+1/1/60+

Programmation impérative en Python — SPUF21

Année 2021-2022 — Partiel de mi-semestre

Nom : .....

Prénom : .....

Numéro d'étudiant : .....

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9

Durée : 2 heures.

Aucun document n'est autorisé. L'usage de la calculatrice ou de tout autre appareil électronique est interdit.

Les exercices sont indépendants. Au sein d'un même exercice, vous pouvez utiliser les variables et fonctions des questions précédentes, même si vous n'avez pas su les faire ; chaque question est donc indépendante.

À part les méthodes et fonctions de base, vous n'avez pas le droit d'utiliser les fonctions et les méthodes « avancées », sauf si l'énoncé vous conseille l'utilisation de certaines d'entre elles.

```python
# Fonctions autorisées
len(…)
range(…)
print(…)

# Méthode autorisée
L.append(x)
```

```python
# Par exemple les méthodes et fonctions suivantes sont entre autres interdites
max(…) min(…) sum(…)
s.split(…) s.index(…) L.extend(…)

# Vous n'avez pas le droit d'utiliser des compréhensions ou des slices
# sur les chaînes.  À la place vous devez utiliser des boucles.
[ x for x in range(L) ]
chaine[début:fin:pas]
```

Exercice 1 Questions rapides . . . . . 3 points

☐0 ☐1 ☐2 ☐3

1. Constuire avec une boucle `for` une liste `L` de la forme `[100, 99, 98, …, 3, 2, 1]`.

```
L=[]
for i in range(100,0,-1):
    L.append(i)
```

2. En utilisant une boucle `for`, définir une fonction `énumérer(chaîne)` qui compte le nombre de caractères en affichant la chaîne à chaque ligne. Par exemple on aura :

```
>>> énumérer("covid") # 5 lettres
1 covid
2 covid
3 covid
4 covid
5 covid
```

```
>>> énumérer("SARS") # 4 lettres
1 SARS
2 SARS
3 SARS
4 SARS
```

```
def énumérer(chaîne):
    for i in range(len(chaîne)):
        print(i+1,chaîne)
```

3. On cherche à écrire une fonction `nombre(chaîne)` qui renvoie `True` si la chaîne représente un entier, c'est-à-dire si elle ne contient que des chiffres. Sinon elle doit renvoyer `False`. Parmi les trois tests avec `assert`, lesquels, éventuellement, renvoient des erreurs ?

```
def nombre(chaîne):
    for caractère in chaîne:
        if ord("0") <= ord(caractère) and ord(caractère) <= ord("9"):
            return True
    return False

assert nombre("3") == True
assert nombre("2019") == True
assert nombre("Pangolin") == False
```

```
Aucun !
```

4. La fonction `nombre` est fausse. Que fait-elle vraiment ? Écrire un test avec `assert` qui échoue mais qui

aurait dû fonctionner si la fonction respectait les consignes de l'énoncé.

```
Elle renvoie True si chaîne contient au moins un chiffre.
assert nombre("ABC1ABC") == False # mais ici nombre renvoie True
```

Exercice 2 Tapis au temps du COVID . . . . . 2 points

☐0 ☐0,5 ☐1 ☐1,5 ☐2

```
def confinement():
    print("C",end="")

def déconfinement():
    print("D",end="")
```

```
def vaccin():
    print("V",end="")

def nouvelle_ligne():
    print()
```

Écrire la fonction `tapis(n)` qui reproduit le motif ci-dessous de largeur et hauteur `n` sans utiliser la fonction `print`, mais uniquement les quatre fonctions définies au-dessus.

```
>>> tapis(5)
CDCDC
VVVVV
CDCDC
VVVVV
CDCDC

```

```
>>> tapis(6)
CDCDCD
VVVVVV
CDCDCD
VVVVVV
CDCDCD
VVVVVV
```

```
def tapis(n):
    for j in range(n):
        for i in range(n):
            if j%2==0:
                if i%2==0:
                    confinement()
                else:
                    déconfinement()
            else:
                vaccin()
        nouvelle_ligne()
```

Exercice 3 Évolution de la pandémie du COVID . . . . . 5 points

☐0 ☐0,5 ☐1 ☐1,5 ☐2 ☐2,5 ☐3 ☐3,5 ☐4 ☐4,5 ☐5

On modélise l'évolution de la maladie du COVID par une formule de récurrence où n représente le nombre de jours depuis le début de la pandémie et u_n le nombre de contaminés au jour n.

$$\begin{cases} u_0 &= 1 \\ u_{n+1} &= 1{,}5 \times u_n \end{cases}$$

1. Écrire une fonction `contaminés(n)` qui calcule **par récursion** le nombre u_n de contaminés au jour n de la pandémie.

```
def contaminés(n):
    if n==0:
        return 1
    else:
        return 1.5 * contaminés(n-1)
```

