Dictionnaires

Un dictionnaire associe à une **clé unique** une **valeur**
C'est une structure de données de **correspondance**, on parle aussi de tableau associatif
Par exemple dans le dictionnaire des variables gérées par Python , à un nom de variable est associée soit une valeur soit une référence, ainsi dans l'exemple suivant à la variable `quotient` est associée la valeur 3

```
>>> quotient = 3
>>> vars()
{'quotient': 3, '__loader__': <class '_frozen_importlib.BuiltinImporter'>, '__spec__': None, '
__package__': None, '__doc__': None, '__name__': '__main__', '__builtins__': <module 'builtins
' (built-in)>}
```

Un autre exemple :
Nous avons tous dans nos téléphones un dictionnaire où sont associés à des noms, des numéros de téléphone, par exemple :

```
contacts = { "Jean Pile": "0708091011",
 "Manu Militari": "0610090807",
 "Jacques Adi ": "0609080710"}
```

Le dictionnaire `contacts` ci-dessus a été défini en extension mais on aurait pu à partir d'un dictionnaire vide ajouter des éléments au fur et à mesure, par exemple

```
contacts = { }
contacts["Jean Pile"] = "0708091011"
contacts["Manu Militari"] = "0610090807"
contacts["Jacques Adi"] = "0609080710"
```

Il n'y a pas de structure d'ordre dans un dictionnaire d'ailleurs à l'affichage les éléments n'apparaissent pas forcément dans l'ordre d'insertion dans le dictionnaire. Ainsi par exemple

```
>>> contacts = {}
>>> contacts["Jean Pile"] = "0708091011"
>>> contacts["Manu Militari"] = "0610090807"
>>> contacts["Jacques Adi"] = "0609080710"
>>> contacts
{'Jean Pile': '0708091011', 'Jacques Adi': '0609080710', 'Manu Militari': '0610090807'}
```

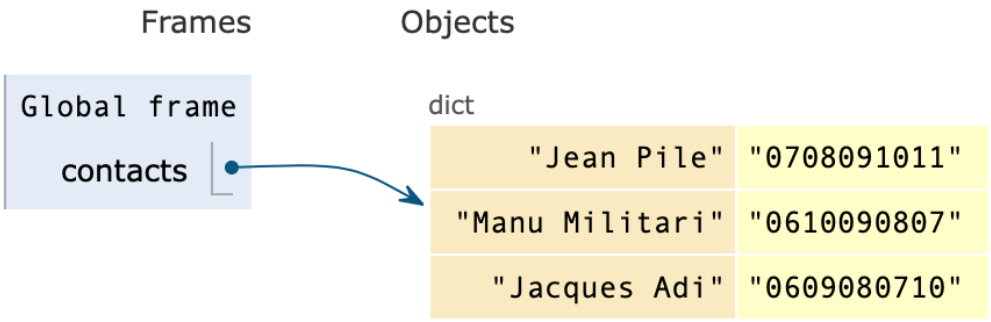
Quelles sont les principales opérations que l'on peut faire sur un dictionnaire ?

1. Se demander si une clé se trouve dans le dictionnaire

   ```
   >>>"Jean Pile" in contacts
   True
   ```

   ou

   ```
   >>>"Jean Pile" in contacts.keys()
   True
   ```



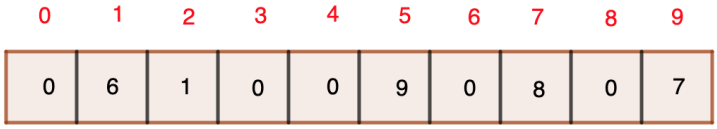
contacts est un **objet** et `keys()` est une fonction spécifique ou méthode de l'objet `contacts` et exercer une certaine action sur un objet par l'intermédiaire d'une méthode `f(...)` s'écrit `objet.f(...)`
La méthode `keys()` appliquée à l'objet `contacts` retourne les clés du dictionnaire `contacts` sous forme de liste à condition que l'on transforme l'objet `contacts.keys()` en liste par

```
l = list(contacts.keys())
```

2. Supprimer une clé et sa valeur

```
>>>del contacts["Jean Pile"]
```

3. Parcourir la liste des clés pour faire un traitement sur les valeurs associées, par exemple afficher les noms des numéros commençant par "06"
Une chaîne de caractères est aussi une liste Python, aussi chaque caractère est repéré par un indice ainsi par exemple dans la chaîne "0610090807", l'indice du premier caractère est 0, du second 1 etc...



