+1/1/60+

Programmation impérative en Python — SPUF21

Année 2020-2021 — Examen terminal

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(…)

# 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 ☐0,5 ☐1 ☐1,5 ☐2 ☐2,5 ☐3

1. Définir une fonction `miroir(mot)` qui prend une chaine de caractères `mot` et qui **affiche** une lettre par ligne en inversant l'ordre des lettres. Par exemple on aura :

```
>>> miroir("rats")
s
t
a
r
```

```
def miroir(s):
    n = len(s)
    for i in range(n-1,-1,-1):
        print(s[i])
```

2. Calculer la somme des entiers pairs $2+4+6+\ldots$ avec une boucle `while` jusqu'à ce que le résultat soit un multiple non nul de 27. On affichera le résultat obtenu.

```
s=2
i=4
while s%27!=0:
    s = s+i
    i = i+2

print(s)
```

Exercice 2 Arbres et Logique . . . . . 6 points

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

Normalement, la logique classique ne contient que deux valeurs : `True` et `False`. Ceci étant, pour modéliser des situations plus complexes, on souhaite introduire une troisième valeur correspondant au cas où l'on ne sait pas : `None`. On se donne le code ci-dessous :

```
1  def et(a,b):
2      if a==False or b==False:
3          return False
4      elif a==None  or b==None:
5          return None
6      else:
7          return True
8
9  def ou(a,b):
10     if a==True or b==True:
11         return True
12     elif a==None  or b==None:
13         return None
14     else:
15         return False
```

1. Écrire une fonction `est_bool(x)` qui renvoie `True` si et seulement si x est une des trois valeurs : `True`, `False` ou `None`. Dans le cas contraire elle renverra `False`.

```
def est_bool(x):
    return x in [True, False, None]
```

2. Donner le résultat du calcul : « `et(ou(True,None),ou(False,None))` ». On détaillera le calcul avec les étapes intermédiaires.

`ou(True,None)` vaut `True`

`ou(None,False)` vaut `None`

`et(True,None)` vaut `None`

3. Faire un programme `table_du_et()` qui affiche avec des boucles `for` la table de la fonction `et`. En particulier on doit obtenir le résultat ci-dessous (l'alignement n'est pas demandé).

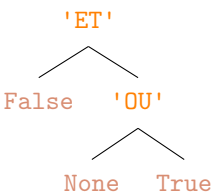
```
>>> table_du_et()
True  et True  vaut True
True  et False vaut False
True  et None  vaut None
False et True  vaut False
False et False vaut False
False et None  vaut False
None  et True  vaut None
None  et False vaut False
None  et None  vaut None
```

```python
def table_du_et():
    for x in [True, False, None]:
        for y in [True, False, None]:
            print("{} et {} vaut {}".format(x,y,et(x,y)))
```

Les arbres que nous considérerons dans cet exercice auront pour feuille une des trois valeurs : `True`, `False` ou `None` et les nœuds ne pourront prendre que deux valeurs : `'ET'` ou `'OU'`.

On ne s'intéresse pas à la façon dont ces arbres sont représentés, nous utiliseront uniquement les fonctions suivantes :

- `arbre(r, Ag, Ad)` qui renvoie un arbre de racine `r` et de fils `Ag` (gauche) et `Ad` (droit) ;
- `est_operateur(obj)` qui renvoie `True` si `obj` est un des opérateurs `"ET"` ou `'OU'`
- `est_feuille(obj)` qui renvoie `True` si `obj` est une feuille, et `False` sinon ;
- `racine(A)` qui renvoie la racine de l'arbre `A`
- `fg(A)` qui renvoie le fils gauche de l'arbre `A`
- `fd(A)` qui renvoie le fils droit de l'arbre `A`.



4. En utilisant uniquement les fonctions ci-dessus, définir une variable `Ar` correspondant à l'arbre ci-dessus.

```
arbre('ET',False,arbre('OU',None,True))
```

5. Écrire une fonction `logique_classique(A)` qui renvoie `True` si et seulement si les feuilles de `A` valent `True` ou `False` ; la fonction renverra `False` si une feuille contient `None`.

```
def logique_classique(A):
    if est_feuille(A):
        return A in [True, False] # A != None
    else:
        return logique_classique(fg(A)) and logique_classique(fd(A))
```

6. Écrire une fonction `calcul(A)` qui calcule la valeur correspondant à l'expression associée à l'arbre `A`.

```
def calcul(A):
    if est_feuille(A):
        return A
    else:
        g = calcul(fg(A))
        d = calcul(fd(A))
        if racine(A)=='ET':
            return et(g,d)
        else: # racine(A)=='OU'
            return ou(g,d)
```

Exercice 3 Calcul des notes . . . . . 5 points

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

Une promotion d'étudiants est représentée sous la forme suivante qui associe à chaque numéro d'étudiant le nom de l'étudiant correspondant :

```
promo = { 126:'Valjean' , 324:'Gavroche' , 213:'Phantine' }
```

1. Quel est le type de `promo` (liste ? matrice ? entier ? autre (précisez) ?)

```
# promo est de type : dictionnaire
```

2. Zut alors, l'enseignant s'est trompé en écrivant le nom de l'étudiante 213 : son vrai nom est Fantine et non Phantine. En une ligne, corrigez le nom de celle-ci dans `promo`.

```
promo[213]='Fantine'
```

