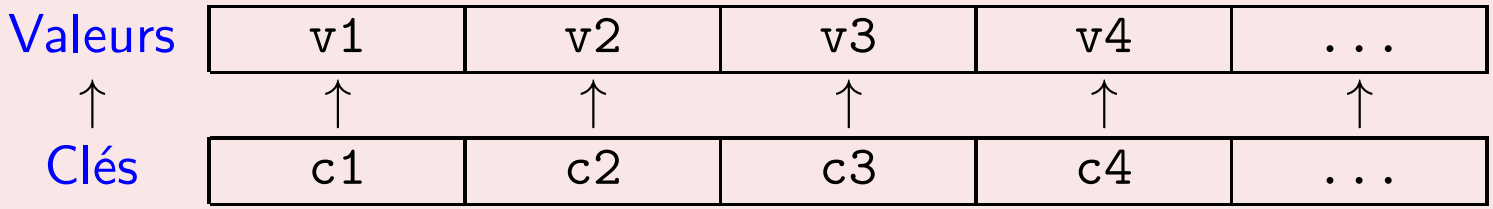
Les dictionnaires de Python

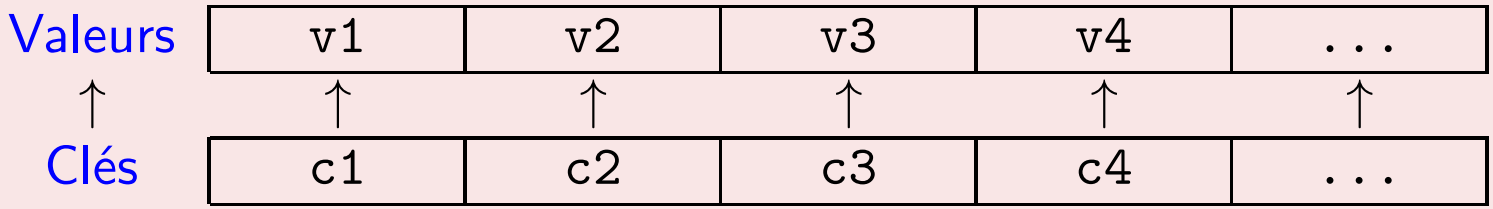
- Les dictionnaires de Python permettent de stocker des données sous forme de tableau associant une clé à une valeur :



Exemples

Les dictionnaires de Python

- Les dictionnaires de Python permettent de stocker des données sous forme de tableau associant une clé à une valeur :

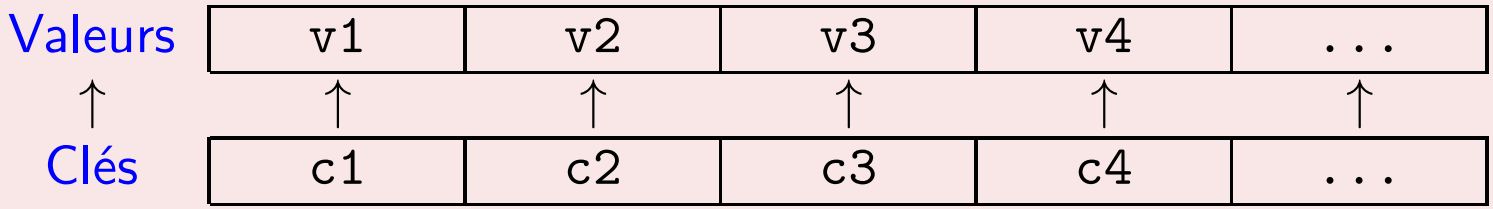


- Un dictionnaire se note entre accolades : { et }

Exemples

Les dictionnaires de Python

- Les dictionnaires de Python permettent de stocker des données sous forme de tableau associant une clé à une valeur :



- Un dictionnaire se note entre accolades : `{` et `}`
- Les paires clés/valeurs sont séparés par des virgules `,`

Exemples

Les dictionnaires de Python

- Les dictionnaires de Python permettent de stocker des données sous forme de tableau associant une clé à une valeur :

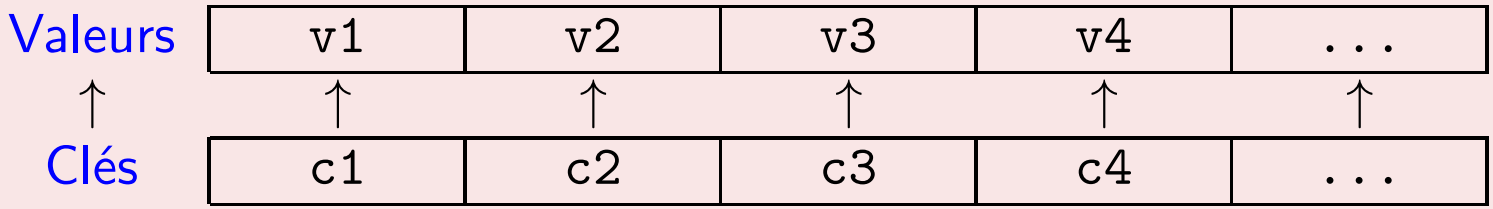
Valeurs	v1	v2	v3	v4	...
↑	↑	↑	↑	↑	↑
Clés	c1	c2	c3	c4	...

