0.4 Les structures linéaires

NSI terminale - JB Duthoit

EXERCICE 2 (4 points)

Cet exercice est consacré à l'analyse et à l'écriture de programmes récursifs.

1.

a. Expliquer en quelques mots ce qu'est une fonction récursive.

b. On considère la fonction Python suivante :

Numéro de lignes	Fonction `compte_rebours`
1	`def compte_rebours(n):`
2	`    """ n est un entier positif ou nul """`
3	`    if n >= 0:`
4	`        print(n)`
5	`        compte_rebours(n - 1)`

L'appel `compte_rebours(3)` affiche successivement les nombres 3, 2, 1 et 0. Expliquer pourquoi le programme s'arrête après l'affichage du nombre 0.

2. En mathématiques, la factorielle d'un entier naturel *n* est le produit des nombres entiers strictement positifs inférieurs ou égaux à *n*. Par convention, la factorielle de 0 est 1. Par exemple :

- la factorielle de 1 est 1
- la factorielle de 2 est 2 x 1 =2
- la factorielle de 3 est 3 x 2 x 1 = 6
- la factorielle de 4 est 4 x 3 x 2 x 1 = 24…

Recopier et compléter sur votre copie le programme donné ci-dessous afin que la fonction récursive `fact` renvoie la factorielle de l'entier passé en paramètre de cette fonction.

Exemple : `fact(4)` renvoie 24.

Numéro de lignes	Fonction `fact`
1	`def fact(n):`
2	`    """ Renvoie le produit des nombres entiers`
3	`    strictement positifs inférieurs à n """`
4	`    if n == 0:`
5	`        return` ***à compléter***
6	`    else:`
7	`        return` ***à compléter***

3. La fonction `somme_entiers_rec` ci-dessous permet de calculer la somme des entiers, de `0` à l'entier naturel `n` passé en paramètre.

Par exemple :

- Pour `n` = 0, la fonction renvoie la valeur 0.
- Pour `n` = 1, la fonction renvoie la valeur 0 + 1 = 1.

…

- Pour `n` = 4, la fonction renvoie la valeur 0 + 1 + 2 + 3 + 4 = 10.

Numéro de lignes	Fonction `somme_entiers_rec`

1
2
3
4
5
6
7
8
	`def somme_entiers_rec(n):`
	`    """ Permet de calculer la somme des entiers,`
	`    de 0 à l'entier naturel n """`
	`    if n == 0:`
	`        return 0`
	`    else:`
	`        print(n) #pour vérification`
	`        return n + somme_entiers_rec(n - 1)`

L'instruction `print(n)` de la ligne 7 dans le code précédent a été insérée afin de mettre en évidence le mécanisme en œuvre au niveau des appels récursifs.

a. Écrire ce qui sera affiché dans la console après l'exécution de la ligne suivante :

```
res = somme_entiers_rec(3)
```

b. Quelle valeur sera alors affectée à la variable `res` ?

4. Écrire en Python une fonction `somme_entiers` non récursive : cette fonction devra prendre en argument un entier naturel `n` et renvoyer la somme des entiers de 0 à `n` compris. Elle devra donc renvoyer le même résultat que la fonction `somme_entiers_rec` définie à la question *3*.

Exemple : `somme_entiers(4)` renvoie 10.

EXERCICE 4 (4 points)

Cet exercice traite du thème « structures de données », et principalement des piles.

La classe `Pile` utilisée dans cet exercice est implémentée en utilisant des listes Python et propose quatre éléments d'interface :

- Un constructeur qui permet de créer une pile vide, représentée par `[]` ;
- La méthode `est_vide()` qui renvoie `True` si l'objet est une pile ne contenant aucun élément, et `False` sinon ;
- La méthode `empiler` qui prend un objet quelconque en paramètre et ajoute cet objet au sommet de la pile. Dans la représentation de la pile dans la console, cet objet apparaît à droite des autres éléments de la pile ;
- La méthode `depiler` qui renvoie l'objet présent au sommet de la pile et le retire de la pile.

Exemples :

```
>>> mapile = Pile()
>>> mapile.empiler(2)
>>> mapile
[2]
>>> mapile.empiler(3)
>>> mapile.empiler(50)
>>> mapile
[2, 3, 50]
>>> mapile.depiler()
50
>>> mapile
[2, 3]
```

La méthode `est_triee` ci-dessous renvoie `True` si, en dépilant tous les éléments, ils sont traités dans l'ordre croissant, et `False` sinon.

```
1 def est_triee(self):
2   if not self.est_vide() :
3      e1 = self.depiler()
4      while not self.est_vide():
5          e2 = self.depiler()
6          if e1 ... e2 :
7              return False
8          e1 = ...
9     return True
```

1. Recopier sur la copie les lignes 6 et 8 en complétant les points de suspension.

On créé dans la console la pile `A` représentée par `[1, 2, 3, 4]`.

