FACULTÉ DES SCIENCES ET TECHNOLOGIES | Université de Lille

Algorithmes et Programmation

TD : SLD

1 Opérations primitives sur les listes

Exercice 1 *Construction de listes*

Pour chacune des listes qui suivent, indiquez comment la construire à partir de la liste vide en utilisant le constructeur de l'implémentation vue en cours.

1. $[3, 1, 4]$
2. $[[3, 1], [], [4, 1, 5]]$

2 Opérations courantes non primitives

Exercice 2 *Longueur d'une liste*

Réalisez une fonction `length` qui renvoie la longueur d'une liste.
Donnez-en une version récursive, et une version itérative.

Exercice 3 *n-ième élément*

Réalisez une fonction nommée `nth` qui renvoie l'élément d'indice n d'une liste ℓ si cet élément existe, et déclenche une exception `IndexError` sinon, l'entier n et la liste ℓ étant les deux paramètres de cette fonction.

Exercice 4 *Dernier élément*

Réalisez la fonction `last` qui renvoie le dernier élément d'une liste non vide et déclenche une exception si la liste est vide.

Exercice 5 *Concaténer deux listes*

Concaténer deux listes ℓ_1 et ℓ_2, c'est construire une liste contenant les éléments de ℓ_1 suivis de ceux de ℓ_2.

Réalisez la fonction `concat` qui renvoie la concaténation des deux listes passées en paramètre.

Exercice 6 *Renverser une liste*

Renverser une liste ℓ, c'est construire une liste qui contient les mêmes éléments que ℓ mais dans l'ordre inverse.

Question 1

Réalisez la fonction `reverse` qui renvoie la liste passée en paramètre renversée.

Réalisez une version récursive d'abord, puis récursive terminale.

Question 2 Donnez une version récursive terminale de la fonction `concat`.

CC BY NC SA

Exercice 7 *Appliquer une fonction à tous les éléments d'une liste*

Appliquer une fonction f à tous les éléments d'une liste ℓ, c'est construire une liste contenant les images $f(x)$ de tous les éléments x de ℓ.

Par exemple, appliquer la fonction carrée à la liste $[1, 2, 3]$ donne la liste $[1, 4, 9]$.

Réalisez une fonction nommée `map` qui applique une fonction aux éléments d'une liste.

Exercice 8 *Aplatir une liste de listes*

Aplatir une liste de listes, c'est construire une liste contenant tous les éléments des listes de la liste. Par exemple, la liste aplatie correspondant à la liste $[[3, 1, 4], [], [1, 5]]$ est la liste $[3, 1, 4, 1, 5]$.

Réalisez une fonction nommée `flatten` qui renvoie la liste aplatie.

Exercice 9 *Conversions listes/listes*

Réalisez une fonction qui convertit une liste PYTHON en une de nos listes, tout en conservant l'ordre des éléments dans la liste produite. Et réciproquement.

```
>>> l = native2list([3, 1, 4, 1, 5])
>>> list2native(l)
[3, 1, 4, 1, 5]
```

Exercice 10 *Zip de deux listes*

« *Zipper* » [1] deux listes ℓ_1 et ℓ_2 c'est construire une liste de même longueur que la longueur supposée commune de ℓ_1 et ℓ_2 dont les éléments sont des couples dont la première composante est dans ℓ_1 et la seconde dans ℓ_2.

Question 1 Réalisez une fonction nommée `zip` qui prend deux listes de même longueur en paramètre et les *zippe*.

En voici un exemple d'utilisation :

```
>>> l1 = native2list([1, 3, 5, 7])
>>> l2 = native2list(['Timoleon', 'Calbuth', 'Talon', 'Carmen'])
>>> l = zip(l1, l2)
>>> list2native(l)
[(1, 'Timoleon'), (3, 'Calbuth'), (5, 'Talon'), (7, 'Carmen')]
```

Question 2 Réalisez la fonction réciproque que vous nommerez `unzip`.

```
>>> l1,l2 = unzip(l)
>>> list2native(l1)
[1, 3, 5, 7]
>>> list2native(l2)
['Timoleon', 'Calbuth', 'Talon', 'Carmen']
```

Exercice 11 *Opérations primitives sur les piles*

Question 1 Donnez les états successifs de la pile dans la séquence d'instructions suivante.

```
st = ApStack()
st.push(1)
st.push(7)
```

1. Rien à voir avec la compression des données, mais plutôt avec les fermetures éclairs.

```
st.push(5)
print("{:d}".format(st.pop()))
print("{:d}".format(st.pop()))
print("{:d}".format(st.pop()))
st.is_empty()
```

Question 2 Que fait la séquence d'instructions suivante ? Au besoin la modifier.

```
st = ApStack()
st.push(1)
st.push(7)
st.push(5)
while not st.is_empty():
   print("{:d}".format (st.top()))
```

Question 3 Indiquez ce que fait la fonction suivante.

```
def what_do_i_do(st):
   n = 0
   while not st.is_empty():
      n = n + st.pop()
   st.push(n)
```

Exercice 12 *Expressions postfixées*

Question 1 Écrivez sous forme postfixée les expressions arithmétiques

1. $(a+b)\times c$;
2. $\frac{a}{b}\times c - \frac{c}{d\times e} - (f-g)$.

