Informatique 1
Introduction à la programmation

Mission 7 : introduction

Kim Mens Siegfried Nijssen Charles Pecheur

Associer des Informations

Présumez que nous voulons stocker une relation entre les noms de fruits et les nombres entiers

apple	banana	cherry	grape	mango	peach	pear
1	10	5	4	8	9	1

Comment on fait ça en Python?

Approche 1:
Listes de Tuples

apple	banana	cherry	grape	mango	peach	pear
1	10	5	4	8	9	1

[("Apple",1), ("Banana",10), ("Cherry",5), ("Grape",4), ("Mango",8), ("Peach",9), ("Pear",1)]

Si on trie les tuples par nom de fruit, on peut utiliser la recherche dichotomique pour trouver le nombre d'un fruit

On ne veut pas reimplémenter la recherche dichotomique tout le temps

Approche 2: Dictionnaires Intégrés de Python

Dictionnaire: Structure de données qui permet de stocker et rechercher des informations associées à une valeur facilement

Notation (1)

Initialiser un dictionnaire et récupérer les informations associées

apple	banana	cherry	grape	mango	peach	pear
1	10	5	4	8	9	1

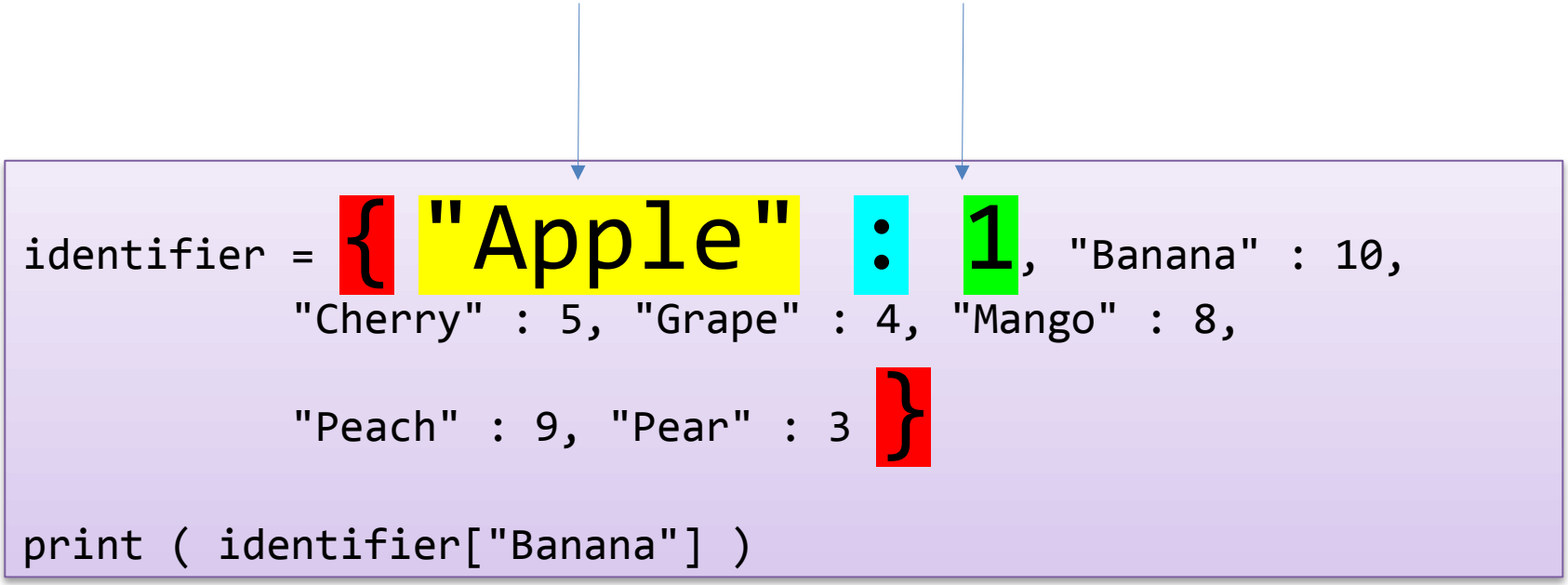
```
identifier = { "Apple" : 1, "Banana" : 10, "Cherry" : 5, \
           "Grape" : 4, "Mango" : 8, "Peach" : 9, "Pear" : 1 }

print ( identifier["Banana"] )
```

```
10
```

Notation (2)

