+1/1/60+

Programmation impérative en Python — SPUF21

Année 2021-2022 — 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.

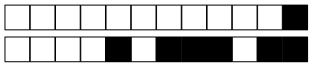
À part les méthodes et fonctions de base, vous n'avez pas le droit d'utiliser les fonctions et les méthodes « avancées », sauf si l'énoncé vous conseille l'utilisation de certaines d'entre elles.

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

# Méthodes autorisées
L.append(x) (ainsi que son équivalent pour les ensembles)
```

```
# 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.
# 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 ☐ 0,5 ☐ 1 ☐ 1,5 ☐ 2 ☐ 2,5 ☐ 3

1. Construire avec une boucle une liste `L = [0.0, 0.5, … , 19.5, 20.0]` contenant toutes les valeurs de 0 à 20 avec un pas de $\frac{1}{2}$.

```
L=[]
for i in range(0,41): # Les pas de 0.5 sont interdits !
    L.append(i/2)
```

2. On se donne une chaîne *s*. En utilisant une boucle `for`, définir un ensemble `E` contenant tous les caractères de `s`. On n'aura évidemment pas le droit de faire `E = set(s)`.

```
E = set()  # Attention E = {} définit un dictionnaire !
for c in s:
    E.add(c)
```

3. On définit ci-dessous une fonction `construire_matrice(n)` qui renvoie une matrice de `n` lignes et `n` colonnes.

```
def construire_matrice(n):
    ligne = [0] * n
    M = []
    for i in range(n):
        M.append(ligne)
    return M
```

En utilisant cette fonction `construire_matrice(n)` on initialise une matrice `M` et on la modifie. Que vaut `M` après la modification ? Justifier.

```
>>> M = construire_matrice(3)
>>> M
[[0, 0, 0], [0, 0, 0], [0, 0, 0]]
>>> M[0][0] = 5
```

```
M vaut [ [5,0,0] , [5,0,0] , [5,0,0] ].
En effet, M contient 3 fois la même liste (même adresse en mémoire),
donc si on en modifie une, on modifie automatiquement les autres.
```

Exercice 2 Transposée d'une matrice . . . . . 3 points

☐0 ☐0,5 ☐1 ☐1,5 ☐2 ☐2,5 ☐3

Pour cet exercice, il vous est conseillé d'utiliser, si nécessaire, les deux fonctions vues en cours : `dimensions(M)` qui renvoie un couple `(n,m)` correspondant aux nombres de lignes et de colonnes de la matrice `M` et `matrice_nulle(n,m)` qui crée une nouvelle matrice de dimensions $n \times m$.

Par la suite, on ne considérera que des matrices carrées. On s'intéresse à la transposée d'une matrice M. La transposée de la matrice M, se note ${}^tM$ et vérifie pour tous indices i et j la formule ${}^tM_{i\,j} = M_{j\,i}$

Exemple : $${}^t\begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{pmatrix} = \begin{pmatrix}1 & 4 & 7\\ 2 & 5 & 8\\ 3 & 6 & 9\end{pmatrix}$$

1. Écrire une fonction `transposée(M)` qui renvoie la matrice ^{t}`M` sans modifier `M`.

```
def transposée(M):
    (n,m) = dimensions(M) # on a forcément n=m
    S = matrice_nulle(m,n)
    for i in range(m):
        for j in range(n):
            S[i][j] = M[j][i]
    return S #
```

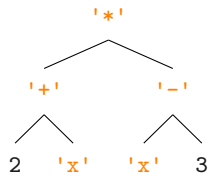
2. Écrire une fonction `transpose(M)` qui modifie la matrice mais ne renvoie rien.

```
def transpose(M): # M = transposée(M) suivie de return ne modifie pas M !
    (n,m) = dimensions(M)
    S = matrice_nulle(m,n)
    for i in range(m):
        for j in range(i): # on ne traite que le cas où j<i :
            temp = M[i][j] # dans cet exercice, for j in range(n) est faux.
            M[i][j] = M[j][i]
            M[j][i] = temp
```

Exercice 3 Des polynômes et des arbres . . . . . 3 points

☐0 ☐0,5 ☐1 ☐1,5 ☐2 ☐2,5 ☐3

Dans cet exercice, vous ne pouvez utiliser que les fonctions suivantes :
— `arbre(r,Ag,Ad)` renvoie un arbre de racine `r` et de fils `Ag` (gauche) et `Ad` (droit) ;
— `est_feuille(A)` renvoie `True` si `A` est une feuille, et `False` sinon ;
— `racine(A)` renvoie la racine de l'arbre `A` : `'+'`, `'*'` ou `'-'` ;
— `fg(A)` (ou `fd(A)`) renvoie le fils gauche (ou droit) de `A`.