Question 2 Indiquez les états successifs de la pile lors de l'évaluation de ces deux expressions.

Exercice 13 *Inversion de l'ordre des éléments d'une liste*

On suppose donné une liste ℓ. Comment à l'aide d'une pile, inverser l'ordre des éléments de ℓ ?

Exercice 14 *Impression des éléments d'une pile*

On veut réaliser une procédure `stack_print` qui imprime (sur la sortie standard) tous les éléments d'une pile d'entiers. Une fois son travail d'impression réalisé, cette procédure de type ne doit pas avoir modifié la pile.

On va étudier deux affichages différents qui seront illustrés en les appliquant sur la pile définie par :

```
>>> st = ApStack()
>>> for i in range(1, 5):
...     st.push(i)
...
```

Question 1 Dans un premier temps on demande un affichage vertical, un entier par ligne, dans l'ordre du sommet vers le bas de pile. Par exemple

```
>>> stack_print1(st)
4
```

```
3
2
1
>>> st.top()
4
```

le sommet de la pile `st` est l'entier 4.

Question 2 Dans un deuxième temps on demande un affichage horizontal, le bas de pile étant à gauche et le sommet à droite. La même pile que précédemment est imprimée comme on le voit ci-dessous.

```
>>> stack_print2(st)
[1 2 3 4
```

Exercice 15 *Textes bien parenthésés*

Dans de très nombreux contextes (compilation, pages HTML, XML, ...), il est nécessaire de vérifier qu'un texte est *bien parenthésé*. Le bon parenthésage d'un texte signifie que

— toute parenthèse ouverte est fermée ;
— à toute parenthèse fermante correspond une parenthèse ouvrante ;
— et qu'à aucune position dans le texte le nombre de parenthèses fermantes n'est strictement supérieur à celui des parenthèses fermantes rencontrées auparavant.

À titre d'exemples, voici trois textes bien parenthésés (à gauche) et trois mal parenthésés (à droite).

`"(((()))))"`	`"(()(()))"`
`"()()()"`	`"(())())"`
`""`	`"(()))(()"`

Question 1 Vérifiez ces exemples.

Dans toute la suite on suppose que le texte est représenté par une liste de chaînes de caractères, certaines d'entre-elles étant des parenthèses ouvrantes ou fermantes.

Question 2 Dans cette question on suppose que les seules parenthèses sont les parenthèses `(` et `)`.

Réalisez un prédicat qui prend un texte en paramètre et renvoie :
— `True` si le texte est bien parenthésé ;
— et `False` dans le cas contraire.

Question 3 On suppose dans cette question que le texte peut contenir trois types de parenthèses : `()`, `[]` et `{}`.

Reprogrammez le prédicat précédent.

Question 4 Enfin dans cette dernière question, les parenthèses peuvent être n'importe quelles balises HTML, ou plus généralement XML, comme par exemple `<html> </html>`, `<body> </body>`, `<a> </a>`, `<author> </author>`, ... à l'exclusion des balises autofermantes comme `<img ... />`.

Reprogrammez le prédicat qui contrôle le bon parenthésage.

Exercice 16 *Conversion en binaire*

En utilisant une pile ne contenant que des 0 et des 1, programmez une fonction de conversion d'un nombre en binaire.

```
>>> to_bin(12)
'1100'
>>> to_bin(0)
'0'
```

Exercice 17 *Fusion de deux piles*

Question 1 Réalisez une fonction nommée `stack_merge` paramétrée par deux piles triées dans l'ordre croissant (du sommet vers le fond) et qui produit une liste triée dans l'ordre croissant des éléments contenus dans ces deux piles.

```
>>> st1 = ApStack()
>>> st1.push(11)
>>> st1.push(5)
>>> st1.push(1)
>>> st2 = ApStack()
>>> st2.push(9)
>>> st2.push(7)
>>> stack_merge(st1, st2)
[1, 5, 7, 9, 11]
```

Question 2 Votre fonction a–t–elle un effet de bord ?

Question 3 Reprendre la première question en supposant les deux piles triées dans l'ordre croissant du fond vers le sommet. La liste à construire doit toujours être triée dans l'ordre croissant.
