append('Joueur ' + str(i))
```

4. a. Il s'agit d'une fonction récursive. **L'appel `suite(6)` renvoie `21`**. En effet,
* l'appel `suite(0)` renvoie `0` ;
* l'appel `suite(2)` renvoie `3+2*0` donc `3` ;
* l'appel `suite(4)` renvoie `3+2*3` donc `9` ;
* l'appel `suite(2)` renvoie `3+2*9` donc `21`.

b. SI on exécute l'appel `suite(7)` on obtient en théorie une succession infinie d'appels récursifs et en pratique un message d'erreur après 1000 appels récursifs. En effet, puisque l'appel récursif est `suite(n-2)`, tous les appels se feront avec des arguments impairs or le code de la fonction ne comporte pas de cas de base avec un nombre impair.

5. a. Le premier `print` affiche le couple renvoyé par la fonction `modif` donc

```
(5, [10])
```

b. Le deuxième `print` affiche le couple formé par les valeurs finales *des variables globales* `x` et `L`. Or la variable `x` (de type `int` non mutable) n'a pas été modifiée par l'appel de fonction `modif(x, L))` alors que la variable `L` (de type `list` mutable) a été modifiée par cet appel. On obtient donc l'affichage suivant :

```
(4, [10])
```

Exercice 2

1. (a) Il s'agit de la **proposition 3**. En effet, si la longueur du texte est de 0 ou 1 alors ce texte est un palindrome

(b) On a les valeurs suivantes :

* `txt[0] : "b"`
* `txt[taille - 1] : "r"`
* `interieur : "onjou"`

2. On doit tester un cas où la fonction doit renvoyer `True` et un autre cas où elle doit renvoyer `False`. Par exemple

* `print(palindrome("laval"))` qui doit afficher `True` ;
* `print(palindrome("bonjour"))` qui doit afficher `False`.

3. On peut écrire la version non récursive suivante :

```
def palindrome(txt) :
    taille = len(txt)
    for i in range(taille//2) :
        if txt[i] != txt[taille-1-i] :
            return False
    return True
```

4. a. On peut écrire le code suivant :

```
def complementaire(txt) :
    res = ""
    for car in txt :
        if car == "A" :
            res = res + "T"
        elif car == "T" :
            res = res + "A"
        elif car == "G" :
            res = res + "C"
        elif car == "C" : # ou 'else :'
            res = res + "G"
    return res
```

b. Le complémentaire de `"GATCGT"` est `"CTAGCA"`. Or la chaîne de caractère `"GATCGTCTAGCA"` (concaténation de ces deux chaînes de caractères) n'est pas un palindrome donc `"GATCGT"` n'est pas palindromique.

c. On peut écrire le code suivant :

```
def est_palindromique(txt) :
    nouv_txt = txt + complementaire(txt)
    return palindrome(nouv_txt)
```