- Un dictionnaire se note entre accolades : `{` et `}`
- Les paires clés/valeurs sont séparés par des virgules `,`
- Le caractère `:` sépare une clé de la valeur associée.

Exemples

Les dictionnaires de Python

- Les dictionnaires de Python permettent de stocker des données sous forme de tableau associant une clé à une valeur :



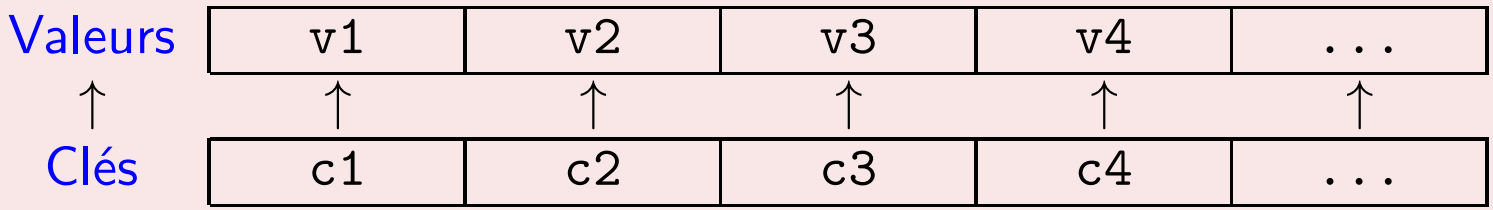
- Un dictionnaire se note entre accolades : `{` et `}`
- Les paires clés/valeurs sont séparés par des virgules `,`
- Le caractère `:` sépare une clé de la valeur associée.

Exemples

- Un dictionnaire contenant des objets et leurs prix :

Les dictionnaires de Python

- Les dictionnaires de Python permettent de stocker des données sous forme de tableau associant une clé à une valeur :



- Un dictionnaire se note entre accolades : { et }
- Les paires clés/valeurs sont séparés par des virgules ,
- Le caractère : sépare une clé de la valeur associée.

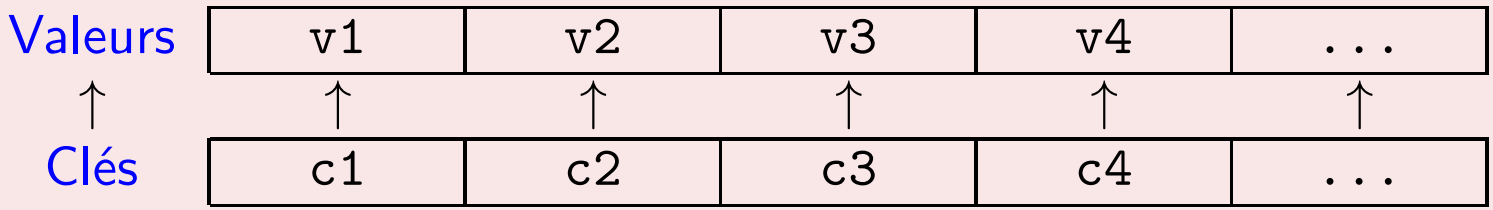
Exemples

- Un dictionnaire contenant des objets et leurs prix :

```
prix = { "verre":12 , "tasse" : 8, "assiette" : 16}
```

Les dictionnaires de Python

- Les dictionnaires de Python permettent de stocker des données sous forme de tableau associant une clé à une valeur :



- Un dictionnaire se note entre accolades : { et }
- Les paires clés/valeurs sont séparés par des virgules ,
- Le caractère : sépare une clé de la valeur associée.

Exemples

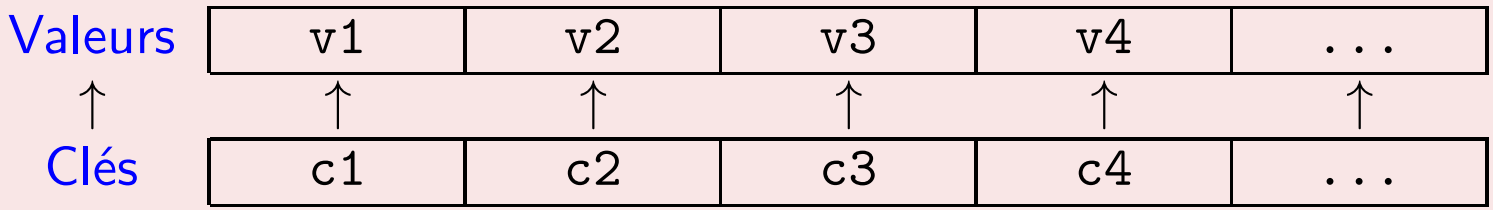
- Un dictionnaire contenant des objets et leurs prix :

```
prix = { "verre":12 , "tasse" : 8, "assiette" : 16}
```

- Un dictionnaire traduisant des couleurs du français vers l'anglais

Les dictionnaires de Python

- Les dictionnaires de Python permettent de stocker des données sous forme de tableau associant une clé à une valeur :



- Un dictionnaire se note entre accolades : { et }
- Les paires clés/valeurs sont séparés par des virgules ,
- Le caractère : sépare une clé de la valeur associée.

