# BNS 2022 : passage en revue des sujets

## 1. Sujet 1

### Exercice 1.1

L'ÉNONCÉ

Écrire une fonction `recherche` qui prend en paramètres `caractere`, un caractère, et `mot`, une chaîne de caractères, et qui renvoie le nombre d'occurrences de `caractere` dans `mot`, c'est-à-dire le nombre de fois où `caractere` apparaît dans `mot`.

Exemples :

**Script Python**

```
>>> recherche('e', "sciences")
2
>>> recherche('i',"mississippi")
4
>>> recherche('a',"mississippi")
0
```

COMMENTAIRES

Pas grand chose mis à part cette distinction caractère/chaîne de caractères qui n'existe pas en Python. La personne qui a rédigé est sûrement une habituée d'Ocaml en prépa :) Il y a aussi cette espace après la virgule des appels de `recherche` plus ou moins existante... Enfin le choix du nom `recherche` est discutable car cela ferait plus penser à un test renvoyant un booléen (je recherche `i` dans `mississippi`). On aurait pu choisr `compte_lettres`, `nb_occurrences`, `compte_occurrences`, etc.

### Exercice 1.2

L'ÉNONCÉ

On s'intéresse à un algorithme récursif qui permet de rendre la monnaie à partir d'une liste donnée de valeurs de pièces et de billets - le système monétaire est donné sous forme d'une liste `pieces=[100, 50, 20, 10, 5, 2, 1]` - (on supposera qu'il n'y a pas de limitation quant à leur nombre), on cherche à donner la liste de pièces à rendre pour une somme donnée en argument. Compléter le code Python ci-dessous de la fonction `rendu_glouton` qui implémente cet algorithme et renvoie la liste des pièces à rendre.

Script Python

```
1  Pieces = [100,50,20,10,5,2,1]
2
3  def rendu_glouton(arendre, solution=[], i=0):
4          if arendre == 0:
5          return ...
6      p = pieces[i]
7      if p <= ... :
```

```
        solution.append(...)
        return rendu_glouton(arendre - p, solution,i)
    else :
        return rendu_glouton(arendre, solution, ...)
```

On devra obtenir :

**Script Python**

```
>>>rendu_glouton_r(68,[],0)
[50, 10, 5, 2, 1]
>>>rendu_glouton_r(291,[],0)
[100, 100, 50, 20, 20, 1]
2/3
```

COMMENTAIRES

Une erreur grossière sur 4 de la version 2021 de cet exercice a disparu mais il en reste 3...

On fermerait presque les yeux sur l'erreur d'indentation de la première ligne du bloc de la fonction, de l'oubli du `_r` en suffixe du nom de la fonction dans le code de celle-ci, du `P` majuscule de `pieces` ...Mais le pire est à venir : la paramètre par défaut mutable. La personne qui a rédigé semble s'en être rendu compte puisqu'elle remet l'argument par défaut dans l'appel de la fonction mais alors on se demande vraiment pourquoi il y a eu un paramètre par défaut. Le raisonnement a sûrement été : " oh ça marche pô ! Je vais remettre le paramètre par défaut dans l'appel pour voir...ah tiens ça marche".

**Pourquoi un paramètre par défaut mutable, c'est le mal ?**

Si on utilise le code proposé dans le sujet 1 (en rétablissant l'indentation, en corrigeant les coquilles) :

**Script Python**

```
pieces = [100, 50, 20, 10, 5, 2, 1]

def rendu_glouton_r(arendre, solution=[], i = 0):
    if arendre == 0:
        return solution
    p = pieces[i]
    if p <= arendre :
        solution.append(p)
        return rendu_glouton_r(arendre - p, solution, i)
    else :
        return rendu_glouton_r(arendre, solution, i + 1)
```

On obtient :

**Script Python**

```
In [2]: rendu_glouton_r(68)
Out[4]: [50, 10, 5, 2, 1]

In [5]: rendu_glouton_r(68)
Out[5]: [50, 10, 5, 2, 1, 50, 10, 5, 2, 1]

In [6]: rendu_glouton_r(68)
Out[6]: [50, 10, 5, 2, 1, 50, 10, 5, 2, 1, 50, 10, 5, 2, 1]
```

L'idée de la personne qui a proposé ce code est sûrement de retrouver l'idée de récursion terminale chère aux utilisateurs de Ocaml en prépa. Mais ici, on travaille en Python...Le code de la fonction est créé une bonne fois pour toute à la création de la fonction i.e. le paramètre par défaut est lié une bonne fois pour toute à `solution`. Quand on appelle la fonction, on l'appelle toujours avec le même paramètre par défaut. Le problème est que `solution` est ici *mutable* et va donc évoluer à chaque exécution de la fonction. Une erreur classique mais délicate à comprendre pour les élèves et qu'on attend sûrement pas à trouver dans un sujet.