Clé (Key) Valeur (Value)



```
identifier = { "Apple" : 1, "Banana" : 10,
               "Cherry" : 5, "Grape" : 4, "Mango" : 8,

               "Peach" : 9, "Pear" : 3 }

print ( identifier["Banana"] )
```

Avantages

- Notation facile: tout valeur **immuable** peut être utilisé comme "index" (clé)
  pas seulement des nombres entiers, mais aussi des chaînes de caractères, des tuples, … et leurs combinaisons!

```
weird = { "Banana" : 2, (2,5) : 3, 10: 5 }
print ( weird["Banana"], weird[(2,5)], weird[10] )
```

- Structure de données efficace

Utilisés souvent en Python au lieu de listes triées

Notation (2)

- Créer un dictionnaire vide:

```
d = { }
```

- Changer **ou** créer une association:

```
d["Apple"] = 1
```

Manipulation de Dictionnaires

- Code correct:

(1) Type des Clés:Valeurs

```
Tuples=[("Apple",1),("Pear",4),("Pear",3)]
```

Manipulation de Dictionnaires

- Code correct:

(1) Type des Clés:Valeurs

```
Tuples=[("Apple",1),("Pear",4),("Pear",3)]
d = { }
```

(2) Créer

Manipulation de Dictionnaires

- Code correct:

(1) Type des Clés:Valeurs

```
Tuples=[("Apple",1),("Pear",4),("Pear",3)]
d = { }
for fruit in Tuples :
    d[fruit[0]] = fruit[1]
```

(2) Créer

(3) Chaque clef : Ajouter

Manipulation de Dictionnaires

- Code correct:

(1) Type des Clés:Valeurs

```
Tuples=[("Apple",1),("Pear",4),("Pear",3)]
d = { }    (2) Créer
for fruit in Tuples :
    d[fruit[0]] = fruit[1]
                (3) Chaque clef : Ajouter
print (d["Pear"])
```

```
{ }
{ "Apple" : 1 }
{ "Apple" : 1, "Pear" : 4 }
{ "Apple" : 1, "Pear" : 3 }
3
```

Manipulation de Dictionnaires

- Code qui donne une erreur:

```
Tuples=[("Apple",1),("Pear",4),("Pear",3)]
d = { }
for fruit in Tuples :
   d[fruit[0]] = fruit[1]
print (d["Grape"])
```

{ "Apple" : 1, "Pear" : 3 }

KeyError: 'Grape'

Manipulation de Dictionnaires

Comment éviter cette erreur?

Approche 1 (défensive): tester si la clé est présente

```
Tuples=[("Apple",1),("Pear",4),("Pear",3)]
d = { }
for fruit in Tuples :
    d[fruit[0]] = fruit[1]
if "Grape" in d:
   print (d["Grape"])
```

{ "Apple" : 1, "Pear" : 3 }

in ici évalue seulement la présence de la clé !!!

Manipulation de Dictionnaires

Comment éviter cette erreur?

Approche 2 (exceptions)

```
Tuples=[("Apple",1),("Pear",4),("Pear",3)]
d = { }
for fruit in Tuples :
    d[fruit[0]] = fruit[1]
try:
  print (d["Grape"])
except:
  print ("Not found")
```

{ "Apple" : 1, "Pear" : 3 }

Not found

Manipulation de Dictionnaires

Comment éviter cette erreur?

Approche 3 (get)

```
Tuples=[("Apple",1),("Pear",4),("Pear",3)]
d = { }
for fruit in Tuples :
   d[fruit[0]] = fruit[1]
print (d.get("Grape",-1))
```

{ "Apple" : 1, "Pear" : 3 }

-1

Valeur retournée si la clé n'est pas présente

Parcourir les Dictionnaires

- Parcourir les clés

```
identifier = { "Apple" : 1, "Banana" : 10, "Cherry" : 5, \
            "Grape" : 4, "Mango" : 8, "Peach" : 9, "Pear" : 3 }

for key in identifier.keys ():
  print ( key )
```

```
Apple
Banana
Cherry
Grape
Mango
Peach
Pear
```

Parcourir les Dictionnaires

- Parcourir les clés (simplifié)

```
identifier = { "Apple" : 1, "Banana" : 10, "Cherry" : 5, \
            "Grape" : 4, "Mango" : 8, "Peach" : 9, "Pear" : 3 }

for key in identifier:
  print ( key )
```

```
Apple
Banana
Cherry
Grape
Mango
Peach
Pear
```