2. a. Donner la valeur renvoyée par l'appel `A.est_triee()`.
   b. Donner le contenu de la pile `A` après l'exécution de cette instruction.

On souhaite maintenant écrire le code d'une méthode `depileMax` d'une pile non vide ne contenant que des nombres entiers et renvoyant le plus grand élément de cette pile en le retirant de la pile.
Après l'exécution de `p.depileMax()`, le nombre d'éléments de la pile `p` diminue donc de 1.

```
1   def depileMax(self):
2       assert not self.est_vide(), "Pile vide"
3       q = Pile()
4       maxi = self.depiler()
5       while not self.est_vide() :
6          elt = self.depiler()
7          if maxi < elt :
8             q.empiler(maxi)
9             maxi = ...
10          else :
11             ...
12       while not q.est_vide():
13          self.empiler(q.depiler())
14       return maxi
```

3. Recopier sur la copie les lignes 9 et 11 en complétant les points de suspension.

On créé la pile `B` représentée par [9, -7, 8, 12, 4] et on effectue l'appel `B.depileMax()`.

4. a. Donner le contenu des piles `B` et `q` à la fin de chaque itération de la boucle `while` de la ligne 5.
   b. Donner le contenu des piles `B` et `q` avant l'exécution de la ligne 14.
   c. Donner un exemple de pile qui montre que l'ordre des éléments restants n'est pas préservé après l'exécution de `depileMax`.

On donne le code de la méthode `traiter()` :

```
1   def traiter(self):
2       q = Pile()
3       while not self.est_vide():
4           q.empiler(self.depileMax())
5       while not q.est_vide():
6           self.empiler(q.depiler())
```

5. a. Donner les contenus successifs des piles `B` et `q`
   - avant la ligne 3,
   - avant la ligne 5,
   - à la fin de l'exécution de la fonction `traiter`

   lorsque la fonction `traiter` est appliquée sur la pile `B` contenant [1, 6, 4, 3, 7, 2].

   b. Expliquer le traitement effectué par cette méthode.

EXERCICE 1 (4 points)

Cet exercice composé de deux parties A et B, porte sur les structures de données.

Partie A : Expression correctement parenthésée

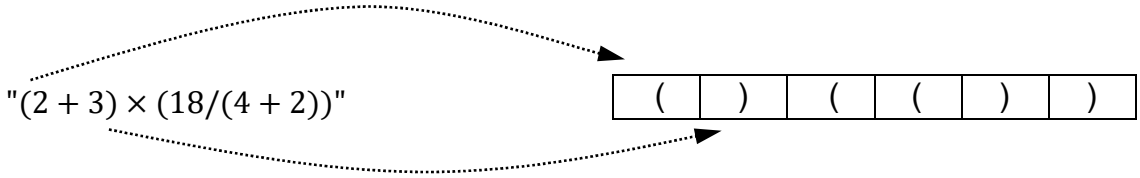
On veut déterminer si une expression arithmétique est correctement parenthésée.

Pour chaque parenthèse fermante ")" correspond une parenthèse précédemment ouverte "(".

Exemples :

- L'expression arithmétique "$(2+3)\times(18/(4+2))$" est correctement parenthésée.
- L'expression arithmétique "$(2+3)\times(18/(4+2$" est non correctement parenthésée.

Pour simplifier les expressions arithmétiques, on enregistre, dans une structure de données, uniquement les parenthèses dans leur ordre d'apparition. On appelle expression simplifiée cette structure.



1. Indiquer si la phrase « les éléments sont maintenant retirés (pour être lus) de cette structure de données dans le même ordre qu'ils y ont été ajoutés lors de l'enregistrement » décrit le comportement d'une file ou d'une pile. Justifier.

Pour vérifier le parenthésage, on peut utiliser une variable `controleur` qui :

- est un nombre entier égal à 0 en début d'analyse de l'expression simplifiée ;
- augmente de 1 si l'on rencontre une parenthèse ouvrante "(" ;
- diminue de 1 si l'on rencontre une parenthèse fermante ")".

Exemple : On considère l' expression simplifiée A : "( )(( ))"
Lors de l'analyse de l'expression A, `controleur` (initialement égal à 0) prend successivement pour valeur 1, 0, 1, 2, 1, 0. Le parenthésage est correct.

