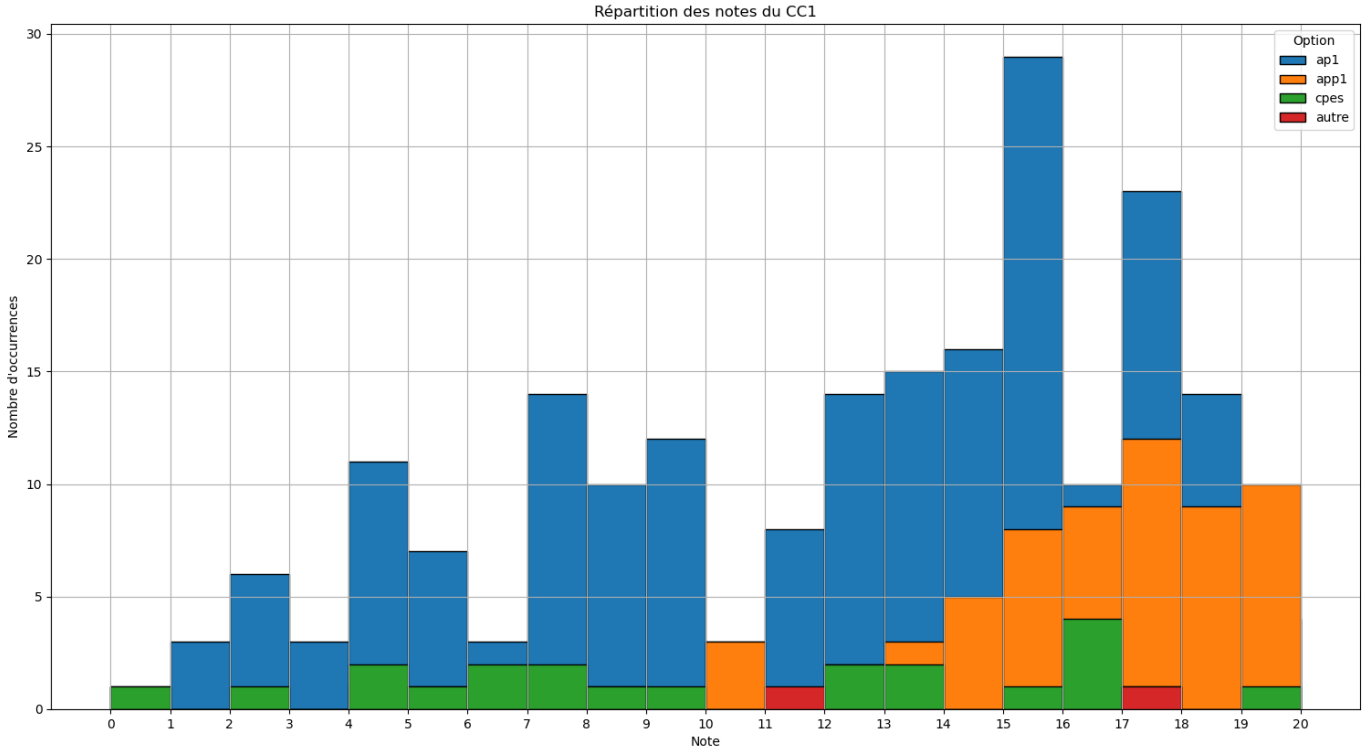
Algorithmique et Programmation 1

Lundi 13 novembre 2023

L1 Mathématiques - L1 Informatique
Semestre 1

CC1

Moyenne : 12,7/20 Médiane : 14/20



Dictionnaires

Exemple : Regrouper les informations d'un joueur

On veut modéliser un jeu à plusieurs joueurs, où les joueurs ont :

- Un pseudo
- Une couleur
- Un score

On peut bien sûr utiliser des listes :

```
pseudos = ["aze89", "yoplait", "azareth"]
couleurs = ["red", "yellow", "green"]
scores = [0, 0, 0]
```

Exemple : Regrouper les informations d'un joueur

On veut modéliser un jeu à plusieurs joueurs, où les joueurs ont :

- Un pseudo
- Une couleur
- Un score

On peut bien sûr utiliser des listes :

```
pseudos = ["aze89", "yoplait", "azareth"]
couleurs = ["red", "yellow", "green"]
scores = [0, 0, 0]
```