**Comment faire ?**

On pouvait dire… Oh ! Dieu !… bien des choses en somme… La 1ère idée est de prendre un paramètre par défaut non mutable comme un `tuple` mais on ne pourra pas le faire évoluer car il est par essence non mutable. On peut prendre `None` : il n'y a pas de solution au départ. Le problème est que ce n'est pas ce qu'on a envie d'avoir. Mais on peut bidouiller ainsi :

**Script Python**

```
pieces = [100, 50, 20, 10, 5, 2, 1]

def rendu_glouton_r(arendre, solution = None, i = 0):
    if solution is None:
        solution = []
    if arendre == 0:
        return solution
    p = pieces[i]
    if p <= arendre :
        solution.append(p)
        return rendu_glouton_r(arendre - p, solution, i)
    else :
        return rendu_glouton_r(arendre, solution, i + 1)
```

Ou on peut simuler de la récursion terminale (inutile en python) avec une fonction auxiliaire.

Ou on ne cherche pas à rendre la récursion terminale car c'est totalement inutile en Python :

**Script Python**

```
def rendu_glouton_r(arendre, i = 0):
    if arendre == 0:
        return []
    p = pieces[i]
    if p <= arendre :
        return rendu_glouton_r(arendre - p, i) + [p]
    else :
        return rendu_glouton_r(arendre, i + 1)
```

Petit bonus : la visualisation avec Pythontutor

# 2. Sujet 2

## 2.1. Exercice 2.1

**Énoncé**

Soit le couple (`note`, `coefficient`):

- `note` est un nombre de type flottant (`float`) compris entre 0 et 20 ;
- `coefficient` est un nombre entier positif.

Les résultats aux évaluations d'un élève sont regroupés dans une liste composée de couples (`note`, `coefficient`).

Écrire une fonction moyenne qui renvoie la moyenne pondérée de cette liste donnée en paramètre.

Par exemple, l'expression `moyenne([(15,2),(9,1),(12,3)])` devra renvoyer le résultat du calcul suivant :

$$\frac{2\times 15+1\times 9+3\times 12}{2+1+3}=12,5$$

**Commentaires**

Pas grand chose à dire, à part que c'est plus facile à écrire qu'un tri ou un algo glouton comme on en demande dans d'autres sujets. Attend-on une gestion de la liste vide, des coefficients négatifs ?

## 2.2. Exercice 2.2

**Énoncé**

On cherche à déterminer les valeurs du triangle de Pascal. Dans ce tableau de forme triangulaire, chaque ligne commence et se termine par le nombre 1. Par ailleurs, la valeur qui occupe une case située à l'intérieur du tableau s'obtient en ajoutant les valeurs des deux cases situées juste au-dessus, comme l'indique la figure suivante :

Compléter la fonction `pascal` ci-après. Elle doit renvoyer une liste correspondant au triangle de Pascal de la ligne `1` à la ligne `n` où `n` est un nombre entier supérieur ou égal à `2` (le tableau sera contenu dans la variable `C`). La variable `Ck` doit, quant à elle, contenir, à l'étape numéro `k`, la `k`-ième ligne du tableau.

Script Python

```
def pascal(n):
    C= [[1]]
    for k in range(1,...):
        Ck = [...]
        for i in range(1,k):
            Ck.append(C[...][i-1]+C[...][...] )
        Ck.append(...)
        C.append(Ck)
    return C
```

Pour `n = 4`, voici ce qu'on devra obtenir :

**Bash Session**

```
>>> pascal(4)
[[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1]]
```

Pour `n = 5`, voici ce qu'on devra obtenir :

**Bash Session**

```
>>> pascal(5)
[[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1], [1, 5, 10, 10, 5, 1]]
```

**Commentaires**

Un sujet un peu mathématique (après la moyenne en 1) d'un professeur français né avant 2000 qui pense encore aux coefficients binomiaux écrits sous la forme $C_n^k$ :) Les élèves ayant vu la formule de Pascal en maths seront avantagés mais...ah...on me parle dans mon écouteur...cela a disparu du programme de spécialité terminale depuis la réforme...mais cela a subsisté en maths complémentaires...j'ai dû mal comprendre.