1. À quelle expression correspond l'arbre ci-dessus ?

```
(2 + x) * (x - 3)
```

2. Écrire une fonction `est_constant(A)` qui renvoie `True` si toutes les feuilles de `A` ont un type égal à `int` et `False` sinon. On supposera donc que les feuilles sont soit des chaînes soit des entiers.

```
def est_constant(A):
    if est_feuille(A):
        return type(A) == int
    else:
        g = est_constant(fg(A))
        d = est_constant(fd(A))
        return g and d
```

3. Horreur, des physiciens rôdent sur le campus et créent des polynômes avec la variable t ! Écrire une fonction `nettoyer(A)` qui renvoie l'arbre semblable à `A` dans lequel les `'t'` sont remplacés par des `'x'`.

```
def nettoyer(A):
    if est_feuille(A):
        if A=='t':
            return 'x'
        else:
            return A
    else:
        g = nettoyer(fg(A))
        d = nettoyer(fd(A))
        return arbre(racine(A),g,d)
```

Exercice 4 Des polynômes et des listes . . . . . 5 points

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

Travailler sur des polynômes avec des arbres n'est guère pratique. On va alors représenter les polynômes avec des listes. Concrètement le polynôme $5+3x-12x^2+7x^3$ sera représenté par la liste `[5,3,-12,7]`.

1. Écrire une fonction `calcul(P,x0)` qui prend une liste P représentant un polynôme p et renvoie la valeur de $p(x_0)$. Ainsi `calcul([3,2,5],10)` renverra $3 + 2x_0 + 5{x_0}^2 = 3 + 2 \times 10 + 5 \times 10^2 = 523$.

```
def calcul(P,x0):
    résultat = 0
    for k in range(len(P)):
        résultat = résultat + P[k] * x0**k
    return résultat
```

2. Écrire une fonction `degré(P)` qui renvoie le degré du polynôme `P`, c'est-à-dire le plus grand indice `i` tel que `P[i]` soit non nul. Ainsi, `degré([2,0,5,4,0,0,0])` renverra 3 (car ici, $P(x) = 2 + 5x^2 + 4x^3$). Si tous les coefficients sont nuls, on renverra alors `0`.

```
def degré(P):
    d = 0
    for i in range(len(P)):
        if P[i] != 0:
            d = i
    return d
```

3. Écrire une fonction `réduire(P)` qui renvoie une nouvelle liste `Q` mais sans `0` inutiles à droite de la liste `P`. Par exemple `réduire([2,0,5,4,0,0,0])` renverra `[2,0,5,4]`.

```
def réduire(P):
    deg = degré(P)
    Q = []
    for i in range(deg+1):
        Q.append(P[i])
    return Q
```

4. Écrire une fonction `somme(P,Q)` qui renvoie la liste correspondant à la somme des deux polynômes `P` et `Q`. Rappel : $(a+bx+cx^2)+(a'+b'x+c'x^2+d'x^3)=(a+a')+(b+b')x+(c+c')x^2+d'x^3$

```
def somme(P,Q):
    if len(P) < len(Q):
        P,Q = Q,P
    R=[]
    for i in range(len(Q)):
            R.append(P[i] + Q[i])
    for i in range(len(Q),len(P)):
            R.append(P[i])
    return R
```

5. Écrire une fonction `limites(P)` qui renvoie le couple (m,p) où m est la limite de P en $-\infty$ et p en $+\infty$. On suppose que `degré(P)>0`. On aura par exemple, `limite([0,5,0,-2]) == ("+ϖ","-ϖ")` car le degré de $5x-2x^3$ (ici 3) est impair et car le coefficient dominant (ici -2) est négatif. Rappel :

Coefficient dominant	**Négatif**	Positif	**Négatif**	Positif
Degré pair	$-\infty$	$+\infty$	$-\infty$	$+\infty$
Degré impair	$+\infty$	$-\infty$	$-\infty$	$+\infty$
	Limite en $-\infty$		Limite en $+\infty$	

```
def limites(P):
    Q = réduire(P)
    deg = degré(Q)
    if Q[-1] >0 and deg%2==0:
        return ("+ϖ","+ϖ")
    elif Q[-1] < 0 and deg%2==0:
        return ("-ϖ","-ϖ")
    elif Q[-1] >0 and deg%2==1:
        return ("-ϖ","+ϖ")
    else
        return ("+ϖ","-ϖ")
```

Exercice 5 Des polynômes et des racines . . . . . 3 points

☐0 ☐0,5 ☐1 ☐1,5 ☐2 ☐2,5 ☐3

Dans cet exercice, vous pouvez utilisez la fonction `calcul(P,x)` qui renvoie la valeur $P(x)$ d'un polynôme (codé avec des listes). On pourra aussi utiliser la fonction `abs(y)` qui calcule la valeur absolue |y| de `y`.