Mais :

- Il faut se trimballer trois listes
- Et ne pas se planter quand on ajoute un joueur
- Ou une caractéristique

Exemple : Regrouper les informations d'un joueur

Groupons donc par joueur :

```
joueur1 = ["qsdf89", "red", 25]
joueur2 = ["yoplait", "yellow", 20]
joueur3 = ["azerath", "green", 28]
joueurs = [joueur1, joueur2, joueur3]
```

Exemple : Regrouper les informations d'un joueur

Groupons donc par joueur :

```
joueur1 = ["qsdf89", "red", 25]
joueur2 = ["yoplait", "yellow", 20]
joueur3 = ["azerath", "green", 28]
joueurs = [joueur1, joueur2, joueur3]
```

Mais il faut se rappeler que :

- 0 correspond au pseudo
- 1 correspond à la couleur
- 2 correspond au score

On aimerait donc donner un nom à chaque indice !

Exemple : Regrouper les informations d'un joueur

C'est précisément ce que font les dictionnaires !

```
joueur1 = {'pseudo': 'qsdf89',
           'couleur': 'red',
           'score': 25}
joueur2 = {'pseudo': 'yoplait',
           'couleur': 'yellow',
           'score': 20}
joueur3 = {'pseudo': 'azerath',
           'couleur': 'green',
           'score': 28}
joueurs = [joueur1, joueur2, joueur3]
```

Les dictionnaires permettent ainsi de rassembler plusieurs informations concernant une personne.

Les dictionnaires

Dictionnaire : objet associant une liste de clés à des valeurs.

- Entrées/clés (*keys*) : d'un type de base *immutable*
- Valeurs associées (*values*) : de n'importe quel type
- Les paires (clé, valeur) sont appelées *objets* (*items*)

Un objet de type `dict` est :

- **une collection**
- **mutable**
- **hétérogène** (peut contenir des types différents)
- **itérable** (utilisable dans un `for`)

Fonctions de base

Créer un dictionnaire

```
# Pour créer un dictionnaire vide
dictionnaire = dict()  # Ou bien
dictionnaire = {}
# Pour l'initialiser avec des valeurs :
# voir précédemment
```

Accéder à une valeur via sa clé

```
dictionnaire[cle]
```

Modifier une valeur

```
dictionnaire[cle] = valeur
```

Fonctions de base

Exemple

```
joueur1 = {'pseudo': 'qsdf89',
           'couleur': 'red',
           'score': 25}
print(joueur1['couleur'])  # red
joueur1['score'] += 3
print(joueur1['score'])  # 28
```

Fonctions de base

Exemple

```
joueur1 = {'pseudo': 'qsdf89',
           'couleur': 'red',
           'score': 25}
print(joueur1['couleur'])  # red
joueur1['score'] += 3
print(joueur1['score'])  # 28
```

```
joueur1['ville'] = 'Champs'
print(joueur1)
# {'pseudo': 'qsdf89', 'couleur': 'red',
#  'score': 28, 'ville': 'Champs'}
```

Les éléments du dictionnaire : les clés (*keys*)

On peut accéder à l'ensemble des clés du dictionnaire :

```
dictionnaire.keys()
```

On peut donc tester si une clé est dans le dictionnaire :

```
if cle in dictionnaire.keys():
```

C'est la même chose d'écrire (et on préfère) :

```
if cle in dictionnaire:
```

On peut aussi itérer sur le dictionnaire :

```
for cle in dictionnaire:
# C'est pareil que (mais on préfère la version courte) :
for cle in dictionnaire.keys():
```

Les éléments du dictionnaire : les valeurs (*values*)

On peut accéder à l'ensemble des valeurs du dictionnaire :

```
dictionnaire.values()
```

On peut donc tester si une valeur est dans le dictionnaire :

```
if cle in dictionnaire.values():
```

On peut aussi itérer sur *les valeurs* du dictionnaire :

```
for cle in dictionnaire.values():
```

Les éléments du dictionnaire : les objets (*items*)

Enfin, on peut accéder à l'ensemble des objets du dictionnaire :

```
dictionnaire.items()
```

On peut donc tester si une couple (clé, valeur) est dans le dictionnaire :

```
if (cle, valeur) in dictionnaire.items():
# C'est pareil que d'écrire (et on préfère)
if dictionnaire[cle] == valeur
```

On peut aussi itérer sur *les objets* du dictionnaire :

```
for (cle, valeur) in dictionnaire.items():
```

Voir aussi le Jupyter Notebook.

