+1/1/60+

Programmation impérative en Python — SPUF21

Année 2020-2021 — 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. On 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(…)

# Méthodes autorisées
L.append(x)
'{} = {}'.format('1+1',2)
```

```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 de cours . . . . . 3 points

☐0 ☐1 ☐2 ☐3

1. Définir une liste L avec une boucle `for` correspondant à `[ 3*x+1 for x in range(2,13,5) ]`

```
L=[]
for i in range(2,13,5):
    L.append(3*i+1)
```

2. Écrire avec une boucle `while` le code correspondant à la boucle suivante :

```
for i in range(3,10):
    print(i)
```

```
i=3
while i<10:
    print(i)
    i=i+1
```

3. Écrire la fonction `u(n)` qui *renvoie* l'entier u_n de la suite récurrente définie par :

$$\begin{cases} u_0 &= 7 \\ u_{n+1} &= 2 - 3u_n \end{cases}$$

```
def u(n):
    if n==0:
        return 7
    else:
        return 2-3*u(n-1)
```

Exercice 2 Tapis . . . . . 1,5 points

☐0 ☐0,5 ☐1 ☐1,5

Écrire la fonction `tapis(n)` qui reproduit le motif ci-dessous de largeur et hauteur `n`.

```
>>> tapis(6)
.O.O.O
O.O.O.
.O.O.O
O.O.O.
.O.O.O
O.O.O.
>>> tapis(11)
.O.O.O.O.O.
O.O.O.O.O.O
.O.O.O.O.O.
O.O.O.O.O.O
.O.O.O.O.O.
O.O.O.O.O.O
.O.O.O.O.O.
O.O.O.O.O.O
.O.O.O.O.O.
O.O.O.O.O.O
.O.O.O.O.O.
```

```python
def tapis(n):
    for j in range(n):
        for i in range(n):
            if (i%2==1  and j%2==0) or (i%2==0  and j%2==1):
                print('O',end='')
            else:
                print('.',end='')
        print('')
```

Exercice 3 Rendu de monnaie . . . . . 5 points

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

On cherche à créer un algorithme capable de décomposer de manière optimale une somme d'argent en pièces et billets. Par exemple : 13 € se décompose en 10 € + 2 € + 1 €. Nous allons commencer par générer la liste `M` des pièces et des billets en euros `M = [0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50, 100, 200, 500 ]` sans avoir à l'écrire à la main.

1. Écrire une fonction `générer()` sans arguments qui renvoie la liste de tous les triplets correspondant à des pièces ou des billets en euros. On utilisera une boucle. On pourra utiliser la liste `P=[0.01, 0.1, 1, 10, 100]` que l'on supposera déjà définie.

```
>>> générer()
[(0.01, 0.02, 0.05), (0.1, 0.2, 0.5), (1, 2, 5), (10, 20, 50), (100, 200, 500)]
```

```
def générer():
    L = []
    P = [0.01 , 0.1, 1, 10, 100]
    for m in P:
        L.append((1*m,2*m,5*m))
    return L
```

2. Écrire une fonction `aplatir(L)` qui à partir d'une liste de tuples renvoie la liste des éléments.

```
>>> aplatir([ (1,2,3) , (4,5) , () , (6,7,8) ])
[1, 2, 3, 4, 5, 6, 7, 8]
```

```
def aplatir(L):
    M = []
    for t in L:
        for x in t:
            M.append(x)
    return M
```

3. Définir `M` à partir des fonctions précédentes.

```
M = aplatir(générer())
```

L'algorithme de rendu de monnaie est le suivant : Pour rendre x € (par exemple 17.23 €) il suffit de prendre la plus grande pièce (ou billet) inférieur à x (ici 10 €) puis de recommencer.

4. Écrire une fonction `plus_grand_possible(x)` qui renvoie la plus grande valeur possible de `M` (définie précédemment de manière globale) plus petite que `x` (on suppose $x \geq 0,01$).

```
>>> plus_grand_possible(43.25)
20
```

```
def plus_grand_possible(x): # On suppose n>=.1
    for i in range(len(M)):
        if x<M[i]:
            return M[i-1]
    return M[-1]
```

5. Écrire une fonction `rendre_monnaie(x)` qui affiche le calcul et renvoie la liste des pièces et billets nécessaires pour obtenir x.

```
>>> L=rendre_monnaie(19.03)
19.03€ je rends 10€ reste 9.03€
9.03€ je rends 5€ reste 4.03€
4.03€ je rends 2€ reste 2.03€
2.03€ je rends 2€ reste 0.03€
0.03€ je rends 0.02€ reste 0.01€
0.01€ je rends 0.01€ reste 0.0€
>>> L
[10, 5, 2, 2, 0.02, 0.01]
```

```
def rendre_monnaie(x):
    L=[]
    while x != 0:
        m = plus_grand_possible(x)
        y = x-m # y=round(x-m,2)
        print('{}€ je rends {}€ reste {}€'.format(x,m,y))
        L.append(m)
        x = y
    return L
# Remarque : sans l'utilisation de round (en commentaire), la
# fonction ne marche pas à cause des approximations dues au flottant.
# On ne s'attendait pas à ce que vous pensiez à ce bug !
```

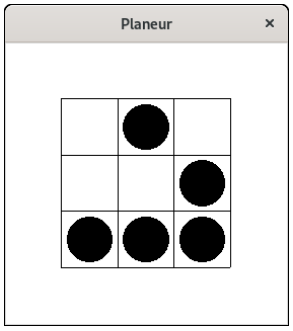
Exercice 4 Planeur . . . . . 5 points

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

Seules deux fonctions seront utiles pour la suite : `Dessin.create_line(p,q)` pour tracer des segments où `p` et `q` sont des couples de coordonnées ; et `disque(x,y,r)` qui permet de tracer un disque de centre `(x,y)` et de rayon `r`.