On considère maintenant un polynôme `P` de degré impair dont le coefficient dominant est positif. Ainsi `P` a pour limite $+\infty$ en $+\infty$ (et $-\infty$ en $-\infty$). Une des conséquences est que pour x assez grand, $P(x)$ est positif, de même pour x assez petit, $P(x)$ est négatif.

1. Écrire une fonction `valeur_positive(P)` qui cherche et renvoie une valeur de x tel que $P(x) > 0$. On pourra essayer des valeurs de x de plus en plus grandes jusqu'à trouver une valeur qui convienne.

```
def valeur_positive(P):
    x = 0
    while calcul(P,x) <= 0:
        x = x + 1
    return x
```

On pourrait écrire de même une fonction `valeur_négative(P)`. Mais cela ne vous est pas demandé.

2. On suppose qu'avec les deux fonctions de la question précédente on a trouvé deux valeurs a et b telles que $P(a) \leq 0 \leq P(b)$. Écrire une fonction `racine(P,h,a,b)` qui, en utilisant un algorithme de recherche dichotomique, trouve et renvoie une valeur approchée d'une racine de P : c'est-à-dire un nombre x_r vérifiant $|P(x_r)| < h$. Indice : on peut considérer m milieu de $[a, b]$ et mettre à jour a et/ou b à chaque étape afin que l'intervalle $[a, b]$ diminue tout en conservant la propriété : $P(a) \leq 0 \leq P(b)$.

```
def racine(P,h,a,b):
    c = (a+b)/2
    while abs(calcul(P,c)) >= h:
        if calcul(P,c) > 0:
            b = c
        else:
            a = c
        c = (a+b)/2
    return c
```

Exercice 6 Élections . . . . . 3 points

☐0 ☐0,5 ☐1 ☐1,5 ☐2 ☐2,5 ☐3

Dans un pays fictif, des citoyens déçus par le résultat de l'élection présidentielle ont décidé de fuir dans l'arrière-pays de Nice pour y créer un groupe de résistance dans le maquis. Pour guider leur nouvelle commune anarchisante et auto-gérée, ils décident d'élire deux chefs sans pouvoir, le pouvoir étant réparti entre tous. Les bulletins de votes sont stockés sous forme de chaînes dans une liste de la forme : `['Honoré Leprof', 'Noé Meslacets', 'Enrichir Sescollègues', 'Honoré Leprof', …]`

1. Écrire une fonction `compter(liste)`, qui à partir d'une liste de noms renvoie le dictionnaire donnant le nombre de voix pour chacun de ces noms.

```
>>> voix = compter(liste) ; print(voix["Honoré Leprof"],  voix["Thésée Vous"])
184 123
```

```
def compter(liste):
    dico = dict()
    for nom in liste:
        if nom in dico:
            dico[nom] = dico[nom] + 1
        else:
            dico[nom] = 1
    return dico
```

2. Écrire une fonction `gagnants(voix)` qui à partir du dictionnaire de la question précédente affiche les deux candidats avec le plus de voix (en cas d'égalité entre deux seconds, peu importe celui qu'on choisit). Pour récupérer deux noms quelconques, on pourra utiliser :

```
>>> tuple(voix)[0:2]
('Thésée Vous', 'Pompée Félix')
```

```
def gagnants(voix):
    (premier,second) = tuple(voix)[0:2]
    if voix[second] > voix[premier]:
        (premier,second) = (second,premier)
    for nom in voix:
        if voix[nom] > voix[premier]:
                (premier,second) = (nom,premier)
        elif voix[nom] > voix[second]:
                (premier,second) = (premier,nom)
  print("Le second tour sera entre", premier,"et", second)
```

```
def gagnants(voix): # Autre version
    (nom1,nom2) = tuple(voix)[0:2] # On récupère deux noms les initialisations
    premier = nom1
    for nom in voix:
        if voix[nom] > voix[premier]:
            premier = nom
    second = nom1
    if premier == second:
        second = nom2
   for nom in voix:
        if voix[nom] > voix[second] and nom != premier:
            second = nom
   print("Le second tour sera entre", premier,"et", second)
```