Résultats du Diplôme du brevet

Christophe Viroulaud

Terminale - NSI

Algo 11

Chaque année l'Éducation Nationale publie les résultats du diplôme national du brevet sous forme d'un fichier `csv`.

Sommaire

Activité 1 :

1. Télécharger et extraire le fichier compressé **`dnb2018.zip`** sur le site https://cviroulaud.github.io.
2. Ouvrir le fichier avec un tableur pour observer les données.
3. Créer le fichier Python **`dnb.py`**
4. Importer le fichier **`csv`** et créer un itérateur à l'aide de la méthode **`csv.DictReader`**.

Avant de regarder la correction

- Prendre le temps de réfléchir,
- Analyser les messages d'erreur,
- Demander au professeur.

Correction

Le fichier contient de nombreuses informations pour chaque établissement, dont le taux de réussite au brevet.

```
1 f = open("dnb2018.csv")
2 dico = csv.DictReader(f)
3 f.close()
```

Sommaire

Un collège

L'objectif est de créer un arbre binaire de recherche ordonné selon les identifiants des collège.

> **Activité 2 :** Créer la classe `College` et son constructeur. Ce-dernier acceptera un dictionnaire `data` en paramètre et initialisera les attributs `id, nom, ville, departement, taux`. Enfin les attributs `gauche` et `droite` seront initialisés à `None`.

Avant de regarder la correction

- Prendre le temps de réfléchir,
- Analyser les messages d'erreur,
- Demander au professeur.

Correction

```python
class College:
    def __init__(self, data: dict):
        self.id = data["num_etab"]
        self.nom = data["patronyme"]
        self.ville = data["lib_commune"]
        self.departement = data["lib_dep"]
        self.taux = float(data["taux"])
        self.gauche = None
        self.droite = None
```

Activité 3 : Écrire la méthode `insere_fils(self, data: dict)` $\rightarrow$ `None` qui insère récursivement un nœud dans le sous-arbre gauche ou droite nœud en cours, en fonction des données contenues dans le dictionnaire `data`. Il faudra veiller à vérifier si le sous-arbre est vide ou non et agir en conséquence.

Avant de regarder la correction

- Prendre le temps de réfléchir,
- Analyser les messages d'erreur,
- Demander au professeur.

Correction

```python
def insere_fils(self, data: dict) -> None:
    id_fils = data["num_etab"]
    if id_fils < self.id:  # gauche
        if self.gauche is None: # sous-arbre vide
            self.gauche = College(data)
        else:
            self.gauche.insere_fils(data)
    else:  # droite
        if self.droite is None: # sous-arbre vide
            self.droite = College(data)
        else:
            self.droite.insere_fils(data)
```

Activité 4 : Écrire la méthode `recherche_fils(self, id_fils)` → `float` qui recherche récursivement le collège `id_fils` dans le sous-arbre gauche ou droite nœud en cours. La fonction renverra le taux de réussite du collège ou -1 si le collège n'est pas trouvé.

Avant de regarder la correction

- Prendre le temps de réfléchir,
- Analyser les messages d'erreur,
- Demander au professeur.

Correction

```python
def recherche_fils(self, id_fils: str) -> float:
    if id_fils == self.id:  # collège trouvé
        return self.taux
    elif id_fils < self.id:  # gauche
        if self.gauche is None:
            return -1
        else:
            return self.gauche.recherche_fils(id_fils)
    else:  # droite
        if self.droite is None:
            return -1
        else:
            return self.droite.recherche_fils(id_fils)
```

Sommaire

L'arbre binaire de recherche

Activité 5 : Créer la classe `Arbre` et son constructeur. Ce-dernier initialisera à `None` un attribut `racine`.

Avant de regarder la correction

- Prendre le temps de réfléchir,
- Analyser les messages d'erreur,
- Demander au professeur.

Correction

```
1 class Arbre:
2     def __init__(self):
3         self.racine = None
```

Activité 6 : Écrire la méthode `insere_college(self, data: dict)` $\rightarrow$ `None` qui initialise la `racine` avec une instance de `College` si elle est vide ou insère un nœud dans l'arbre en utilisant la méthode `insere_fils` de la classe `College`.

Avant de regarder la correction

- Prendre le temps de réfléchir,
- Analyser les messages d'erreur,
- Demander au professeur.

Correction

```
1 def insere_college(self, data: dict) -> None:
2     if self.racine is None:
3         self.racine = College(data)
4     else:
5         self.racine.insere_fils(data)
```

Activité 7 : Écrire la méthode `recherche_college(self, id: str)` $\rightarrow$ `float` qui recherche le collège en partant de la `racine`. La méthode renverra le taux de réussite ou -1 si le collège n'est pas trouvé.

Avant de regarder la correction

- Prendre le temps de réfléchir,
- Analyser les messages d'erreur,
- Demander au professeur.

Correction

```
1 def recherche_college(self, id: str) -> float:
2     if self.racine.id is None:
3         return -1
4     else:
5         return self.racine.recherche_fils(id)
```

Sommaire

Créer l'arbre

Activité 8 :

1. Dans le programme principal, créer une instance de **`Arbre`**.
2. En bouclant sur l'itérateur des données, créer l'arbre de recherche.

Avant de regarder la correction

- Prendre le temps de réfléchir,
- Analyser les messages d'erreur,
- Demander au professeur.

Correction

```
1 arbre = Arbre()
2 for ligne in dico:
3     arbre.insere_college(ligne)
```

Sommaire

Rechercher dans l'arbre

Activité 9 :

1. Rechercher le taux de réussite du collège Bertran de Born d'identifiant 0241042C.
2. En considérant que l'arbre est équilibré, combien d'appels récursifs seront nécessaires dans le pire des cas pour trouver le taux de réussite ?

Correction

```
1 print(arbre.recherche_college("0241042C"), "%
    ")
2 print(arbre.recherche_college("000"), "%")
```

Le fichier compte environ 9000 entrées.
$2^{13} = 8192$ donc la recherche effectuera entre 13 et 14 appels au maximum.