2. Pour l'instant le nombre de contaminés est multiplié par 1,5 à chaque journée. On veut modéliser un comportement plus complexe. Écrire la fonction `jour_suivant(nb)` qui, à partir d'une valeur `nb` positive (correspondant au nombre de contaminés), renverra le nombre de contaminés du jour d'après donné par les formules suivantes : • $2\texttt{nb}$ si $\texttt{nb} \in]100, 1000]$ • $1,5\texttt{nb}$ si $\texttt{nb} \in]1000, +\infty[$ • $3\texttt{nb}$ si $\texttt{nb} \in [0, 100]$

```
>>> jour_suivant(50)  # renvoie 3×50
150
>>> jour_suivant(150) # renvoie 2×150
300
```

```
def jour_suivant(nb):
    if nb>1000:
        return nb*1.5
    elif nb>100:
        return nb*2
    else:
        return nb*3
```

3. Écrire une fonction `premier_mois(c0)` qui renvoie la liste du nombre de contaminés pour le premier mois (30 jours). L'entier `c0` correspond au nombre initial de contaminés. Par exemple :

```
>>> L = premier_mois(1)
>>> L[0:10] # les dix premières valeurs sur 30
[1, 3, 9, 27, 81, 243, 486, 972, 1944, 2916.0]
```

```
def premier_mois(c0):
    L = []
    for i in range(30):
        L.append(c0)
        c0 = jour_suivant(c0)
    return L
```

4. On souhaite suivre l'évolution de la maladie pour lancer un confinement lorsque le nombre de contaminés dépassera le million. Écrire une fonction `affiche_début_confinement(c0)`, qui calcule l'évolution de la maladie et s'arrête lorsque le nombre de contaminés dépasse strictement le million. Elle affichera alors le numéro du jour ainsi que le nombre de contaminés comme on peut le voir sur l'exemple.

```
>>> affiche_début_confinement(11)
jour 35 : 1223589 contaminés : ALERTE CONFINEMENT !!!
```

```
def affiche_début_confinement(c0):
    jour=1
    contaminés = c0
    while contaminés <= 10**6:
        jour = jour+1
        contaminés = jour_suivant(contaminés)
    print("jour",jour,":",contaminés,"contaminé",": ALERTE CONFINEMENT !!!")
```

Exercice 4 Séquencer le génome du COVID . . . . . 7 points

☐0 ☐0,5 ☐1 ☐1,5 ☐2 ☐2,5 ☐3 ☐3,5 ☐4 ☐4,5 ☐5 ☐5,5 ☐6 ☐6,5 ☐7

Le monde du vivant n'utilise pas le langage Python, mais le langage de l'ADN pour coder des protéines. L'objectif de ce problème est de convertir une séquence d'ADN en séquence de protéines.

1. Écrire une fonction `est_adn(chaîne)` qui prend une chaîne de caractères en argument et qui renvoie `True` si la chaîne ne contient que les caractères majuscules `'A'`, `'C'`, `'G'`, `'T'` et `False` sinon.

```
>>> est_adn("gattaca")
False
>>> est_adn("GATTACA")
True
```

```
>>> est_adn("GAT C")
False
>>> est_adn("Choucroute")
False
```

```
def est_adn(chaîne):
    for lettre in chaîne:
        if lettre!='A' and lettre!='G' and lettre!='T' and lettre!='C':
            return False
    return True
```

2. En pratique, les séquences d'ADN, comme les chaussettes, vont toujours par paire. Le complémentaire d'une séquence est obtenue en remplaçant `A` par `T` et `T` par `A` ainsi que `C` par `G` et `G` par `C`. Écrire la fonction `complémentaire(adn)` qui prend une chaîne `adn` et renvoie son complémentaire.

```
>>> complémentaire("AAGGTTCCAGTC")
'TTCCAAGGTCAG'
```

```
def complémentaire(adn):
    s=''
    for e in adn:
        if e=='A':
            s = s + 'T'
        elif e=='T':
            s = s + 'A'
        elif e=='G':
            s = s + 'C'
        elif e=='C':
            s = s + 'G'
    return s
```

Un brin d'ADN est une séquence de nucléotides (A, C, T, G). Pour traduire l'ADN en protéines, on utilise le code génétique : à un codon (séquence de trois nucléotides) on associe une protéine. Chaque protéine sera représentée par une lettre. On se donne une liste `traducteur` qui donne les associations (codon, protéine) sous la forme d'un couple de chaînes de caractères. C'est le fameux code génétique.

```
>>> traducteur[31]
('CGT', 'R')
>>> traducteur[1]
('ATC', 'I')
```

```
>>> traducteur[51]
('TCT', 'S')
>>> traducteur[23]
('CCT', 'P')
```

La liste `traducteur` est définie de manière globale. On pourra donc l'utiliser dans toutes les fonctions.

