



Exercice 1. Récurtivité

9 points

Cet exercice porte sur les langages et la programmation (récurtivité).

1. Voici une fonction codée en Python :

```
def f(n) :  
    if n == 0 :  
        print("Partez!")  
    else :  
        print(n)  
        f(n-1)
```

a. Qu'affiche la commande `f(5)` ?



Corrigé

La commande `f(5)` va afficher une séquence décroissante de nombres, en commençant par 5 et en décrémentant jusqu'à 1, suivie du message "Partez!". Plus précisément, elle affichera :

```
5  
4  
3  
2  
1  
Partez!
```

b. Pourquoi dit-on de cette fonction qu'elle est récurtivité?



Corrigé

La fonction `f` est récurtivité car elle s'appelle elle-même à l'intérieur de son propre corps. Dans ce cas, lorsque `n` est différent de zéro, `f` est appelée avec `n-1` comme argument, jusqu'à ce que `n` atteigne la valeur zéro, moment où la récurtivité s'arrête (c'est le cas de base) et affiche "Partez!".

2. On rappelle qu'en Python, l'opérateur `+` a le comportement suivant sur les chaînes de caractères :

```
>>> S = 'a'+ 'bc'  
>>> S  
'abc'
```

Et le comportement suivant sur les listes :

```
>>> L = ['a'] + ['b', 'c']  
>>> L  
'a', 'b', 'c'
```

On a besoin pour les questions suivantes de pouvoir ajouter une chaîne de caractères s en préfixe à chaque chaîne de caractères de la liste $liste$. On appellera cette fonction `ajouter`.

Par exemple, `ajouter("a", ["b", "c"])` doit retourner `["ab", "ac"]`.

a. Recopiez le code suivant et complétez sur votre copie :

```
def ajouter(s, liste):
    res = []
    for m in liste:
        res .....
    return res
```



Corrigé

| Le code complété est :

```
1 def ajouter(s, liste):
2     res = []
3     for m in liste:
4         res.append(s + m)
5     return res
```

b. Que renvoie la commande `ajouter("b", ["a", "b", "c"])` ?



Corrigé

| La commande `ajouter("b", ["a", "b", "c"])` renvoie `["ba", "bb", "bc"]`.

c. Que renvoie la commande `ajouter("a", [""])` ?



Corrigé

| La commande `ajouter("a", [""])` renvoie `["a"]`.

3. On s'intéresse ici à la fonction suivante écrite en Python où s est une chaîne de caractères et n un entier naturel.

```
def produit(s, n):
    if n == 0:
        return [""]
    else:
        res = []
        for i in range(len(s)):
            res = res + ajouter(s[i], produit(s, n-1))
        return res
```

a. Que renvoie la commande `produit("ab", 0)` ? Le résultat est-il une liste vide ?

**Corrigé**

La commande `produit("ab", 0)` renvoie `[""]`. Ce n'est pas une liste vide; c'est une liste contenant une chaîne de caractères vide.

b. Que renvoie la commande `produit("ab", 1)`?

**Corrigé**

La commande `produit("ab", 1)` renvoie `["a", "b"]`, car elle ajoute chaque caractère de "ab" comme chaîne distincte dans la liste.

c. Que renvoie la commande `produit("ab", 2)`?

**Corrigé**

La commande `produit("ab", 2)` renvoie `["aa", "ab", "ba", "bb"]`, correspondant à toutes les combinaisons de deux caractères formées à partir de "ab".

Exercice 2. Structures de données : Piles

10 points

Cet exercice porte sur les structures de données.

La poussette est un jeu de cartes en solitaire. Cet exercice propose une version simplifiée de ce jeu basée sur des nombres.

