Corrigé Sujet 5 épreuve pratique NSI

Michel Billaud (michel.billaud@laposte.net)

1er février 2022

Table des matières

1 Licence

- Les notes sont publiées dans https://www.mbillaud.fr/notes/
- Sources dans https://github.com/MichelBillaud/notes-diverses

2 Le sujet

Se trouve sur la page https://eduscol.education.fr/2661/banque-des-epreuves-pratiques-de-specialite-nsi, dans https://eduscol.education.fr/document/33190/download

- minimum et maximum d'une liste
- construction d'un paquet de cartes

3 Question 1 : recherche du minimum et du maximum d'une liste de nombres

Résumé : retourner un dictionnaire qui contient le minimum et le maximum d'une liste de nombres

```
>>> 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}
```

3.1 Solution 1 (directe)

Sachant qu'il existe des fonctions `min` et `max` qui s'appliquent aux listes non vides, on peut écrire directement

```
def rechercheMinMax(tab):
    if len(tab) == 0:
        return {'min' : None, 'max' : None }
    else:
        return {'min' : min(tab), 'max' : max(tab)}
```

3.2 Solution 2

Si on en ignore l'existence, on peut les écrire

```
def min_tableau(tab):
    if len(tab) == 0:
        return None
    m = tab[0]
    for v in tab:
        if v < m:
            m = v
    return m

def max_tableau(tab):
   # similaire, en remplaçant < par >

def rechercheMinMax(tab):
        return {'min' : minTableau(tab),
                'max' : maxTableau(tab) }
```

3.3 Solution 3 (fusion de boucles)

Certains correcteurs s'attendent certainement à avoir une fonction qui fait tout d'un coup

```
def rechercheMinMax(tab):
    if len(tab) == 0:
        mi = None            # évitons les noms réservés
        ma = None            # min et max
    else:
        mi = tab[0]
        ma = tab[0]
        for v in tab:
            if v < mi:
                mi = v
            if v > ma:
                ma = v
    return { 'min' : mi, 'max' : ma }
```

ou pire, avec des indices. Contrôlés par une boucle `while`.

4 Exercice : paquet de carte

Résumé du sujet : on dispose d'une classe `Carte`, avec un constructeur paramétré par une couleur (entre 1 et 4) et une valeur (entre 1 et 13), et des accesseurs `getNom` et `getValeur`. On doit compléter le squelette d'une classe `PaquetDeCarte` de façon à pouvoir exécuter le scénario

```
>>>un Paquet = PaquetDeCarte()
>>> unPaquet.remplir()
>>> uneCarte = unPaquet.getCarteAt(20)
>>> print(uneCarte.getNom() + " de " + uneCarte.getCouleur())
6 de coeur
```

On demande aussi :

> puis ajouter des assertions dans l'initialiseur de `Carte`, ainsi que dans la méthode `getCarteAt()`. Le squelette fourni est le suivant :

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

    """Remplit le paquet de cartes"""
    def remplir(self):
        #A compléter                                    # 1

    """Renvoie la Carte qui se trouve à la position donnée"""
    def getCarteAt(self, pos):
        #A compléter                                    # 2
```

4.1 Résolution

Remplissage : les cartes seront visiblement stockées dans l'attribut `contenu`. qui est une liste. Pour remplir, il suffit d'y mettre les 4 x 13 cartes.

Accès à une carte : le paramètre `pos` servira d'indice dans la liste.

On va supposer que l' "initaliseur" du sujet désigne le corps du constructeur….

4.2 Solution

```
    def remplir(self):
        for couleur in range(1, 4+1):
            for valeur in range(1, 13+1):
                self.contenu.append(Carte(couleur, valeur))

    def getCarteAt(self, pos):
        assert 0 <= pos < len(self.contenu)
        return self.contenu[pos]
```

Dans l'assertion, on utilise de préférence l'enchainement de comparaisons spécifique à Python. A défaut, on imite les autres langages :

```
    assert 0 <= pos  and pos < len(self.contenu)
```

Pour le constructeur de `Carte` :

```
class Carte:
    def __init__(self, c, v):
        assert 1 <= c <= 4
        assert 1 <= v <= 13
        self.Couleur = c
        self.Valeur = v
```

4.3 Critique

Le code source présente ne tient aucun compte des conventions usuelles de programmation de Python :

- les noms d'attributs (`Couleur`) devraient être en minuscules (`couleur`)
- les noms composés de méthode (`getNom`) devraient être en snake_case (`get_nom`) et éviter de mélanger les langues (`nom()` et `couleur()` feraient l'affaire).
- le tableau de chaines "en dur" devrait être une constante

```
COULEURS = ['pique', 'coeur', 'carreau', 'trefle']
...
    def getCouleur(self):
        return COULEURS[self.couleur - 1]
```