Devoir Surveillé 1

6 Novembre 2020 8h30-10h30

Les piles

1. On définit la structure type d'une pile à l'aide de l'acronyme LIFO (Last In, First Out). Expliquer quelle propriété d'une pile est précisée par cet acronyme.
2. Donner un exemple d'utilisation d'une pile en informatique.
3. Quels sont les avantages et inconvénients de l'utilisation d'une pile en informatique ?
4. On souhaite implémenter une pile non bornée en python, quelle structure de donnée peut-on utiliser ?
5. On donne les deux fonctions `empiler` et `depiler` suivantes :

```
1   def empiler(p, v):
2     p.append(v)
3
4   def depiler(p):
5     assert len(p) > 0
6     return p.pop()
7
8
```

   Expliquer ce que font ces deux fonctions : c'est-à-dire quels sont leurs arguments, quelles sont les modifications apportées, quelles sont les sorties ?

6. On dispose uniquement des fonctions `empiler(p,v)`, `depiler(p)` définies précédemment et aussi des fonctions :

```
1
2   creer_pile() # cette fonction n'a pas d'argument et retourne en sortie une
      pile vide non bornee
3
4   est_vide(p) # cette fonction prends en argument une pile non bornee p et
      retourne en sortie la variable booleenne True si la pile est vide et
      False sinon.
5
```

   Écrire à l'aide des fonctions ci-dessus une fonction `superposer(p1,p2)` qui prends en argument deux piles `p1` et `p2`, et modifie la pile `p1` de manière à superposer la pile `p2` par-dessus.

   Par exemple si la pile `p1` contient les éléments 'chat' et [1, 2, 3] et la pile `p2` contient les éléments 3 et 1 alors les lignes de commande ci-dessous

```
1
2     superposer(p1,p2)
3
4     print(p1)
5
6
```

donnent comme résultat

```
1     => ['chat', [1, 2, 3], 3, 1]
2
```

7. Soit `s` une liste contenant une chaîne de caractère ne contenant que des caractères '(' et ')'. La fonction `parentheses(s)` ci-dessous détermine si le mot est bien parenthèsé ou pas et indique, pour chaque parenthèse ouvrante, la position de la parenthèse fermante correspondante.

```
1     def parentheses(s):
2       p = creer_pile() # on cree une pile p
3       for i in range(len(s)): # on parcourt tous les caracteres du 'mot' de la
        gauche vers la droite
4         if s[i] == '(': # si on rencontre une parenthese ouvrante, on vient d'
        ouvrir une parenthese
5           empiler(p, i) # alors on note dans la liste l'indice de la parenthese
         ouvrante
6         else: # sinon c'est que c'est une parenthese fermante
7           if est_vide(p): # si la pile est vide ca veut dire qu'on n'a pas
        ouvert de parenthese
8             return False # donc le mot est mal parenthese
9           j = depiler(p) # si la pile n'est pas vide, on recupere l'indice de
        la derniere parenthese ouverte
10          print((j, i)) # et on affiche les indices des parenthese ouvrante/
        fermante correspondante
11      return est_vide(p) # une fois le mot parcouru, on verifie qu'il ne reste
        pas de parenthese ouverte
12
```

Quel est le résultat de `: parentheses('()(()())')` ?

8. Modifier la fonction `parentheses(s)` pour une fonction `parentheses+acolades(s)` qui pour une chaîne de caractère `s` ne contenant que des caractères '(', ')', '{', et '}' détermine si le mot est bien parenthèsé ou pas et si les accolades sont cohérentes ou pas. Par contre on n'indiquera plus la position des parenthèses ouvrantes et fermantes correspondantes.

La Récursivité

9. Définir une fonction récursive.
10. Les deux fonctions ci-dessous calculent la factorielle d'un nombre entier.

```
1     def factorielle1(n):
2       resultat = 1
3       for i in range(n):
4         resultat = (i+1)*resultat
5       return resultat
6
```

```
7     def factorielle2(n):
8       if n==1:
9         return 1
10      else :
11        resultat = n*factorielle2(n    1)
12
13      return resultat
14
15
16
```

Justifier laquelle des deux fonctions est récursive.

11. Pour quelles valeurs de n la fonction récursive calculant la factorielle définie à la question précédente ne fonctionne pas ? Vous justifierez votre réponse.
12. Écrire une fonction récursive ayant en argument un indice n et donnant en résultat le terme u_n de la suite définie par

$$u_{n+2} = \sqrt{u_{n+1} + u_n}$$

avec $u_0 = 0$ et $u_1 = 1$

13. De même écrire une fonction récursive ayant en argument un indice n et donnant en résultat le terme v_n de la suite définie par

$$v_{n+2} - \sqrt{2v_{n+3} + v_n} = 0$$

avec $v_0 = 0$, $v_1 = 1$, et $v_2 = 1$,

14. Soit deux fonctions telles que :

$$f(n) = g(n-1)$$

avec $f(0) = 0$
et

$$g(n) = f(n-1)$$

avec $g(0) = 1$
Écrire un programme python qui implémente ces deux fonctions, calculer f(1), f(2), f(3), f(4), puis donner l'intérêt de cette fonction.

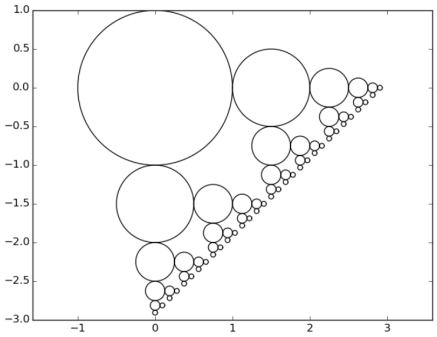
15. Calculer la complexité temporelle de la fonction que vous avez écrite à la question 12.
16. Écrire une fonction itérative qui réalise le même but que la fonction de la question 12. Puis calculer sa complexité temporelle.
17. A l'aide des deux questions précédentes, discuter des avantages et inconvénients des programmes récursifs par rapport aux programmes itératifs.
18. Soit la fonction `cercle(x,y,r)` suivante :

```
1   import matplotlib.pyplot as plt
2   import numpy as np
3
4   def cercle(x,y,r):
5     theta = np.linspace(0, 2*np.pi, 100)
6     X = r*np.cos(theta)+x
```

```
7       Y = r*np.sin(theta)+y
8       plt.plot(X,Y)
9
10
```

Que fait cette fonction ?

19. Écrire une fonction `bulles1(n)` telle que l'appel de `bulles1(5)` donne le graphe suivant



20. Écrire une fonction `bulles2(n)` telle que l'appel de `bulles2(5)` donne le graphe suivant