Une autre application : un problème de comptage

Liste de prénoms américains (avec doublons) : quel est le plus courant ?

➔ Écrire un programme qui renvoie une liste de tuples `(prenom, nombre_occurrences)`.

Une autre application : un problème de comptage

Liste de prénoms américains (avec doublons) : quel est le plus courant ?

➔ Écrire un programme qui renvoie une liste de tuples `(prenom, nombre_occurrences)`.

Première approche

- Déterminer la liste des prénoms sans doublons
- Pour chaque prénom dans cette liste, compter combien de fois il apparaît

Une autre application : un problème de comptage

Liste de prénoms américains (avec doublons) : quel est le plus courant ?

➔ Écrire un programme qui renvoie une liste de tuples `(prenom, nombre_occurrences)`.

Deuxième approche

- Créer une liste vide `occurrences`
- Parcourir la liste des prénoms (avec doublons)
- Pour chaque prénom `prenom` :
  - S'il n'apparaît pas dans `occurrences`, y ajouter une liste `[prenom, 1]`
  - S'il apparaît déjà dans `occurrences`, ajouter 1 au nombre d'occurrences correspondant

Une autre application : un problème de comptage

Liste de prénoms américains (avec doublons) : quel est le plus courant ?

➔ Écrire un programme qui renvoie une liste de tuples `(prenom, nombre_occurrences)`.

Deuxième approche

- Créer une liste vide `occurrences`
- Parcourir la liste des prénoms (avec doublons)
- Pour chaque prénom `prenom` :
  - S'il n'apparaît pas dans `occurrences`, y ajouter une liste `[prenom, 1]`
  - S'il apparaît déjà dans `occurrences`, ajouter 1 au nombre d'occurrences correspondant

C'est pénible... Est-ce qu'on peut faire mieux ?

Le problème de comptage avec des dictionnaires

Liste de prénoms américains (avec doublons) : quel est le plus courant ?

➔ Écrire un programme qui renvoie un dictionnaire contenant les occurrences de chaque prénom.

- Créer un dictionnaire vide `occurrences`
- Parcourir la liste des prénoms (avec doublons)
- Pour chaque prénom `prenom` :
  - S'il n'apparaît pas dans `occurrences`, y ajouter une entrée `prenom : 1`
  - S'il apparaît déjà dans `occurrences`, ajouter 1 au nombre d'occurrences correspondant

Le problème de comptage avec des dictionnaires

Liste de prénoms américains (avec doublons) : quel est le plus courant ?

➔ Écrire un programme qui détermine quels sont les prénoms utilisés.

- Calculer le dictionnaire des occurrences
- Afficher ses clés une par une.