3. Ajoutez à `promo` une nouvelle étudiante dénommée Cosette dont le numéro étudiant est 404

```
promo[404]='Cosette'
```

Les notes du partiel sont stockées dans une liste de couples regroupant l'identifiant et la note de l'élève.

```
résultats = [ (213,17.5), (324,8), (126,12), (404,4) ]
```

4. Écrire une fonction `moyenne(liste)` qui prend une liste non vide de couples comme `résultats` et calcule la moyenne des notes. On utilisera une boucle `for` (la fonction `sum` ne doit pas être utilisée).

```
def moyenne(liste):
    somme = 0
    for x in liste:
        (identifiant,note) = x
        somme = somme + note
    return somme/len(liste)
```

5. Écrire une fonction `valide_id(résultats)` qui renvoie la liste de tous les identifiants d'étudiant ayant validé l'UE (c'est-à-dire, ceux qui ont 10 ou plus).

```
def valide_id(résultats):
    L = []
    for c in résultats:
        if c[1] >= 10:
            L.append(c[0])
    return L
```

6. Écrire une fonction `valide_nom(résultats,promo)` qui utilise la fonction précédente et qui renvoie l'**ensemble** des noms des étudiants qui valident.

```
def valide_nom(résultats,promo):
    liste_id = valide_id(résultats)
    e = set()
    for identifiant in liste_id:
        e.add(promo[identifiant])
    return e
```

Exercice 4 Courbes . . . . . 6 points

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

On veut représenter une courbe comme une liste de points $p_0, p_1, p_2, \ldots$ qu'il suffit de relier. Pour cela, on va représenter notre image comme une matrice. Si le point p_k d'indice `k` est présent sur la case `(i,j)`, on aura alors `img[i][j]==k`. Si aucun point ne se trouve sur le pixel de coordonnée `(i,j)`, on aura par défaut `img[i][j]==None`. On pourra utiliser la fonction `dimensions(M)` du cours qui renvoie un couple `(n,m)` donnant le nombre de lignes et de colonnes d'une matrice `M`.

```
1 img = [ [ None, None,    1 ],
2         [    2, None, None ],
3         [ None,    3,    0 ] ]
```

	0	1	2
0			1
1	2		
2		3	0

1. Créez une fonction `nouvelle_liste(n)` qui renvoie une liste de taille `n` ne contenant que des `None` (on utilisera une boucle `for`).

```
def nouvelle_liste(n):
    L=[]
    for i in range(n):
        L.append(None)
    return L
```

2. Créez une fonction `entier_max(img)` qui renvoie le plus grand entier d'une matrice `img`. On suppose que la matrice ne contient que des `None` et un ou plusieurs entiers (au moins l'entier `0`).

```
def entier_max(img):
    (n,m)=dimensions(img) #  ou  n,m = len(img),len(img[0])
    maxi=0
    for i in range(n):
        for j in range(m):
            if img[i][j] != None and img[i][j] > maxi:
                maxi=img[i][j]
    return maxi
```

3. Écrire une fonction `matrice_vers_liste(img)` qui renvoie la liste `L` des points. En particulier on aura `L[k]==(i,j)` si et seulement si `img[i][j]==k`. Par exemple on aura :

```
>>> matrice_vers_liste(img) # L[0]==(2,2) L[1]==(0,2) L[2]==(1,0) L[3]==(2,1)
[(2, 2), (0, 2), (1, 0), (2, 1)]
```

```
def matrice_vers_liste(img):
    (n,m)=dimensions(img) #  ou  n,m = len(img),len(img[0])
    L = nouvelle_liste(entier_max(img)+1)
    for i in range(n):
        for j in range(m):
            if img[i][j] != None:
                L[img[i][j]] = (i,j)
    return L
```

4. Écrire une fonction `distance(p,q)` qui prend deux points `p=(x1,y1)` et `q=(x2,y2)` et renvoie la distance entre ces deux points.

```
def distance(p,q):
    (x1,y1)=p
    (x2,y2)=q
    return sqrt((x2-x1)**2 + (y2-y1)**2)
```

5. Écrire une fonction `longueur(L)` qui prend une liste de points en entrée et renvoie la longueur totale du chemin.

```
def longueur(L):
    n = len(L)
    d=0
    for i in range(n-1):
       d = d+distance(L[i],L[i+1])
    return d
```

6. Écrire une fonction `supprime(img,i,j)` qui supprime l'élément `img[i][j]` en remplaçant par `None`. Si nécessaire, il faudra modifier les valeurs entières dans `M` pour éviter un « trou » dans la suite. Par exemple `supprime(img,0,2)` modifiera la matrice `img` de la façon suivante :

```
img = [ [ None, None, None ],
        [    1, None, None ],
        [ None,    2,    0 ] ]
```

	0	1	2
0			
1	1		
2		2	0

```
def supprime(img,i,j):
    if img[i][j] != None:
        k = img[i][j]
        img[i][j] = None
        (n,m)=dimensions(img)
        for i in range(n):
            for j in range(m):
                if img[i][j] != None and img[i][j] > k:
                    img[i][j] = img[i][j] - 1
```