2. Écrire, pour chacune des 2 expressions simplifiées B et C suivantes, les valeurs successives prises par la variable `controleur` lors de leur analyse.
Expression simplifiée B : " (((( )( )"
Expression simplifiée C : "(( ))("

3. L'expression simplifiée B précédente est mal parenthésée (parenthèses fermantes manquantes) car le `controleur` est différent de zéro en fin d'analyse. L'expression simplifiée C précédente est également mal parenthésée (parenthèse fermante sans parenthèse ouvrante) car le `controleur` prend une valeur négative pendant l'analyse.
Recopier et compléter uniquement les lignes 13 et 16 du code ci-dessous pour que la fonction `parenthesage_correct` réponde à sa description.

```
1 def parenthesage_correct(expression):
2    ''' fonction retournant True si l'expression arithmétique
3    simplifiée (str) est correctement parenthésée, False
4    sinon.
5    Condition: expression ne contient que des parenthèses
6    ouvrantes et fermantes '''

7   controleur = 0
8   for parenthese in expression: #pour chaque parenthèse
9      if parenthese == '(':
10        controleur = controleur + 1
11     else:# parenthese == ')'
12        controleur = controleur - 1
13        if controleur ...  : # test 1 (à recopier et compléter)
14           #parenthèse fermante sans parenthèse ouvrante
15           return False
16   if controleur ... : # test 2  (à recopier et compléter)
17      return True #le parenthésage est correct
18   else:
19      return False #parenthèse(s) fermante(s) manquante(s)
```

Partie B : Texte correctement balisé

On peut faire l'analogie entre le texte simplifié des fichiers HTML (uniquement constitué de balises ouvrantes <nom> et fermantes </nom>) et les expressions parenthésées :

Par exemple, l'expression HTML simplifiée :
" " est correctement balisée.

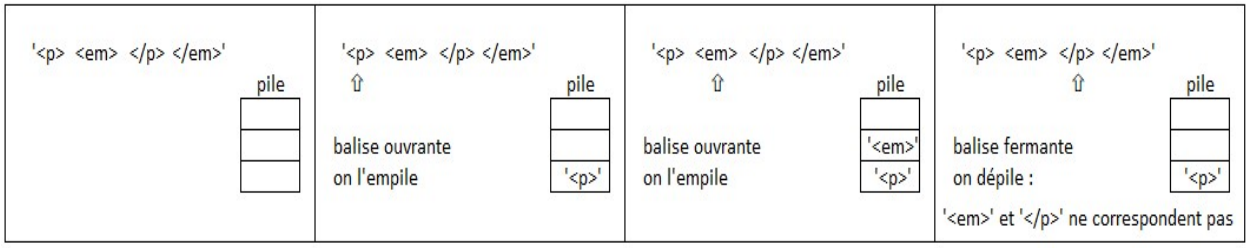
On ne tiendra pas compte dans cette partie des balises ne comportant pas de fermeture comme <br> ou <img …>.

Afin de vérifier qu'une expression HTML simplifiée est correctement balisée, on peut utiliser une pile (initialement vide) selon l'algorithme suivant :

On parcourt successivement chaque balise de l'expression :

- lorsque l'on rencontre une balise ouvrante, on l'empile ;
- lorsque l'on rencontre une balise fermante :
  - si la pile est vide, alors l'analyse s'arrête : le balisage est incorrect ,
  - sinon, on dépile et on vérifie que les deux balises (la balise fermante rencontrée et la balise ouvrante dépilée) correspondent (c'est-à-dire ont le même nom) si ce n'est pas le cas, l'analyse s'arrête (balisage incorrect).

Exemple : État de la pile lors du déroulement de cet algorithme pour l'expression simplifiée `"<p><em></p></em>"` qui n'est pas correctement balisée.



État de la pile lors du déroulement de l'algorithme

4. Cette question traite de l'état de la pile lors du déroulement de l'algorithme.

a. Représenter la pile à chaque étape du déroulement de cet algorithme pour l'expression `"<p><em></em></p>"` (balisage correct).

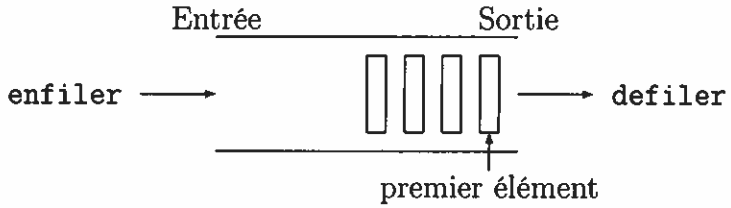
b. Indiquer quelle condition simple (sur le contenu de la pile) permet alors de dire que le balisage est correct lorsque toute l'expression HTML simplifiée a été entièrement parcourue, sans que l'analyse ne s'arrête.