Encore du laisser-aller dans les espaces entre les opérateurs binaires.

Une liste présentée en "triangle rectangle" aurait peut-être été plus facile à visualiser et la gestion des bornes des `range` est à mener avec attention.

Le choix des indices est peut-être perturbant car on a plus l'habitude de `i` et `j` mais je pinaille.

Je suppose qu'on attend ça:

**Script Python**

```
def pascal(n):
    C = [[1]]
    for k in range(1, n + 1):
        Ck = [1]
        for i in range(1, k):
            Ck.append(C[k-1][i-1] + C[k-1][i])
        Ck.append(1)
        C.append(Ck)
    return C
```

Je suppose aussi que les coups des 1 en bordure vont se régler par essais successifs.

On est presque content qu'il n'y ait pas de grosse absurdité ici...

# 3. Sujet 3

## 3.1. Exercice 3.1

**Énoncé**

Le codage par différence (delta encoding en anglais) permet de compresser un tableau de données en indiquant pour chaque donnée, sa différence avec la précédente (plutôt que la donnée elle-même). On se retrouve alors avec un tableau de données assez petites nécessitant moins de place en mémoire. Cette méthode se révèle efficace lorsque les valeurs consécutives sont proches.

Programmer la fonction `delta` qui prend en paramètre un tableau non vide de nombres entiers et qui renvoie un tableau contenant les valeurs entières compressées à l'aide cette technique.

Exemples :

**Script Python**

```
>>> delta([1000, 800, 802, 1000, 1003])
[1000, -200, 2, 198, 3]
>>> delta([42])
42
```

### Commentaires

Allez, la coquille, la coquille, la coquille ! Oui, bien sûr, vous l'avez vue : `delta([42])` doit renvoyer `[42]` et non pas `42`, à moins que la personne qui a testé le sujet n'ait utilisé Deep Thought et recherchait la réponse à la question ultime sur la vie, l'univers et tout le reste.

Puisqu'on ne peut pas modifier les sujets, encore un à jeter à la poubelle en plus du 1 et du 2 pour ne pas favoriser les maths comps. On espère qu'on va bientôt en trouver un de publiable.

Sinon, il y a quand même des pièges...

On peut penser à un changement en place :

**Script Python**

```
def delta1(xs):
    for i in range(1, len(xs)):
        xs[i] = xs[i] - xs[i - 1]
    return xs
```

mais ça pose problème:

**Bash Session**

```
In [142]:  delta1([1000, 800, 802, 1000, 1003])
Out[142]: [1000, -200, 1002, -2, 1005]
```

Donc en bon habitué de la programmation fonctionnelle qui ne fait pas de mal à ses arguments, on préfèrera ça :

**Script Python**

```
def delta2(xs):
    ys = [xs[0]] # mais il y a le problème de la liste vide
    for i in range(1, len(xs)):
        ys.append(xs[i] - xs[i - 1])
    return ys
```

mais on aura un problème avec la liste vide.

On peut arranger comme ça :

**Script Python**

```
def delta3(xs):
    ys = xs[:] # il faut connaître les problèmes de copie de liste
    for i in range(1, len(xs)):
```

```
        ys[i] = xs[i] - xs[i - 1]
    return ys
```

mais il faut savoir faire une copie de liste car sinon avec ça :

**Script Python**

```
def delta4(xs):
    ys = xs
    for i in range(1, len(xs)):
        ys[i] = xs[i] - xs[i - 1]
    return ys
```

on a ce problème :

**Bash Session**

```
In [146]:  delta4([1000, 800, 802, 1000, 1003])
Out[146]: [1000, -200, 1002, -2, 1005]
```

On peut aussi penser à ça:

**Script Python**

```
def delta5(xs):
    return [xs[0]] + [xs[i] - xs[i - 1] for i in range(1, len(xs))]
```

Bref, il y a quand même des pièges.