Exemples

- Un dictionnaire contenant des objets et leurs prix :

```
prix = { "verre":12 , "tasse" : 8, "assiette" : 16}
```

- Un dictionnaire traduisant des couleurs du français vers l'anglais

```
couleurs = { "vert":"green" , "bleu" : "blue", "rouge" : "red" }
```

Opérations sur un dictionnaire

- On accède aux éléments d'un dictionnaire avec la syntaxe `nom_dictionnaire[cle]`

Opérations sur un dictionnaire

- On accède aux éléments d'un dictionnaire avec la syntaxe
  `nom_dictionnaire[cle]`

```
prix = { "verre":12 , "tasse" : 8, "assiette" : 16, "plat" : 30 }
```

Par exemple, `prix["verre"]` contient 12

Opérations sur un dictionnaire

- On accède aux éléments d'un dictionnaire avec la syntaxe `nom_dictionnaire[cle]`

```
prix = { "verre":12 , "tasse" : 8, "assiette" : 16, "plat" : 30 }
```

Par exemple, `prix["verre"]` contient 12

- On peut ajouter une clé à un dictionnaire existant en effectuant une affectation `nom_dictionnaire[nouvelle_cle]=nouvelle_valeur`

Opérations sur un dictionnaire

- On accède aux éléments d'un dictionnaire avec la syntaxe `nom_dictionnaire[cle]`

```
prix = { "verre":12 , "tasse" : 8, "assiette" : 16, "plat" : 30 }
```

Par exemple, `prix["verre"]` contient 12

- On peut ajouter une clé à un dictionnaire existant en effectuant une affectation `nom_dictionnaire[nouvelle_cle]=nouvelle_valeur`

On ajoute un nouvel objet avec son prix :

```
prix["couteau"]=20
```

Opérations sur un dictionnaire

- On accède aux éléments d'un dictionnaire avec la syntaxe `nom_dictionnaire[cle]`

```
prix = { "verre":12 , "tasse" : 8, "assiette" : 16, "plat" : 30 }
```

Par exemple, `prix["verre"]` contient 12

- On peut ajouter une clé à un dictionnaire existant en effectuant une affectation `nom_dictionnaire[nouvelle_cle]=nouvelle_valeur`

On ajoute un nouvel objet avec son prix :

```
prix["couteau"]=20
```

- On peut modifier la valeur associée à une clé avec une affectation `nom_dictionnaire[cle]=nouvelle_valeur`

Opérations sur un dictionnaire

- On accède aux éléments d'un dictionnaire avec la syntaxe `nom_dictionnaire[cle]`

```
prix = { "verre":12 , "tasse" : 8, "assiette" : 16, "plat" : 30 }
```

Par exemple, `prix["verre"]` contient 12

- On peut ajouter une clé à un dictionnaire existant en effectuant une affectation `nom_dictionnaire[nouvelle_cle]=nouvelle_valeur`

On ajoute un nouvel objet avec son prix :

```
prix["couteau"]=20
```

- On peut modifier la valeur associée à une clé avec une affectation `nom_dictionnaire[cle]=nouvelle_valeur`

Le pris d'une tasse passe à 10 :

```
prix["tasse"]=10
```

Présence dans un dictionnaire

- Attention, essayer d'accéder à une clé qui n'est pas dans un dictionnaire renvoie une erreur !

Présence dans un dictionnaire

- Attention, essayer d'accéder à une clé qui n'est pas dans un dictionnaire renvoie une erreur !

  Il n'y a pas de clé `"fourchette"` dans le dictionnaire prix, donc `prix["fourchette"]` renvoie une erreur (`KeyError`).

Présence dans un dictionnaire

- Attention, essayer d'accéder à une clé qui n'est pas dans un dictionnaire renvoie une erreur !

  Il n'y a pas de clé `"fourchette"` dans le dictionnaire prix, donc `prix["fourchette"]` renvoie une erreur (`KeyError`).

- On teste la présence d'une clé dans un dictionnaire avec `cle in nom_dictionnaire`

Présence dans un dictionnaire

- Attention, essayer d'accéder à une clé qui n'est pas dans un dictionnaire renvoie une erreur !

  Il n'y a pas de clé `"fourchette"` dans le dictionnaire prix, donc `prix["fourchette"]` renvoie une erreur (`KeyError`).