Ainsi puisque `contacts[cle]` est une chaîne de caractères, rechercher un caractère particulier dans la chaîne se fait comme dans une liste par
`contacts[cle][i]`
où i est un entier compris entre 0 et `len(contacts[cle]) - 1`

```
for cle in contacts.keys():
    if contacts[cle][1] == "6":
        print(cle," a pour numéro de téléphone : ",
        contacts[cle])
```

4. Il existe aussi la méthode `values()` qui appliquée à l'objet `contacts` retourne les valeurs sous la forme d'un objet. On n' aurait pu afficher que les numéros de téléphone commençant par 06 **sans leur antécédents** c'est à dire les noms

```
for val in contacts.values():
    if val[1] == "6":
        print(val)
```

5. Il existe aussi la méthode `items()` qui appliquée à l'objet `contacts` retourne un objet contenant les tuples (clés,valeurs)

```
for couple in contacts.items():
    if couple[1][1] == "6":
        print(couple[0]," a pour numéro de téléphone : ",
        couple[1])
```

   Ici couple est un tuple dont le premier élément `couple[0]` est une clé et le deuxième élément `couple[1]` est une valeur de type chaîne de caractères. `couple[1][1]` est le deuxième caractère de cette chaîne

6. **Bien retenir que** `contacts.keys()`, `contacts.values()` **et** `contacts.items()` **ne sont pas des listes mais des objets, et que l'on peut parcourir ces objets comme on parcourt une liste**

7. Comme pour les listes on peut créer un dictionnaire en **compréhension**

   Par exemple on peut créer un tableau de valeurs pour la fonction affine f définie par $f(x) = 2*x+1$, pour les valeurs entières de 0 jusqu'à 10 compris en procédant ainsi

```
tableau_valeurs = {i : 2*i+1 for i in range(0,11)}
```

1 Exercices

Ex 1

Comment savoir le type des objets `contacts.keys()`, `contacts.values()` et `contacts.items()` ?

Ex 2

Le serveur DNS (Dynamic Name System) d'authentification confirme l'association d'un nom de domaine à une adresse IP (Internet Protocol)
On dispose du dictionnaire suivant :

```
dns = {"fr.wikipedia.org":"91.198.174.192",
"mathly.fr":"85.236.158.88",
"gallerianazionale.com":"94.130.217.148"}
```

1. Qu'affiche l'instruction `print(dns["mathly.fr"])` ?
2. Qu'affiche l'instruction `len(dns.keys())` ?

Ex 3

On dispose d'un dictionnaire `volcans` qui associe à des noms de volcans (de type str) des couples (type,date) où type est le type du volcan, le caractère "E" pour explosif ou "e" pour effusif et date la date de la dernière éruption connue.
Par exemple l'instruction `print(volcans["Piton de la Fournaise"])` a affiché `("e","22/12/2021")`
On dit que le Piton de la Fournaise est en cours d'éruption

1. Définir une fonction `volcans_effusifs(volcans)` qui à partir du dictionnaire `volcans` retourne la liste des volcans effusifs
2. Définir une fonction `volcans_en_cours_eruption(volcans)` qui à partir du dictionnaire `volcans` retourne la liste des volcans en cours d'éruption

Ex 4

A chaque nom de polygone régulier (type str) on lui associe le couple (type tuple) constitué du nombre de sommets (type int) et du nombre de diagonales (type int)
On a ainsi un dictionnaire `polygones`
Par exemple l'instruction `print(polygones["carré"])` a affiché `(4,2)`

1. On aimerait trouver une loi qui relie le nombre de sommets (ou de côtés) du polygone et le nombre de diagonales
   Pour cela construire une liste `nb_diagonales` à partir du dictionnaire `polygones` en parcourant les valeurs de ce dictionnaire telle que
   `nb_diagonales[n]` est le nombre de diagonales d'un polygone à `n` sommets et `nb_diagonales[n]` $= 0$ si $0 \leqslant n \leqslant 3$
   Définir une fonction `cree_liste(polygones)` qui crée la liste `nb_diagonales` à partir du dictionnaire `polygones`
2. Etant donnée une liste `liste1` on définit la liste des variations `liste2` telle que `liste2[n] = liste1[n] - liste1[n-1]` et `liste2[0] =0`
   Définir une fonction `cree_liste_variations(liste)` qui retourne la liste des variations de `liste`

3. Les valeurs de la liste des variations de la liste `nb_diagonales`
   [0, 0, 0, 0, 2, 3, 4, 5, 6, 7, 8,....]
   nous emmènent à penser que la loi qui relie le nombre de sommets (ou de côtés) du polygone et le nombre de diagonales est un trinôme $d(n) = an^2 + bn$ avec $d(3) = 0$ et $d(4) = 2$.
   Déterminer a et b en résolvant un système de deux équations en a et b (Indication : Ecrire en fonction de a et b $d(3) = 0$ et $d(4) = 2$
4. Vérifier que $d(n) = \dfrac{n(n-1)}{2} - n$
5. Utiliser cette loi pour engendrer le dictionnaire `polygones` par compréhension de la manière suivante :
   (a) Les clés seront des chaines de caractères engendrées suivant le format n-gone où n est le nombre de sommets
   (b) les valeurs correspondantes sont $d(n) = \dfrac{n(n-1)}{2} - n$