## 3.2. Exercice 3.2

**Énoncé**

Une expression arithmétique ne comportant que les quatre opérations +, −,×,÷ peut être représentée sous forme d'arbre binaire. Les nœuds internes sont des opérateurs et les feuilles sont des nombres. Dans un tel arbre, la disposition des nœuds joue le rôle des parenthèses que nous connaissons bien.

En parcourant en profondeur infixe l'arbre binaire ci-dessus, on retrouve l'expression notée habituellement :

$$3 \times (8 + 7) - (2 + 1)$$

La classe `Noeud` ci-après permet d'implémenter une structure d'arbre binaire. Compléter la fonction récursive `expression_infixe` qui prend en paramètre un objet de la classe `Noeud` et qui renvoie l'expression arithmétique représentée par l'arbre binaire passé en paramètre, sous forme d'une chaîne de caractères contenant des parenthèses.

Résultat attendu avec l'arbre ci-dessus :

**Script Python**

```
>>> e = Noeud(Noeud(Noeud(None, 3, None), '*', Noeud(Noeud(None, 8, None),
'+', Noeud(None, 7, None))), '-', Noeud(Noeud(None, 2, None), '+',
Noeud(None, 1, None)))

>>> expression_infixe(e)
'((3*(8+7))-(2+1))'
```

**Script Python**

```
class Noeud:
    def __init__(self, g, v, d):
        self.gauche = g
        self.valeur = v
        self.droit = d

    def __str__(self):
        return str(self.valeur)

    def est_une_feuille(self):
        '''Renvoie True si et seulement si le noeud est une feuille'''
        return self.gauche is None and self.droit is None


def expression_infixe(e):
    s = ...
    if e.gauche is not None:
        s = s + expression_infixe(...)
    s = s + ...
    if ... is not None:
        s = s + ...
    if ...:
        return s

    return '('+ s +')'
```

**Commentaire**

Ouh la, c'est pas évident évident ici. J'ai répondu un peu au hasard pour équilibrer les écritures et ça a marché.

Je suppose qu'on attend ça :

**Script Python**

```
def expression_infixe(e):
    s = ''
    if e.gauche is not None:
        s = s + expression_infixe(e.gauche)
    s = s + str(e.valeur)
    if e.droit is not None:
        s = s + expression_infixe(e.droit)
    if e.est_une_feuille():
        return s
    return '('+ s +')'
```

Cette construction de plusieurs `if` successifs sans `else` me met très mal à l'aise.

Je ne pense pas non plus que les candidats comprennent grand chose à cette fonction et vont faire des essais jusqu'à ce que ça marche ou vont utiliser les aides de la personne qui les examine.

Un exercice à éviter de prendre dans sa liste.

# 4. Sujet 4

## 4.1. Exercice 4.1

**Énoncé**

Écrire une fonction `recherche` qui prend en paramètre un tableau de nombres entiers `tab`, et qui renvoie la liste (éventuellement vide) des couples d'entiers consécutifs successifs qu'il peut y avoir dans `tab`.

Exemples :

**Script Python**

```
>>> recherche([1, 4, 3, 5])
[]
>>> recherche([1, 4, 5, 3])
[(4, 5)]
>>> recherche([7, 1, 2, 5, 3, 4])
[(1, 2), (3, 4)]
>>> recherche([5, 1, 2, 3, 8, -5, -4, 7])
[(1, 2), (2, 3), (-5, -4)]
```

**Commentaires**

Tiens, encore une recherche...Sauf que c'est une toute autre recherche que dans le sujet 1. On peut donc l'appeler `recherche_consecutifs` peut-être.

Ensuite la liste en paramètre est désignée comme étant un tableau et la liste renvoyée est désignée comme étant une liste. Comme M. Preskovic, je suis dans toutes mes confuses avec un tel énoncé.

Je suppose qu'on attend quelque chose comme ça:

**Script Python**

```
def recherche(tab):
    couples = []
    for i in range(len(tab) - 1):
        if tab[i + 1] - tab[i] == 1:
            couples.append((tab[i], tab[i + 1]))
    return couples
```

## 4.2. Exercice 4.2

**Énoncé**

Soit une image binaire représentée dans un tableau à 2 dimensions. Les éléments `M[i][j]`, appelés pixels, sont égaux soit à `0` soit à `1`.