- On teste la présence d'une clé dans un dictionnaire avec `cle in nom_dictionnaire`

  la fourchette n'est pas dans le dictionnaire prix
  Le test `fourchette in prix` renvoie `False`

Présence dans un dictionnaire

- Attention, essayer d'accéder à une clé qui n'est pas dans un dictionnaire renvoie une erreur !

  Il n'y a pas de clé `"fourchette"` dans le dictionnaire prix, donc `prix["fourchette"]` renvoie une erreur (`KeyError`).

- On teste la présence d'une clé dans un dictionnaire avec `cle in nom_dictionnaire`

  la fourchette n'est pas dans le dictionnaire prix

  Le test `fourchette in prix` renvoie `False`

  ❗ Ce test d'appartenance s'effectue en temps constant (indépendant de la taille du dictionnaire)

- On peut supprimer une clé existante dans un dictionnaire avec `del nom_dictionnaire[cle]`

Présence dans un dictionnaire

- Attention, essayer d'accéder à une clé qui n'est pas dans un dictionnaire renvoie une erreur !

  Il n'y a pas de clé `"fourchette"` dans le dictionnaire prix, donc `prix["fourchette"]` renvoie une erreur (`KeyError`).

- On teste la présence d'une clé dans un dictionnaire avec `cle in nom_dictionnaire`

  la fourchette n'est pas dans le dictionnaire prix

  Le test `fourchette in prix` renvoie `False`

  ❗ Ce test d'appartenance s'effectue en temps constant (indépendant de la taille du dictionnaire)

- On peut supprimer une clé existante dans un dictionnaire avec `del nom_dictionnaire[cle]`

  On supprimer le couteau :

  ```
  del prix["couteau"]
  ```

Parcours d'un dictionnaire

- Le parcours par clé s'effectue directement avec `for cle in nom_dictionnaire`

Parcours d'un dictionnaire

- Le parcours par clé s'effectue directement avec `for cle in nom_dictionnaire`

```
prix = { "verre":12 , "tasse" : 8, "assiette" : 16, "plat" : 30 }
```

Par exemple, `for objet in prix` permettra à la variable `objet` de prendre successivement les valeurs des clés : `"verre"`, `"tasse"`, `"assiette"` et `"plat"`.

Parcours d'un dictionnaire

- Le parcours par clé s'effectue directement avec `for cle in nom_dictionnaire`

  ```
  prix = { "verre":12 , "tasse" : 8, "assiette" : 16, "plat" : 30 }
  ```

  Par exemple, `for objet in prix` permettra à la variable `objet` de prendre successivement les valeurs des clés : `"verre"`, `"tasse"`, `"assiette"` et `"plat"`.
- Le parcours par valeur s'effectue en ajoutant `.values()` au nom du dictionnaire : `for valeur in nom_dictionnaire.values()`

Parcours d'un dictionnaire

- Le parcours par clé s'effectue directement avec `for cle in nom_dictionnaire`

  ```
  prix = { "verre":12 , "tasse" : 8, "assiette" : 16, "plat" : 30 }
  ```

  Par exemple, `for objet in prix` permettra à la variable `objet` de prendre successivement les valeurs des clés : `"verre"`, `"tasse"`, `"assiette"` et `"plat"`.
- Le parcours par valeur s'effectue en ajoutant `.values()` au nom du dictionnaire : `for valeur in nom_dictionnaire.values()`

  Par exemple, `for p in prix.values()` permettra à la variable `p` de prendre successivement les valeurs du dictionnaire : 12, 8 , 16 et 30.

Exemple

On dispose d'une liste de nombres entiers et on veut obtenir le nombre d'occurence du (ou des) entiers(s) les plus fréquents dans cette liste. Par exemple si la liste est `[1,7,1,3,4,1,3,4,3,1,5,108,2,3]` alors la réponse est 4, car les entiers les plus fréquents sont 1 et 3 qui apparaissent tous les deux à 4 reprises.

1. Proposer une solution qui pour chaque élément de la liste calcule son nombre d'apparitions à l'aide d'une fonction `compte_occurence`

Exemple

On dispose d'une liste de nombres entiers et on veut obtenir le nombre d'occurence du (ou des) entiers(s) les plus fréquents dans cette liste. Par exemple si la liste est `[1,7,1,3,4,1,3,4,3,1,5,108,2,3]` alors la réponse est 4, car les entiers les plus fréquents sont 1 et 3 qui apparaissent tous les deux à 4 reprises.

1. Proposer une solution qui pour chaque élément de la liste calcule son nombre d'apparitions à l'aide d'une fonction `compte_occurence`
2. Proposer une solution utilisant un dictionnaire dont les clés sont les entiers présents dans la liste et les valeurs leurs nombre d'apparitions

Exemple

On dispose d'une liste de nombres entiers et on veut obtenir le nombre d'occurence du (ou des) entiers(s) les plus fréquents dans cette liste. Par exemple si la liste est `[1,7,1,3,4,1,3,4,3,1,5,108,2,3]` alors la réponse est 4, car les entiers les plus fréquents sont 1 et 3 qui apparaissent tous les deux à 4 reprises.