5. Une expression HTML correctement balisée contient 12 balises.
Indiquer le nombre d'éléments que pourrait contenir au maximum la pile lors de son analyse.

Exercice 5 (4 points).

Cet exercice porte sur les files, les tableaux et les algorithmes associés.

L'objectif de cet exercice est de travailler sur les températures relevées par une station météorologique. Les données sont enregistrées une fois par jour, à la même heure, et traitées dans l'ordre dans lequel elles arrivent.
On choisit d'utiliser une file : une file est une structure de données abstraite fondée sur le principe « premier arrivé, premier sorti ».



On munit la structure de données `File` des quatre fonctions primitives définies ci-dessous :

Structure de données abstraite : File

Utilise : Élément, Booléen

Opérations :

- `creer_file_vide` : $\emptyset \longrightarrow$ `File`
  `creer_file_vide()` renvoie une file vide
- `est_vide` : `File` $\longrightarrow$ `Booléen`
  `est_vide(F)` renvoie `True` si la file `F` est vide, `False` sinon
- `enfiler` : `File, Élément` $\longrightarrow \emptyset$
  `enfiler(F, element)` ajoute `element` en entrée de la file `F`
- `defiler` : `File` $\longrightarrow$ `Élément`
  `defiler(F)` renvoie l'élément en sortie de la file `F` (premier élément) en le retirant de la file `F`

1. Les températures relevées ont été 15, puis 17, puis 14.

   (a) Parmi les quatre propositions suivantes, indiquer celle qui représente correctement cette file :

   Proposition 1 : Entrée 15 17 14 Sortie — Le premier élément est 14

   Proposition 2 : Entrée 14 17 15 Sortie — Le premier élément est 15

   Proposition 3 : Entrée 15 17 14 Sortie — Le premier élément est 15

   Proposition 4 : Entrée 14 17 15 Sortie — Le premier élément est 14

   (b) En utilisant les fonctions primitives précédentes, donner les instructions permettant de créer cette file.

2. On appelle longueur d'une file le nombre d'éléments qu'elle contient.

La fonction `longueur_file` prend en paramètre une file `F` et renvoie sa longueur `n`.

Après appel de cette fonction, la file `F` doit avoir retrouvé son état d'origine.

Exemple :

Si F = | 10 | 10 | 12 | 12 | alors `longueur_file(F)` vaut 4.

Recopier et compléter le programme Python suivant, implémentant la fonction `longueur_file`.

Dans le code de la fonction, les trois points (...) peuvent correspondre à une ou plusieurs lignes de programme.

```
def longueur_file(F):
    """File -> Int"""
    G = creer_file_vide() # file temporaire
    n = 0 # initialisation du nombre d'elements
    while not(est_vide(F)):
        ...
    while not(est_vide(G)): # reconstruction de la file initiale
        ...
    return ...
```

3. On s'intéresse à la variation de température d'un jour sur l'autre.

Par exemple, lorsque les températures relevées sont dans l'ordre d'arrivée 15, 17 et 14, les variations sont 2 et -3.

Recopier et compléter le programme Python implémentant la fonction `variations` qui prend en paramètre une file non vide `F` et qui renvoie le tableau `tab` contenant les variations successives, ou un tableau vide si la file `F` ne contient qu'une seule température. Il n'est pas demandé ici que la file `F` retrouve son état d'origine après appel de la fonction `variations`.

Exemple : si `F` est la file qui contient dans l'ordre des relevés les valeurs 15, 17 et 14, `variations(F)` vaut `[2, -3]`.

Dans le code de la fonction, les trois points (...) peuvent correspondre à une ou plusieurs lignes de programme.

```
def variations(F):
    """File -> Tableau"""
    taille = longueur_file(F)
    if taille == 1:
        ...
    else:
        tab = [0 for k in range(taille - 1)]
        element1 = defiler(F)
        for i in range (taille - 1):
            element2 = defiler(F)
            ...
    return ...
```

4. Écrire une fonction `nombre_baisses` qui prend en paramètre un tableau `tab`, non vide, des variations des températures et qui renvoie un p-uplet contenant le nombre de jours où la

température a baissé par rapport au jour précédent (soit le nombre de valeurs strictement négatives de `tab`), ainsi que la baisse journalière la plus importante (soit la valeur minimale de `tab`).

S'il n'y a aucune baisse (toutes les valeurs de `tab` sont positives), la fonction renvoie le p-uplet `(0,0)` .

Exemple 1 : `nombre_baisses([1, -4, 2, -1, 3])` vaut `(2,-4)`.

Exemple 2 : `nombre_baisses([1, 5, 3, 1])` vaut `(0,0)`.