Une composante d'une image est un sous-ensemble de l'image constitué uniquement de `1` et de `0` qui sont côte à côte, soit horizontalement soit verticalement.

Par exemple, les composantes de

sont

On souhaite, à partir d'un pixel égal à `1` dans une image `M`, donner la valeur `val` à tous les pixels de la composante à laquelle appartient ce pixel.

La fonction `propager` prend pour paramètre une image `M`, deux entiers `i` et `j` et une valeur entière `val`. Elle met à la valeur `val` tous les pixels de la composante du pixel `M[i][j]` s'il vaut `1` et ne fait rien s'il vaut `0`.

Par exemple, `propager(M,2,1,3)` donne

Compléter le code récursif de la fonction `propager` donné ci-dessous :

Script Python

```
def propager(M, i, j, val):
    if M[i][j]== ...:
        return None

    M[i][j] = val

    # l'élément en haut fait partie de la composante
    if ((i-1) >= 0 and M[i-1][j] == ...):
        propager(M, i-1, j, val)

    # l'élément en bas fait partie de la composante
    if ((...) < len(M) and M[i+1][j] == 1):
        propager(M, ..., j, val)

    # l'élément à gauche fait partie de la composante
    if ((...) >= 0 and M[i][j-1] == 1):
        propager(M, i, ..., val)

    # l'élément à droite fait partie de la composante
    if ((...) < len(M) and M[i][j+1] == 1):
        propager(M, i, ..., val)
```

Exemple :

**Bash Session**

```
>>> M = [[0,0,1,0],[0,1,0,1],[1,1,1,0],[0,1,1,0]]
>>> propager(M,2,1,3)
>>> M
[[0, 0, 1, 0], [0, 3, 0, 1], [3, 3, 3, 0], [0, 3, 3, 0]]
```

**Commentaires**

Je vais être désagréable mais cet exercice n'a strictement aucun intérêt en l'état si ce n'est être capable de comprendre un énoncé pas clair.

Ensuite, on peut compléter les trous sans du tout chercher à comprendre l'algo. Je le sais puisque c'est comme ça que j'ai fait.

À éviter donc.

Ah, voici le jeu des devinettes à la Pif Gadget :

**Script Python**

```
def propager(M, i, j, val):
    if M[i][j]== 0:
        return None

    M[i][j] = val

    # l'élément en haut fait partie de la composante
    if ((i-1) >= 0 and M[i-1][j] == 1):
        propager(M, i-1, j, val)

    # l'élément en bas fait partie de la composante
    if ((i + 1) < len(M) and M[i+1][j] == 1):
        propager(M, i + 1, j, val)

    # l'élément à gauche fait partie de la composante
    if ((j - 1) >= 0 and M[i][j-1] == 1):
        propager(M, i, j - 1, val)

    # l'élément à droite fait partie de la composante
    if ((j + 1) < len(M) and M[i][j+1] == 1):
        propager(M, i, j + 1, val)
```

# 5. Sujet 5

## 5.1. Exercice 5.1

**Énoncé**

Écrire une fonction `rechercheMinMax` qui prend en paramètre un tableau de nombres non triés `tab`, et qui renvoie la plus petite et la plus grande valeur du tableau sous la forme d'un dictionnaire à deux clés 'min' et 'max'. Les tableaux seront représentés sous forme de liste Python.

Exemples :

**Script Python**

```
>>> tableau = [0, 1, 4, 2, -2, 9, 3, 1, 7, 1]
>>> resultat = rechercheMinMax(tableau)
>>> resultat
{'min': -2, 'max': 9}
>>> tableau = []
>>> resultat = rechercheMinMax(tableau)
```

```
>>> resultat
{'min': None, 'max': None}
```

**Commentaires**

Tiens, une fonction recherche...Celle-ci a un suffixe. Mais avec une écriture de Haskellien. On doit être patient. Un jour les personnes qui rédigent les sujets liront la PEP 8. On aurait aimé un `recherche_min_max`.

On ne parlera pas du choix de nom un peu perturbant pour qui a fait un peu d'algo.

On trouve encore ici le nom de tableau. Au moins, on ne parle pas de liste juste après.

Bon, sinon c'est du quatre en un: recherche d'un max **et** d'un min **et** manipulation d'un dictionnaire **et** faire attention à l'initialisation avec les `None`.