1. Proposer une solution qui pour chaque élément de la liste calcule son nombre d'apparitions à l'aide d'une fonction `compte_occurence`
2. Proposer une solution utilisant un dictionnaire dont les clés sont les entiers présents dans la liste et les valeurs leurs nombre d'apparitions
3. Commenter l'efficacité de ces deux solutions.

Correction question 1

```
1  def compte_occurence(elt,liste):
2      occ = 0
3      for x in liste:
4          if x==elt:
5              occ+=1
6      return occ
7
8  def plus_frequent(liste):
9      max_occ = 0
10     for elt in liste:
11         elt_occ = compte_occurence(elt,liste)
12         if elt_occ>max_occ:
13             max_occ = elt_occ
14     return max_occ
15
```

Correction question 2

```
1  def plus_frequent(liste):
2      nb_occ = {}
3      for elt in liste:
4          if elt not in nb_occ:
5              nb_occ[elt] = 1
6          else:
7              nb_occ[elt] += 1
8      return max(nb_occ[elt] for elt in nb_occ)
```

Correction question 2

```
1  def plus_frequent(liste):
2      nb_occ = {}
3      for elt in liste:
4          if elt not in nb_occ:
5              nb_occ[elt] = 1
6          else:
7              nb_occ[elt] += 1
8      return max(nb_occ[elt] for elt in nb_occ)
```

Correction question 3

La solution avec les dictionnaires est bien plus efficace car on effectue un seul parcours de la liste et que le test d'appartenance au dictionnaire est une opération élémentaire (temps constant en moyenne).

Implémentation des dictionnaires

- On crée un tableau T de liste de longueur N (donc indicé par les entiers $[\![0; N-1]\!]$).

Implémentation des dictionnaires

- On crée un tableau T de liste de longueur N (donc indicé par les entiers $[\![0; N-1]\!]$).
- Une fonction de hachage h transforme les clés en entier. Les clés doivent donc être non mutables (ce qui exclu les listes). Ces entiers sont ramenés dans l'intervalle $[\![0; N-1]\!]$ à l'aide d'un modulo.

Implémentation des dictionnaires

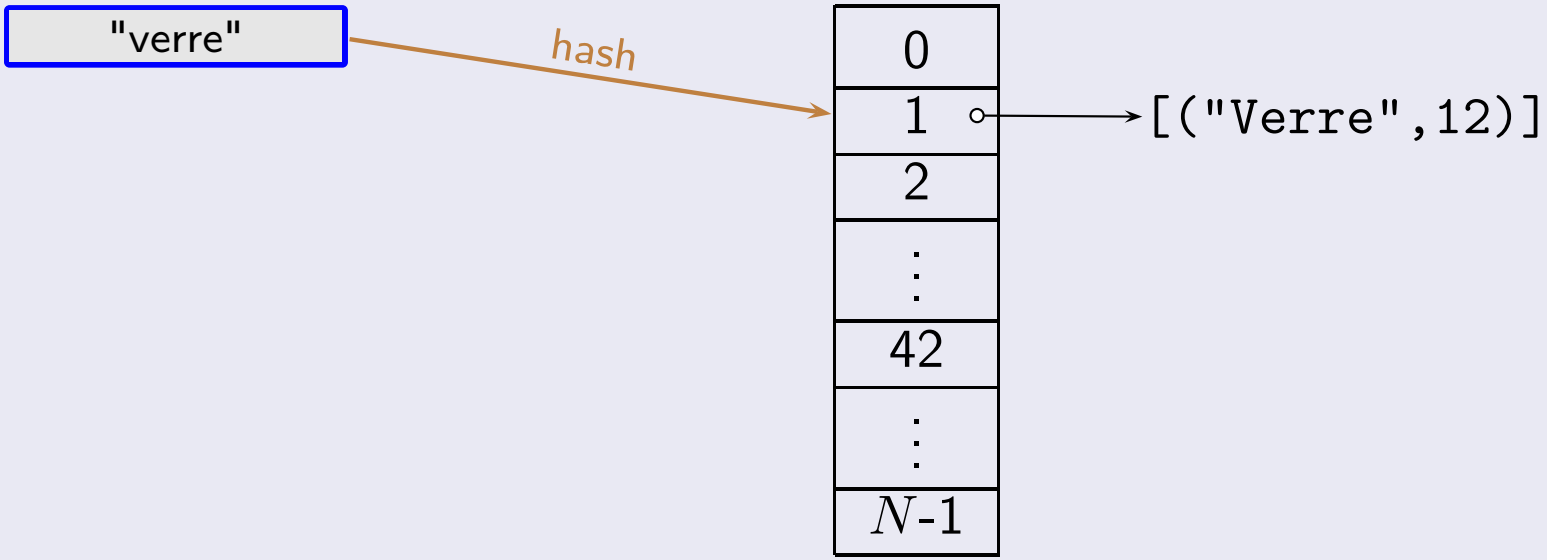
- On crée un tableau T de liste de longueur N (donc indicé par les entiers $[\![0; N-1]\!]$).
- Une fonction de hachage h transforme les clés en entier. Les clés doivent donc être non mutables (ce qui exclu les listes). Ces entiers sont ramenés dans l'intervalle $[\![0; N-1]\!]$ à l'aide d'un modulo.
- Chaque paire de clé/valeur (c, v) est stockée dans le tableau T à l'indice $h(c)$ (modulo N)