3. Écrire une fonction `traduire(codon)` qui prend en entrée une chaîne `codon` et qui renvoie la protéine associée dans la liste `traducteur`. Si la chaîne `codon` ne se trouve pas dans la liste `traducteur`, la fonction renverra le caractère `'_'`.

```
>>> traduire("CGT")
'R'
>>> traduire("CFDT")
'_'
```

```
def traduire(codon):
    for e in traducteur:
        (c,p)=e
        if c==codon:
            return p
    return '_'
```

4. On se donne une chaîne `adn` dont la longueur est un multiple de trois. Écrire la fonction `protéines(adn)` qui renvoie la chaîne des protéines associées à chacun des codons.

```
>>> protéines("TCTCCTATCAAAGAA") # TCT CCT ATC AAA GAA
'SPIKE'
```

```
def protéines(adn):
    n=len(adn)//3
    protéine=''
    for i in range(n):
        codon = adn[3*i] + adn[3*i+1] + adn[3*i+2]
        protéine = protéine + traduire_codon(codon)
    return protéine
```

5. Certains codons ne sont associés à aucune protéine. Dans ce cas, le symbole correspondant est donné par le caractère `'_'`. Ainsi une séquence d'ADN code plusieurs séquences de protéines.
Écrire une fonction `liste_protéines(adn)` qui prend une séquence ADN et renvoie la liste des gènes associés.

```
>>> adn
'CTGACGTCAGTGAGTTAGTCTCCTATCAAAGAATAACCATATTAAAGGTCTTAGGCGCTTGAA'
>>> protéines(adn) # Écrite à la question précédente.
'LTSVS_SPIKE_PY_RS_ALE'
>>> liste_protéines(adn)
['LTSVS', 'SPIKE', 'PY', 'RS', 'ALE']
```

```
def liste_protéines(adn):
    chaîne = protéines(adn)
    prot =''
    L=[]
    for p in chaîne:
        if p=='_' and prot !='':
            L.append(prot)
            prot=''
        elif p!='_':
            prot = prot + p
    if prot != '':
        L.append(prot)
    return L
```

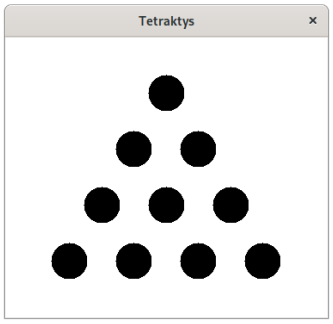
Exercice 5 Déjouer le complot du COVID . . . . . 3 points

☐0 ☐0,5 ☐1 ☐1,5 ☐2 ☐2,5 ☐3

Le COVID est-il un infâme complot de la secte des pythagoriciens, les disciples de Pythagore ? Au risque de vous décevoir la réponse est clairement non, la secte ayant disparu pendant l'Antiquité. Ceci étant, cela nous donne un bon prétexte pour le dernier exercice !
L'objectif est de tracer le symbole des pythagoriciens : le célèbre Τετρακτύς (en français Tetraktys). C'est le triangle composé de quatre lignes que vous trouverez page suivante.
Vous trouverez aussi en début de page suivante, pour information, le code qui permet de créer la fenêtre.
Vous pourrez utiliser la fonction `disque(x,y,r)` qui trace un disque noir de rayon `r` et de centre `x,y`.

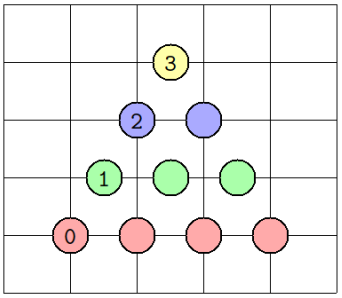

```
import tkinter as tk
Titre = "Tetraktys"
(H,L) = (75*(3)**.5/2, 75)
(Hf,Lf) = (5*H,5*L)

root = tk.Tk()
root.title(Titre)
Dessin=tk.Canvas(root,height=Hf,width=Lf,bg='white')
Dessin.pack()
```



1. Pour vous aider, voici une figure avec pour indication un quadrillage et des couleurs. Ce n'est pas à vous de la tracer, votre objectif est la figure en haut à droite en noir et blanc ! Chaque rectangle est de largeur `L` et de hauteur `H`.

Quelles sont les coordonnées des disques de `0` à `3` ? On donnera la réponse en fonction de `L` et `H`.




```
0:(L,4H)     1:(L+1/2L,3H)
2:(2L,2H)    3:(2L+1/2L,H)
```

2. Écrire une fonction `ligne(x0,y0,n)` qui trace `n` disques, espacés de `L` sur la même ligne en partant du point `(x0,y0)`. On utilisera la fonction du cours `disque(x,y,r)` en choisissant un rayon `r=20`.

```
def ligne(x0,y0,n):
    for i in range(n):
        disque(x0+i*L,y0,20)
```

3. En utilisant la fonction `ligne` et une boucle `for`, écrire le code qui trace le tetraktys en noir.

```
x0 = L
y0 = 4*H
for j in range(4):
    ligne(x0,y0,4-j)
    y0 = y0 - H
    x0 = x0 + 1/2*L
```