Récursivité

Exercices corrigés

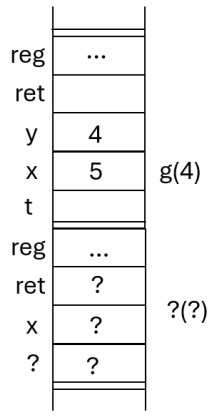
Exercice 1

Étant données les fonctions Python suivantes.

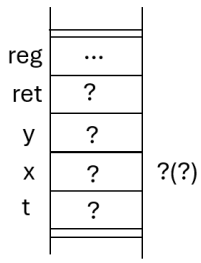
```
def f(x) :
    z = x * x
    return z – x

def g(y) :
    x = y +1
    t = f(x)
    return x + y + t
```

Décrire la configuration de la pile pour l'appel à g(4), au moment (A) ou l'appel interne f(x) s'apprête à renvoyer sa valeur et (B) où la fonction g s'apprête à renvoyer son résultat.

(A)

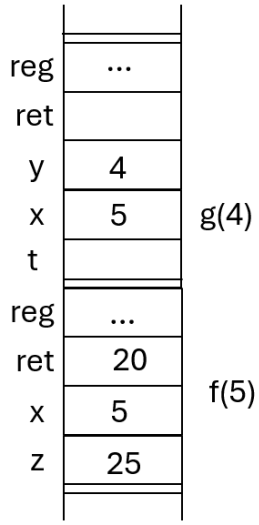


(B)

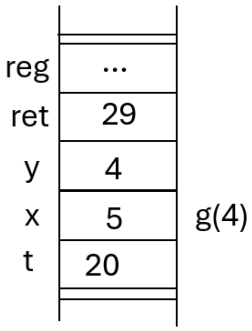


Correction

(A)



(B)



Exercice 2

Implémenter ces fonctions de manière récursive.

```
def compte_a_rebours(n: int) -> None:
    """Affiche n, n - 1, n - 2, ..., 2, 1, Partez !"""

def n_entiers(n: int) -> None:
    """Affiche les n premiers entiers naturels de 0 à n - 1"""
```

```
def compte_a_rebours(n):
    if n == 0:
        print("Partez !")
    else:
        print(n)
        compte_a_rebours(n - 1)


def n_entiers(n):
    if n == 1:
        print(n - 1)
    else:
        n_entiers(n - 1)
        print(n - 1)
```

Exercice 3

Implémenter ces fonctions de manière récursive.

a)

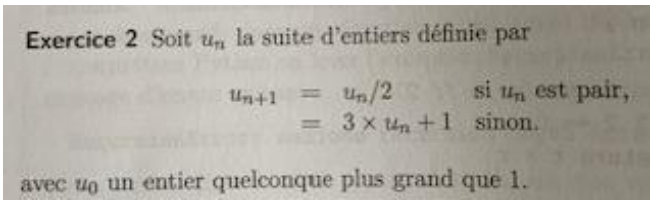
```
# Triangle croissant
def triangle_croissant(taille):
    """
    triangle_croissant(4) renvoie :
    #
    ##
    ###
    ####
    """
    pass
```

b)

```
# Triangle décroissant
def triangle_decroissant(taille):
    """
    triangle_decroissant(4) renvoie :
    ####
    ###
    ##
    #
    """
    pass
```

```python
# Triangle croissant
def triangle_croissant(taille):
    """
    triangle_croissant(4) renvoie :
    #
    ##
    ###
    ####
    """
    if taille == 1:
        return "#"
    else:
        return f"{triangle_croissant(taille - 1)}\n{'#' * taille}"

# Triangle décroissant
def triangle_decroissant(taille):
    """
    triangle_decroissant(4) renvoie :
    ####
    ###
    ##
    #
    """
    if taille == 1:
        return "#"
    else:
        return f"{'#' * taille}\n{triangle_decroissant(taille - 1)}"
```