Implémentation des dictionnaires

- On crée un tableau T de liste de longueur N (donc indicé par les entiers $[\![0; N-1]\!]$).
- Une fonction de hachage h transforme les clés en entier. Les clés doivent donc être non mutables (ce qui exclu les listes). Ces entiers sont ramenés dans l'intervalle $[\![0; N-1]\!]$ à l'aide d'un modulo.
- Chaque paire de clé/valeur (c, v) est stockée dans le tableau T à l'indice $h(c)$ (modulo N)
- Le cas où dont deux clés différentes $c1$ et $c2$ produisent le même indice s'appelle une collision.

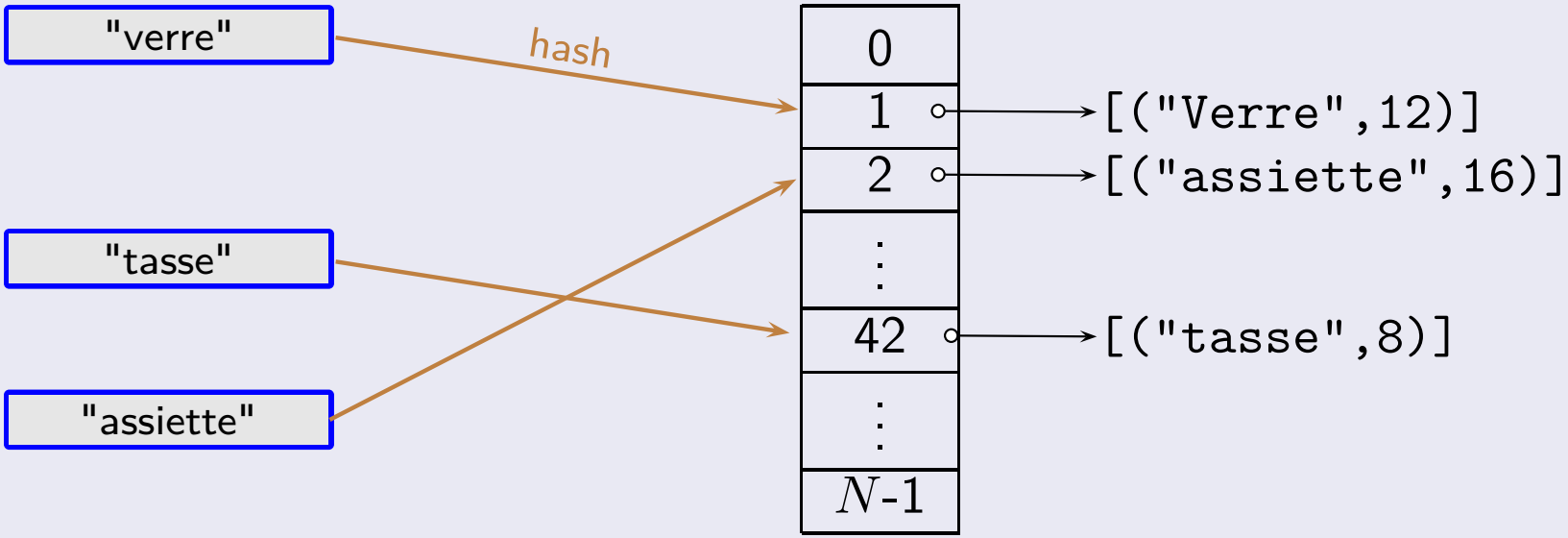
Visualisation

```
prix = { "verre":12 , "tasse" : 8, "assiette" : 16, "bol" : 10}
```