Parcourir les Dictionnaires

- Parcourir les valeurs

```
identifier = { "Apple" : 1, "Banana" : 10, "Cherry" : 5, \
            "Grape" : 4, "Mango" : 8, "Peach" : 9, "Pear" : 3 }

for value in identifier.values ():
  print ( value )
```

```
1
10
5
4
8
9
3
```

Parcourir les Dictionnaires

- Parcourir les combinaisons (clé, valeur)

```
identifier = { "Apple" : 1, "Banana" : 10, "Cherry" : 5, \
            "Grape" : 4, "Mango" : 8, "Peach" : 9, "Pear" : 3 }

for key, value in identifier.items ():
  print ( key, value )
```

```
Apple 1
Banana 10
Cherry 5
Grape 4
Mango 8
Peach 9
Pear 3
```

Structures de Données Imbriquées

On peut imbriquer
des listes/tuples dans des dictionnaires,
des dictionnaires dans des listes/tuples,
des dictionnaires dans des dictionnaires, ...

```
matrix = { (0,0): 1, (0,2): 2, (1,1): 2, (2,2): 4, (3,3): 1 }

nested = { "Jean" : { "Beer" : 2, "Wine" : 3 }, \
           "Valérie" : { "Wine" : 4 } }

other_nested = { "Jean" : [ "Beer", "Wine" ], \
                 "Valérie" : [ "Wine" ] }

l = [ { "Beer" : 2, "Wine" : 3 }, { "Wine" : 4 } ]
```

Dictionnaires

- Peuvent associer une liste à une clé

```
postcodes = { "Bruxelles": [1000], \
              "Louvain-la-Neuve": [1348] }
postcodes["Bruxelles"].append ( 1020 )
```

Copier des Dictionnaires

On veut créer un nouveau dictionnaire qui reflète

- la création d'une nouvelle commune "Bruxelles-la-Neuve" avec code postal 1001
- l'ajout du code postal 1002 à Bruxelles
- À base d’un dictionnaire d’origine

Copier des Dictionnaires

```
postcodes2022 = { "Bruxelles": [1000], \
                  "Louvain-la-Neuve": [1348] }
… (code que l'on veut écrire) …
print(postcodes2022)
print(postcodes2100)
```

```
{ "Bruxelles": [1000], "Louvain-la-Neuve": [1348] }
{ "Bruxelles": [1000,1002], "Louvain-la-Neuve": [1348],
  "Bruxelles-la-Neuve": 1001 }
```

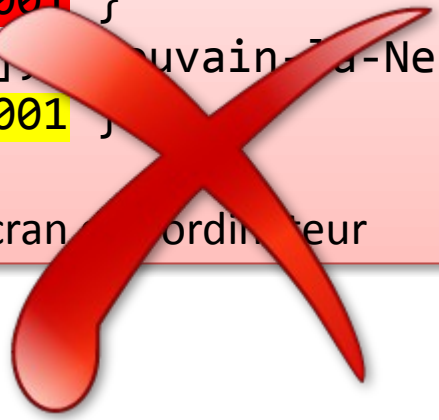
Écran de l'ordinateur

Copier des Dictionnaires

```
postcodes2022 = { "Bruxelles": [1000], \
                  "Louvain-la-Neuve": [1348] }
postcodes2100 = postcodes2022
postcodes2100["Bruxelles-la-Neuve"] = [1001]
postcodes2100["Bruxelles"].append ( 1002 )
print(postcodes2022)
print(postcodes2100)
```

```
{ "Bruxelles": [1000,1002], "Louvain-la-Neuve": [1348],
  "Bruxelles-la-Neuve": 1001 }
{ "Bruxelles": [1000,1002], "Louvain-la-Neuve": [1348],
  "Bruxelles-la-Neuve": 1001 }
```

Écran d'ordinateur



Copier des Dictionnaires

```
import copy
postcodes2022 = { "Bruxelles": [1000], \
                  "Louvain-la-Neuve": [1348] }
postcodes2100 = copy.deepcopy(postcodes2022)
postcodes2100["Bruxelles-la-Neuve"] = [1001]
postcodes2100["Bruxelles"].append ( 1002 )
print(postcodes2022)
print(postcodes2100)
```

```
{ "Bruxelles": [1000], "Louvain-la-Neuve": [1348] }
{ "Bruxelles": [1000,1002], "Louvain-la-Neuve": [1348],
  "Bruxelles-la-Neuve": 1001 }
```

Écran de l'ordinateur

Mission 7

- Objectifs
  - dictionnaires

- Problème
  - Création d'un outil pour chercher dans un fichier de texte en utilisant un index stocké dans un dictionnaire