Exercice 4

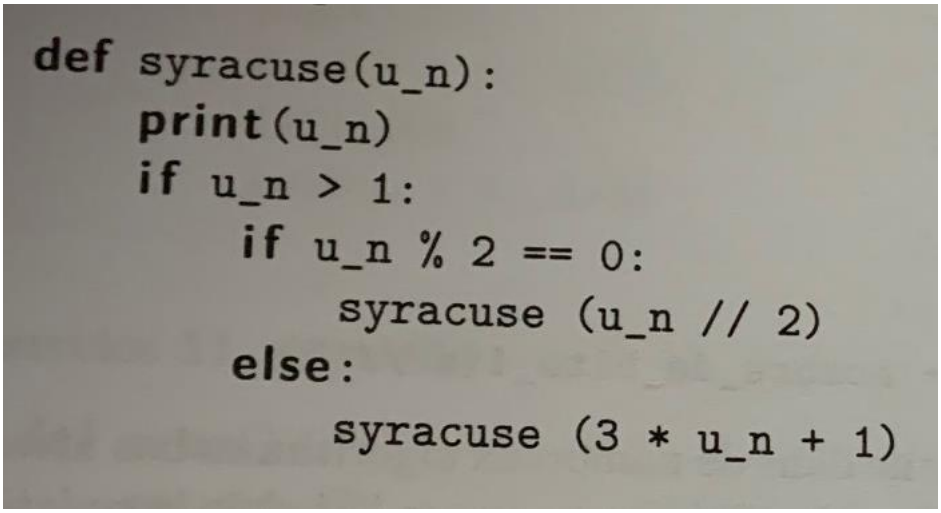


Exercice 2 Soit u_n la suite d'entiers définie par

$$\begin{aligned} u_{n+1} &= u_n/2 && \text{si } u_n \text{ est pair,} \\ &= 3 \times u_n + 1 && \text{sinon.} \end{aligned}$$

avec u_0 un entier quelconque plus grand que 1.

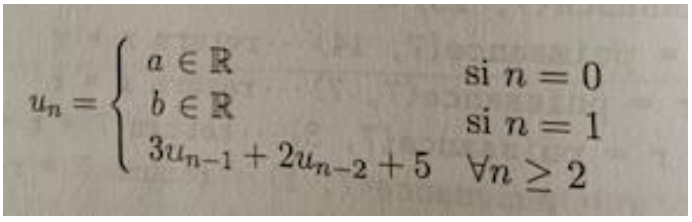
Écrire une fonction récursive Syracuse(u_n) qui affiche les valeurs successives de la suite Un tant que Un est plus grand que 1.



```
def syracuse(u_n):
    print(u_n)
    if u_n > 1:
        if u_n % 2 == 0:
            syracuse (u_n // 2)
        else:
            syracuse (3 * u_n + 1)
```

Exercice 5

On considère la suite suivante définie par la relation de récurrence suivante, où a et b sont des réels quelconques :



$$u_n = \begin{cases} a \in \mathbb{R} & \text{si } n = 0 \\ b \in \mathbb{R} & \text{si } n = 1 \\ 3u_{n-1} + 2u_{n-2} + 5 & \forall n \geq 2 \end{cases}$$

Écrire une fonction récursive serie(n, a, b) qui renvoie le n-ème terme de cette suite par des valeurs a et b données en paramètres.

```python
def serie(n, a, b):
    if n == 0:
        return a
    elif n == 1:
        return b
    else:
        r = 3 * serie(n - 1, a, b) + 2 * serie(n - 2, a, b) + 5
        return r
```

Exercice 6

Écrire une fonction récursive pgcd(a, b) qui renvoie le PGCD de deux entiers a et b.

En arithmétique élémentaire, le **plus grand commun diviseur** ou **PGCD** de deux nombres entiers non nuls est le plus grand entier qui les divise tous les deux. Par exemple, le PGCD de 20 et de 30 est 10, puisque leurs diviseurs communs sont 1, 2, 5 et 10.

```python
def pgcd(a, b):
    if a == 0:
        return b
    else:
        return pgcd(b % a, a)
```

Exercice 7

```python
def somme(tableau: list) -> int:
    """Calculer la somme des valeurs d'un tableau"""
```

```python
# Somme d'une liste
def somme(tableau):
    "Calculer la somme des valeurs d'un tableau"
    if len(tableau) == 1:
        return tableau[0]
    else:
        return tableau[0] + somme(tableau[1:])
```