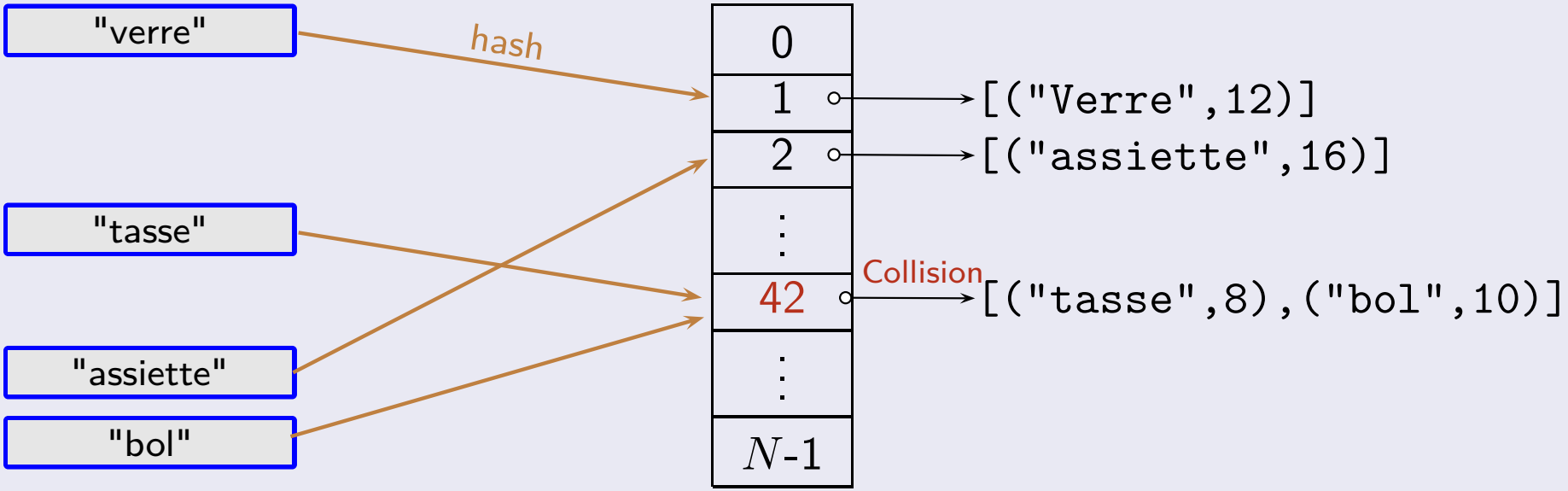
Visualisation

```
prix = { "verre":12 , "tasse" : 8, "assiette" : 16, "bol" : 10}
```



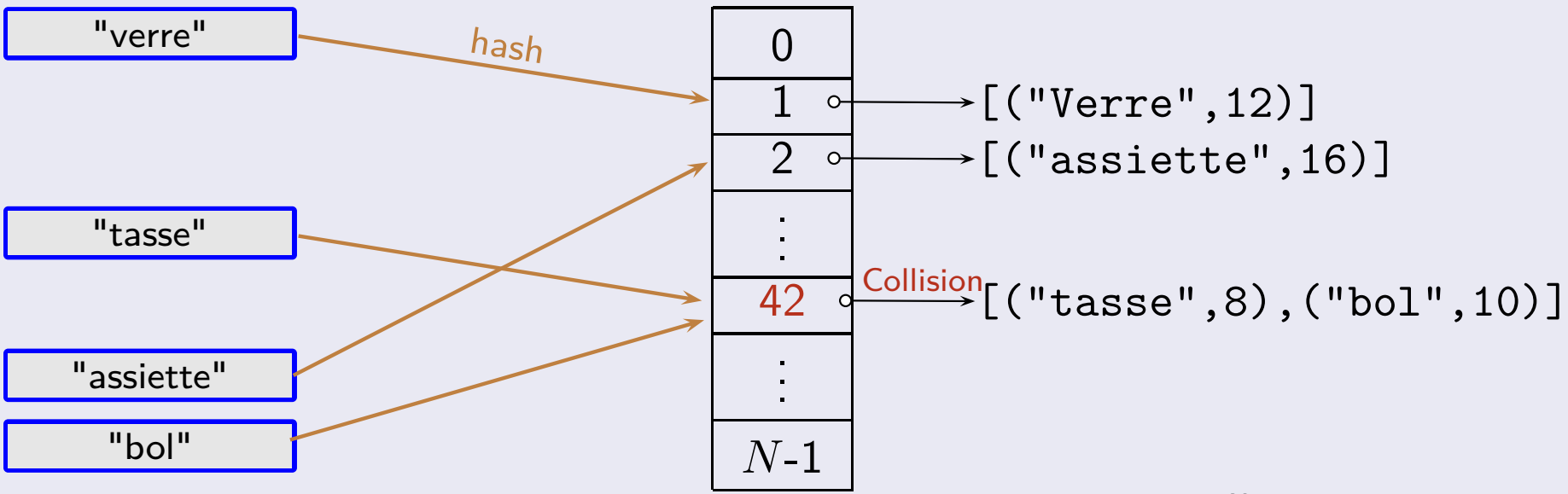
Visualisation d'une collision

```
prix = { "verre":12 , "tasse" : 8, "assiette" : 16, "bol" : 10}
```



Visualisation d'une collision

```
prix = { "verre":12 , "tasse" : 8, "assiette" : 16, "bol" : 10}
```



Pour rechercher si une clé est présente dans le dictionnaire il suffit de calculer son *hash* et de regarder à l'indice correspondant dans le tableau.