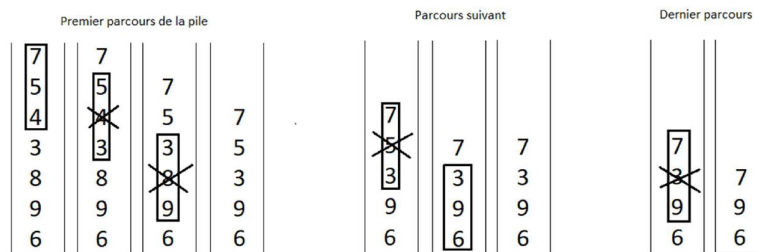
On considère une pile constituée de nombres entiers tirés aléatoirement. Le jeu consiste à réduire la pile suivant la règle suivante : quand la pile contient du haut vers le bas un triplet dont les termes du haut et du bas sont de même parité, on supprime l'élément central.

Par exemple :

- Si la pile P contenant le triplet 1 0 3, on supprime le 0.
- Pour la pile contenant le triplet 1 0 8, la pile reste inchangée.

On parcourt la pile ainsi de haut en bas et on procède aux réductions. Arrivé en bas de la pile, on recommence la réduction en repartant du sommet de la pile jusqu'à ce que la pile ne soit plus réductible. Une partie est « gagnante » lorsque la pile finale est réduite à deux éléments exactement.

Voici un exemple détaillé de déroulement d'une partie.

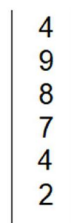


1. Donner la signification des termes « LIFO » et « FIFO ».

Lequel de ces deux termes correspond aux piles ?

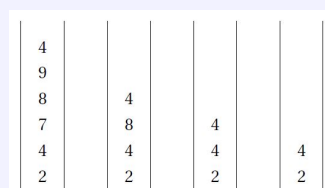
2.

(a) Donner les étapes de réduction de la pile suivante :

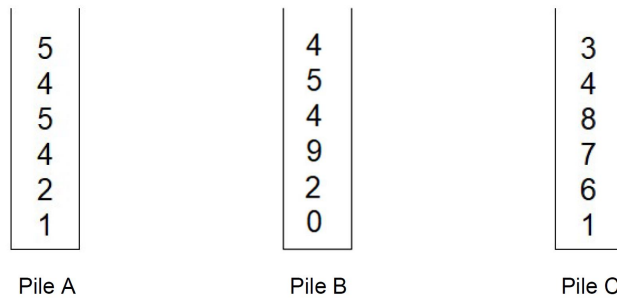


Corrigé

Voici les étapes de réduction :

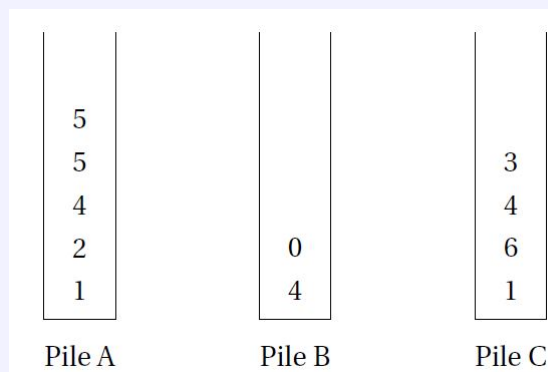


(b) Parmi les piles suivantes, donner celle qui est gagnante (justifiez).



Corrigé

Après réduction, on obtient les 3 piles suivantes :



La pile gagnante est la pile B, car elle est réduite à deux éléments.

L'interface d'une pile est proposée ci-dessous. On utilisera uniquement les fonctions figurant dans le tableau suivant :

Structure de données abstraite : Pile

- `creer_pile_vide()` renvoie une pile vide
- `est_vide(p)` : renvoie `True` si `p` est vide, `False` sinon
- `empiler(p, element)` : ajoute `element` au sommet de `p`
- `depiler(p)` : retire l'élément au sommet de `p` et le renvoie
- `sommet(p)` : renvoie l'élément au sommet de `p` sans le retirer de `p`
- `taille(p)` : renvoie le nombre d'éléments de `p`

3. La fonction `reduire_triplet_au_sommet` permet de supprimer l'élément central des trois premiers éléments (du haut vers le bas) si le haut et le bas sont de même parité. Les éléments dépilés non supprimés sont remplacés dans le bon ordre dans la pile.