On cherche à reproduire l'image ci-dessous. On suppose que les marges autour de la figure sont de la même largeur et hauteur que les cases. Le code de la création de la fenêtre est donné ci-dessous.

```
1  import tkinter as tk
2  Titre = "Planeur"
3  Dimension=277
4  (H,L) = (Dimension,Dimension)
5
6  root = tk.Tk()
7  root.title(Titre)
8
9  Dessin = tk.Canvas(root,height=H,width=L,bg='white')
10 Dessin.pack()
```



1. Définissez une variable `delta` pour définir la largeur et la hauteur commune aux cases et aux marges. en fonction de la variable `Dimension`. Pour la suite on supposera que `delta` est définie de manière globale.

```
delta = Dimension/5
```

2. Écrire une fonction `quadrillage()` (avec une ou plusieurs boucles `for`) qui affiche le quadrillage. Votre code devra utiliser la valeur de `delta` précédemment définie.

```
for k in range(1,5):
    Dessin.create_line((delta,k*delta),(L-delta,k*delta))
    Dessin.create_line((k*delta,delta),(k*delta,H-delta))
```

3. On numérote chacune des cases par deux coordonnées `(i,j)` allant chacune de `1` à `3` comme indiqué sur le schéma ci-dessous. Écrire une fonction `traduire(i,j)` qui renvoie les coordonnées du centre de la case de coordonnées `(i,j)`.

(1,3)	(2,3)	(3,3)
(1,2)	(2,2)	(3,2)
(1,1)	(2,1)	(3,1)

```
def traduire(i,j):
    x = (i + 0.5) * delta
    y = (5 - (j + 0.5)) * delta
    return (x,y)
```

4. Écrire une fonction `figure(L)`, qui prend une liste de couples et affiche un disque de rayon `delta*3/8` au centre de chacun des carrés correspondants. Ainsi on pourra tracer la figure en prenant `L=[ (1,1), (2,1), (3,1), (3,2), (2,3) ]`.

```
def figure(L):
    for p in L:
        (i,j) = p
        (x,y) = traduire(i,j)
        disque(x,y,delta*3/8)
```

Exercice 5 Correcteur orthographique . . . . . 5,5 points

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

On se donne une liste `dico` définie de manière globale et contenant les mots du dictionnaire. L'objectif de l'exercice est d'utiliser cette liste pour implémenter un correcteur orthographique. Si un mot n'est pas dans la variable `dico`, on le remplace par le mot le plus proche.

1. Calculer une fonction `différence(mot1,mot2)` qui calcule le nombre de différences entre les deux mots. Si un mot possède plus de lettres que l'autre, chacune des lettres supplémentaires comptera pour une différence.

```
>>> différence('abcdef','a-cd')
3
```

```
def différence(m1,m2):
    if len(m1)>len(m2):
        (m,M)=(m2,m1)
    else:
        (m,M)=(m1,m2)
    diff = len(M)-len(m)
    for i in range(len(m)):
        if m[i]!= M[i]:
            diff=diff+1
    return diff
```

2. Écrire une fonction : `plus_petite_différence(mot)` qui renvoie la plus petite différence entre `mot` et un mot du dictionnaire. Si `mot` est déjà dans le dictionnaire, la fonction devra renvoyer 0.

```
>>> plus_petite_différence('python') # 0 car c'est un mot du dictionnaire
0
>>> plus_petite_différence('pithonb')# 2 car pithonb a 2 différences avec python
2
```

```
def plus_petite_différence(m):
    ppd=différence(m,dico[0])
    for mot in dico:
        if différence(m,mot) < ppd:
            ppd = différence(m,mot)
    return ppd
```

3. Écrire une fonction `mots_les_plus_proches(mot)` qui renvoie la liste des mots du dictionnaire les plus proches de `mot`. Vous pouvez utiliser la fonction `plus_petite_différence(mot)`, mais essayez de vous en passer afin d'obtenir tous les points.

```
def mots_les_plus_proches(mot):
    ppd = différence(mot,dico[0])
    L = []
    for m in dico:
        if différence(m,mot) == ppd:
            L.append(m)
        elif différence(m,mot) < ppd:
            ppd=différence(m,mot)
            L=[m]
    return L
```

4. Écrire une fonction `correcteur_orthographique(L)` qui prend en entrée une liste de mots correspondant à un texte et qui remplace chaque mot de ce texte n'apparaissant pas dans le dictionnaire par un mot au hasard parmi ceux qui sont les plus proches. On fera usage de la fonction `randint(a,b)` qui renvoie un nombre aléatoire entre `a` et `b`. Par exemple `randint(0,3)` peut retourner 4 valeurs possibles : 0, 1, 2 ou 3.

```
def correcteur_orthographique(L):
    for i in range(len(L)):
        possibles = mots_les_plus_proches(L[i])
        a = randint(0,len(possibles)-1)
        L[i]=possibles[a]
    return L
```