Exemple introductif

1. Ecrire une fonction récursive qui prend en argument un entier n et renvoie le $n$ième terme de la suite de Fibonacci défini par :

$$\begin{cases} f_0 &= 0, \\ f_1 &= 1, \\ f_n &= f_{n-1} + f_{n-2} \quad \text{pour tout} \quad n \geq 2. \end{cases}$$

Exemple introductif

1. Ecrire une fonction récursive qui prend en argument un entier n et renvoie le $n$ième terme de la suite de Fibonacci défini par :
$$\begin{cases} f_0 &= 0, \\ f_1 &= 1, \\ f_n &= f_{n-1} + f_{n-2} \quad \text{pour tout} \quad n \geq 2. \end{cases}$$
2. Tracer le graphe des appels récursifs de cette fonction pour $n = 5$

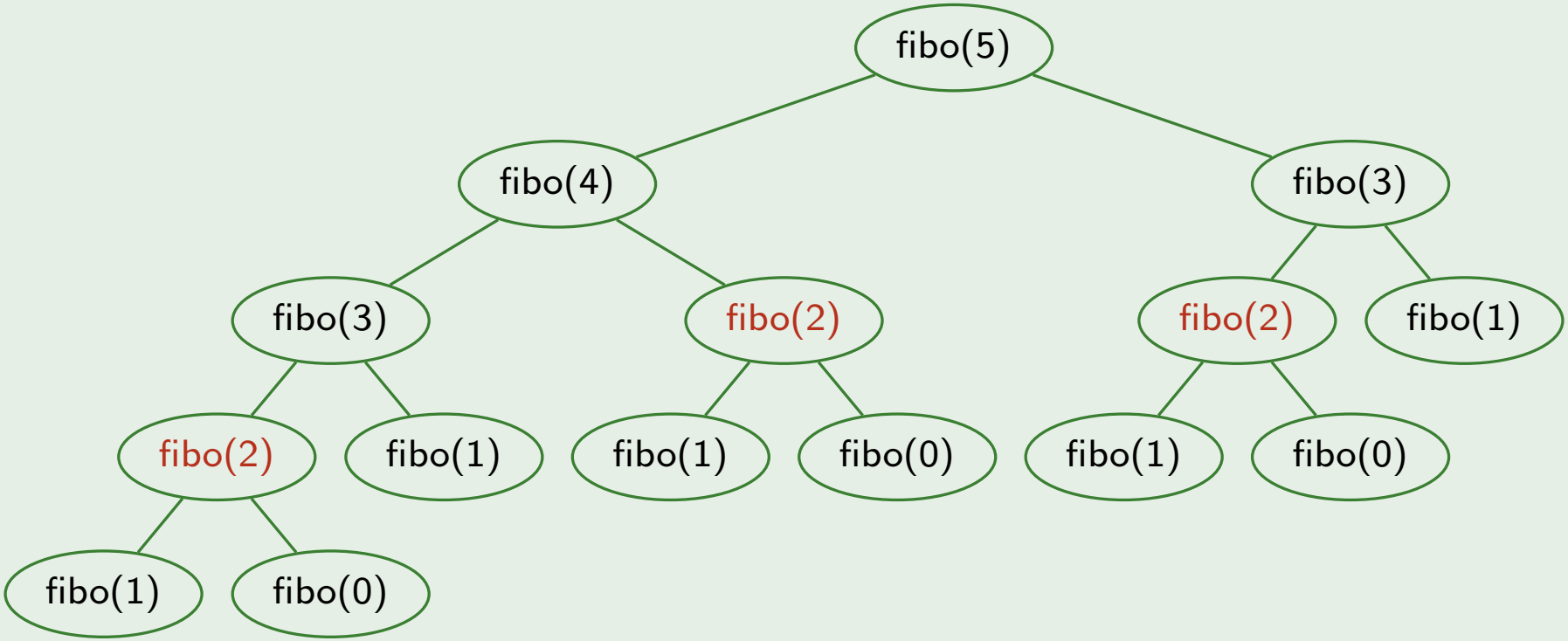
Exemple introductif

1. Ecrire une fonction récursive qui prend en argument un entier n et renvoie le $n$ième terme de la suite de Fibonacci défini par :
$$\begin{cases} f_0 &= 0, \\ f_1 &= 1, \\ f_n &= f_{n-1} + f_{n-2} \quad \text{pour tout} \quad n \geq 2. \end{cases}$$
2. Tracer le graphe des appels récursifs de cette fonction pour $n = 5$
3. Conclure

Correction question 1

```
1  def fibonacci(n):
2      assert n>=0
3      if n<2:
4          return 1
5      return fibonacci(n-1)+fibonacci(n-2)
```

Correction questions 2-3



On calcule à plusieurs reprises fibo(2).