Recopier et compléter sur la copie le code de la fonction `reduire_triplet_au_sommet` prenant une pile `p` en paramètre et qui la modifie en place. Cette fonction ne renvoie donc rien.

```

1 def reduire_triplet_au_sommet (p) :
2     a = depiler(p)
3     b = depiler(p)
4     c = sommet (p)
5     if a % 2 != c % 2:
6         empiler(p, b)
7     empiler(p, a)

```

**Corrigé**

Le code de la fonction `reduire_triplet_au_sommet` est complété pour supprimer l'élément central si les extrémités sont de même parité.

4.

(a) Donner la taille minimale que doit avoir une pile pour être réductible.

**Corrigé**

La taille minimale pour une pile réductible est de 3, car il faut au moins trois éléments pour former un triplet.

(b) Compléter la fonction `parcourir_pile_en_reduisant` qui parcourt la pile du haut vers le bas pour chaque triplet :

```

1  def parcourir_pile_en_reduisant (p) :
2      q = creer_pile_vider()
3      while taille(p) >= 3:
4          reduire_triplet_au_sommet (p)
5          e = depiler(p)
6          empiler(q, e)
7      while not est_vide(q) :
8          e = depiler(q)
9          empiler(p, e)
10     return p

```

**Corrigé**

Le code de la fonction `parcourir_pile_en_reduisant` est complété pour parcourir la pile et appliquer les réductions nécessaires.

5. Partant d'une pile d'entiers `p`, on propose ici d'implémenter une fonction récursive `jouer` renvoyant la pile `p` entièrement simplifiée. Une fois la pile parcourue de haut en bas et réduite, on procède à nouveau à sa réduction à condition que cela soit possible. Ainsi :

- Si la pile `p` n'a pas subi de réduction, on la renvoie.
- Sinon on appelle à nouveau la fonction `jouer`, prenant en paramètre la pile réduite.

Recopier et compléter sur la copie le code ci-dessous :

```

def jouer (p) :
    taille_p = taille(p)
    q = parcourir_pile_en_reduisant (p)
    if taille(q) == taille_p: # Pas de réduction
        return p
    else:
        return jouer (q)

```

**Corrigé**

La fonction `jouer` est complétée pour appeler récursivement `parcourir_pile_en_reduisant` jusqu'à stabilisation.

Exercice 3. Structures de données : Files**3 points**

Cet exercice porte sur les structures de données et les Files.

Nous avons créé une classe `File` pour implémenter la structure de file.

```
class File:
    def __init__(self):
        """Initialisation d'une file vide"""

    def vide(self):
        """teste si la file est vide"""

    def defiler(self):
        """Defile une file et renvoie le sommet"""

    def enfiler(self, x):
        """enfile l'element x dans la file """
```

Réaliser la méthode `taille` qui retourne la taille de la file.

- Vous ne pouvez utiliser que les méthodes de la classe `File`.
- La file doit être à son état d'origine à la fin des processus.

```

1  def taille(self):
2      """retourne la taille d'une file"""
3      taille = 0
4      q=File()
5      while not self.vide():
6          x=self.defiler() # enlève et renvoie le premier élément de la file
7          q.enfiler(x) #ajoute x à la file f
8          taille += 1
9      while not q.vide():
10         x=q.defiler() # enlève et renvoie le premier élément de la file
11         self.enfiler(x) # ajoute x à la file f
12     return taille
13
14
15  def sommet(self):
16      """renvoie le sommet d'une file p"""
17      assert not self.vide(), "file vide"
18      q=File()
19      sommet=self.defiler() # enlève et renvoie le premier élément de la file
20      q.enfiler(sommet) #ajoute x à la file f
21      while not self.vide():
22          x=self.defiler() # enlève et renvoie le premier élément de la file
23          q.enfiler(x) #ajoute x à la file f
24      while not q.vide():
25          x=q.defiler() # enlève et renvoie le premier élément de la file
26          self.enfiler(x) # ajoute x à la file f
27      return sommet

```

↩️ Fin du devoir 🏹



Question Bonus

🔗 Ecrire la méthode sommet qui renvoie le sommet de la file (sans la modifier).



Epreuve pratique

| Faire l'exercice suivant : <https://capitale2.ac-paris.fr/web/c/1ffd-7775513>