**Script Python**

```
def recherche_min_max(tab):
    if len(tab) == 0:
        return  {'min': None, 'max': None}
    dic = {'min': tab[0], 'max': tab[0]}
    for nombre in tab[1:]:
        if nombre < dic['min']:
            dic['min'] = nombre
        if nombre > dic['max']:
            dic['max'] = nombre
    return dic
```

## 5.2. Exercice 5.2

**Énoncé**

On dispose d'un programme permettant de créer un objet de type `PaquetDeCarte`, selon les éléments indiqués dans le code ci-dessous. Compléter ce code aux endroits indiqués par `#A compléter`, puis ajouter des assertions dans l'initialiseur de `Carte`, ainsi que dans la méthode `getCarteAt()`.

**Script Python**

```
class Carte:
    """Initialise Couleur (entre 1 à 4), et Valeur (entre 1 à 13)"""
    def __init__(self, c, v):
        self.Couleur = c
        self.Valeur = v

    """Renvoie le nom de la Carte As, 2, ... 10, Valet, Dame, Roi"""
    def getNom(self):
        if (self.Valeur > 1 and self.Valeur < 11):
            return str(self.Valeur)
        elif self.Valeur == 11:
            return "Valet"
        elif self.Valeur == 12:
            return "Dame"
        elif self.Valeur == 13:
            return "Roi"
        else:
```

```
            return "As"

    """Renvoie la couleur de la Carte (parmi pique, coeur, carreau, trefle"""
    def getCouleur(self):
        return ['pique', 'coeur', 'carreau', 'trefle'][self.Couleur - 1]

class PaquetDeCarte:
    def __init__(self):
        self.contenu = []

    """Remplit le paquet de cartes"""
    def remplir(self):
        #A compléter

    """Renvoie la Carte qui se trouve à la position donnée"""
    def getCarteAt(self, pos):
        #A compléter
```

**Commentaires**

Bon, vous allez encore dire que je ne suis jamais content.

- les *docstrings* sont mises n'importe où. Pas pro.
- des majuscules pour les noms d'attributs.
- les noms à la Haskell au lieu de les écrire à la Python
- on aurait pu utiliser un dictionnaire pour ces associations valeur: nom puisque c'est au programme et plus joli.
- Les assertions, c'est limite. Et de toute façon, c'est prévu pour le débugage seulement.

Ça c'est des détails. Ensuite, je n'ai pas compris la question (mon côté un peu maniaco-depressif). Ça veut dire quoi remplir le paquet de carte ? Avec quoi ? Ensuite, selon la manière dont on remplit le jeu, la carte numéro 20 ne sera pas celle du test proposé (moi j'ai eu le roi de trèfle).

Encore un sujet à éviter.

Une proposition:

**Script Python**

```
class Carte:
    """Initialise Couleur (entre 1 à 4), et Valeur (entre 1 à 13)"""
    def __init__(self, c, v):
        if c not in range(1, 5) or v not in range(1, 14):
            raise ValueError("Hauteur ou couleur invalide(s)")
        self.Couleur = c
        self.Valeur = v

    """Renvoie le nom de la Carte As, 2, ... 10, Valet, Dame, Roi"""
    def getNom(self):
        if (self.Valeur > 1 and self.Valeur < 11):
            return str(self.Valeur)
        elif self.Valeur == 11:
            return "Valet"
        elif self.Valeur == 12:
            return "Dame"
        elif self.Valeur == 13:
            return "Roi"
```

```
        else:
            return "As"

    """Renvoie la couleur de la Carte (parmi pique, coeur, carreau, trefle"""
    def getCouleur(self):
        return ['pique', 'coeur', 'carreau', 'trefle'][self.Couleur - 1]

class PaquetDeCarte:
    def __init__(self):
        self.contenu = []

    """Remplit le paquet de cartes"""
    def remplir(self):
        # self.contenu = [Carte(couleur, valeur) for couleur in range(1, 5) for valeur in range(1,
14)]
        # ou
        self.contenu = [Carte(couleur, valeur) for valeur in range(1, 14) for couleur in range(1,
5)]

    """Renvoie la Carte qui se trouve à la position donnée"""
    def getCarteAt(self, pos):
        if pos not in range(52):
            raise ValueError("Y a que 52 cartes, tricheur")
        return self.contenu[pos]
```