Exercice 8

```python
# Maximum d'une liste
def maximum(tableau):
    "Déterminer la valeur maximale dans un tableau"
    pass
```

```
# Maximum d'une liste
def maximum(tableau):
  "Déterminer la valeur maximale dans un tableau"
  if len(tableau) == ... :
    return tableau[0]
  else:
    premier = ...
    maxi_fin = ...
    if premier > ...:
      return premier
    else:
      ...
```

```python
# Maximum d'une liste
def maximum(tableau):
    "Déterminer la valeur maximale dans un tableau"
    if len(tableau) == 1:
        return tableau[0]
    else:
        premier = tableau[0]
        maxi_fin = maximum(tableau[1:])
        if premier > maxi_fin:
            return premier
        else:
            return maxi_fin
```

Exercice 9

Écrire une fonction récursive appartient(v, t, i) prenant en paramètre une valeur v, un tableau t et un entier i et renvoyant True si v apparaît dans t entre l'indice i (inclus) et len(t) (exclu), et False sinon. On supposera que i est toujours compris entre 0 et len(t).

```
def appartient(v, t, i):
    if i == len(t):
        return False
    else:
        return t[i] == v or appartient(v, t, i + 1)
```

Exercice 10

```
def nombre_chiffres(n: int) -> int:
    """
    Renvoie le nombre de chiffres nécessaires à l'écriture de n en base 10
    n est un entier positif
    On rappelle que 536 // 10 = 53
    nombres_chiffres(509) -> 3
    """

def somme_chiffres(n: int) -> int:
    """
    Renvoie la somme des chiffres composant n
    somme_chiffres(509) -> 14
    """


def listes_imbriquees(n: int) -> list:
    """
    Renvoie n listes imbriquées
    listes_imbriquées(1) -> []
    listes_imbriquées(2) -> [[]]
    listes_imbriquées(3) -> [[[]]]
    """

def profondeur(liste: list) -> int:
    """
    liste est une liste telle que renvoyée par la fonction listes_imbriquées
    La fonction profondeur renvoie le nombre de listes utilisées
    profondeur([]) -> 1
    profondeur([[]]) -> 2
    profondeur([[[]]]) -> 3
    """
```

```python
def nombre_chiffres(n):
    # Si n est compris entre 0 et 9, alors il ne faut qu'un chiffre pour l'écrire
    if -9 <= n <= 9:
        return 1
    # Sinon, on divise n par 10 (ce qui retire un chiffre) et on appelle récursivement la fonction
    return 1 + nombre_chiffres(n // 10)

# Exemple d'utilisation
print(nombre_chiffres(12345))  # Affiche 5
print(nombre_chiffres(-987))   # Affiche 3
```

```python
def somme_chiffres(n):
    if n < 10:
        return n
    else:
        return n % 10 + somme_chiffres(n // 10)


def listes_imbriquees(n):
    return [] if n == 1 else [listes_imbriquees(n - 1)]


def profondeur(liste):
    if liste == []:
        return 1
    else:
        return 1 + profondeur(liste[0])
```

Exercice 11

Les palindromes

On appelle palindrome un mot qui se lit dans les deux sens comme « été » ou « radar ».
Écrire une fonction récursive palindrome qui teste si un mot est un palindrome.
• Entrée : Un mot (type str).
• Sortie : Un booléen égal à True si le mot est un palindrome, False sinon.

Principe
Cas de base :
• si le mot est la chaîne vide, c'est un palindrome ;
• si le mot ne contient qu'une seule lettre, c'est un palindrome.

Dans les autres cas :
• le mot est un palindrome si et seulement si la première et la dernière lettre sont égales et le mot tronqué de ses première et dernière lettres est un palindrome.

```
def est_palindrome(mot):
    # Si le mot a une longueur de 0 ou 1, il est automatiquement un palindrome
    if len(mot) <= 1:
        return True
    # Vérifier si le premier et le dernier caractère sont identiques
    elif mot[0] != mot[-1]:
        return False
    # Appeler récursivement la fonction en supprimant le premier et le dernier caractère
    else:
        return est_palindrome(mot[1:-1])
```