Le problème de comptage avec des dictionnaires

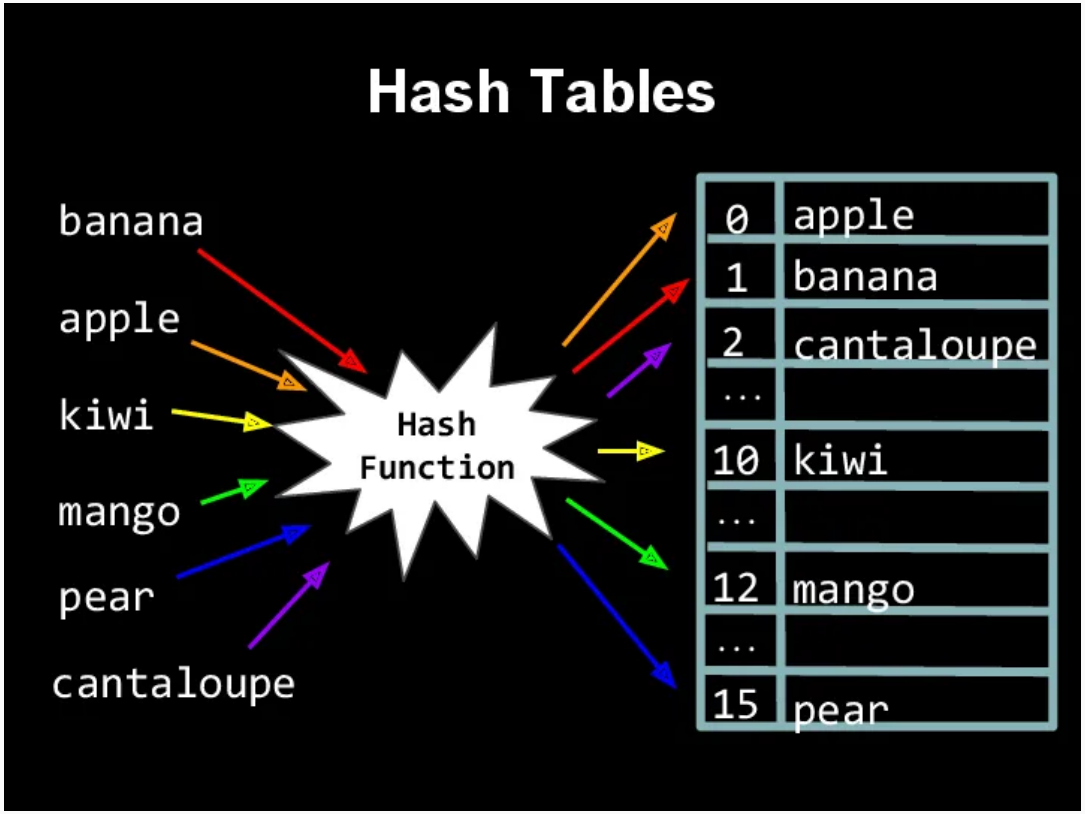
Liste de prénoms américains (avec doublons) : quel est le plus courant ?

➔ Écrire un programme qui détermine quel est le prénom le plus courant.

- Calculer le dictionnaire des occurrences
- Parcourir ses paires (clé, valeur) pour déterminer la clé ayant la plus grande valeur

Comment ça marche ?

Les tables de hachage



Restrictions sur les clés

Les clés doivent être *hashables*. En pratique : d'un type de base *immutable*.

Les fichiers

Itérables (encore une fois)

Définition

Un objet dont on peut parcourir les éléments.

- ➔ Les dictionnaires `dict`
- ➔ Les intervalles `range`
- ➔ Les listes `list`
- ➔ Les chaînes de caractères `str`
- ➔ Les tuples `tuple`
- Les ensembles `set`
- Les fichiers `file`

Un système de sauvegarde

Quand on quitte un jeu, les données en mémoire de Python sont perdues. Comment faire ?

Un système de sauvegarde

Quand on quitte un jeu, les données en mémoire de Python sont perdues. Comment faire ?

Écrire dans un fichier !

```
def sauvegarde(pseudos, fichier):
    f = open('pseudos.txt', 'w')  # w = write
    for pseudo in pseudos:
        f.write(pseudo)
        f.write('\n')  # retour à la ligne
    f.close()  # penser à fermer le fichier
```

Un système de sauvegarde

Mais après il faut lire !

Un système de sauvegarde

Mais après il faut lire !

```
def recuperer_sauvegarde(fichier):
    f = open('pseudos.txt', 'r')  # r = read
    pseudos = []
    for ligne in f:
        # Pour retirer le retour à la ligne
        pseudo = ligne.strip('\n')
        pseudos.append(pseudo)
    f.close()
    return pseudos
```

Fermeture automatique

C'est pénible de devoir penser à fermer le fichier à chaque fois.

Python peut le faire automatiquement pour nous :

```
def sauvegarde(pseudos, fichier):
    # Mieux :
    with open('pseudos.txt', 'w') as f: # w = write
        for pseudo in pseudos:
            f.write(pseudo)
            f.write('\n')  # retour à la ligne
    # le fichier est fermé automatiquement
```

Fermeture automatique

Idem quand on veut le lire :

```
def recuperer_sauvegarde(fichier):
    # Mieux :
    with open('pseudos.txt', 'r') as f: # r = read
        pseudos = []
        for ligne in f:
            # Pour retirer le retour à la ligne
            pseudo = ligne.strip('\n')
            pseudos.append(pseudo)
    # le fichier est fermé automatiquement
    return pseudos
```

Les fichiers

Pour en savoir plus : Jupyter Notebook (section « Fichiers »).