Ex 5

1. Ecrire une fonction `occurrences(tableau:list)` qui renvoie un dictionnaire où les clés sont les éléments du tableau et les valeurs, les occurrences des clés dans le tableau (le nombre de fois où apparaissent les éléments dans le tableau)
   Par exemple

```
>>> occurrences( ['chat','lapin','chat','python'])
{"lapin" : 1, "chat" : 2, "python" : 1}
```

2. Télécharger du cahier de textes le fichier `tour_du_monde.txt` contenant le roman de Jules Verne, Le tour du Monde en quatre-vingt jours.
   Observer le contenu du fichier avec un éditeur de texte.
3. La méthode `read()` permet de transformer le contenu du fichier en une seule chaîne de caractères. Aussi le code suivant permet de transformer le contenu du fichier texte en un tableau de chaînes de caractères dont la plupart sont les mots du texte.

```
with open('tour_du_monde.txt','r') as fichier:
    data = fichier.read().split()
```

   Observer le contenu du tableau.
   Il faudrait peut-être corriger certains éléments du tableau afin d'avoir le plus possible de mots du texte dans ce tableau. Proposer une solution.
4. Quel est le mot de sept lettres le plus fréquent dans le texte ?

Ex 6

Une expression littérale étant donnée par exemple sous cette forme '5 - 2' c'est à dire une chaîne de caractères représentant un nombre **puis un espace** puis un symbole parmi + - * / **puis un espace** puis à nouveau une chaîne de caractères représentant un nombre, il s'agit **d'évaluer** cette expression littérale c'est à dire de lui attribuer la valeur numérique correspondante, ici 3
On va utiliser la fonction `split()` qui appliquée à une chaîne de caractères contenant

```
>>> expression = '5 - 2'
>>> elt1, op, elt2 = expression.split(' ')
>>> type(expression.split(' '))
<class 'list'>
>>> print(elt1)
5
>>> type(elt1)
<class 'str'>
```

un séparateur ici l'espace, "coupe" cette chaîne selon le séparateur et les éléments sont des chaînes de caractères insérés dans une liste
On utilise un dictionnaire pour **associer** au symbole une **fonction** qui permet ensuite d'évaluer l'expression

```
def somme(x,y):
    return x + y

def difference(x,y):
    return x - y

def mult(x,y):
    return x*y

def div(x,y):
    if y == 0:
        return 'infini'
    else:
        return x/y

operateurs = {'+' : somme,
              '-' : difference,
              '*' : mult,
              '/' : div}

def evaluation(expression):
    elt1, op, elt2 = expression.split(' ')
    operateur = operateurs[op]
    return operateur(int(elt1),int(elt2))
#----------------main---------------------
expression = '6 + 7'
print(evaluation(expression))
```

Le nom d'une fonction est une **variable** associée à une **référence**, on voit cette association ci-dessous dans le dictionnaire `vars()`

On retrouve cette association dans le dictionnaire `operateurs` que l'on peut créer

```
>>> def somme(x,y):
        return x + y

>>> def difference(x,y):
        return x - y

>>> def mult(x,y):
        return x*y

>>> def div(x,y):
        if y == 0:
                return 'infini'
        else:
                return x/y


>>> vars()
{'mult': <function mult at 0x1129e3510>, '__package__': None, 'div': <function div at 0x1129e3598>, '__do
__': None, 'difference': <function difference at 0x101e4cbf8>, 'somme': <function somme at 0x10ffce510>,
__name__': '__main__', '__builtins__': <module 'builtins' (built-in)>, '__loader__': <class '_frozen_impo
tlib.BuiltinImporter'>, '__spec__': None}
```

dynamiquement en créant chaque association en autant d'instructions

```
>>> operateurs = {}
>>> operateurs['+'] = somme
>>> operateurs['-'] = difference
>>> print(operateurs)
{'+': <function somme at 0x10ffce510>, '-': <function difference at 0x101e4cbf8>
>>> operateurs['+'](7,5)
12
```

```
>>> operateurs['*'] = mult
>>> operateurs['/'] = div
>>> operateurs.keys()
dict_keys(['+', '/', '*', '-'])
>>> operateurs.values()
dict_values([<function somme at 0x10ffce510>, <function div at 0x1129e3598>, <function mult at 0x1129e351
>, <function difference at 0x101e4cbf8>])
>>> operateurs.items()
dict_items([('+', <function somme at 0x10ffce510>), ('/', <function div at 0x1129e3598>), ('*', <function
mult at 0x1129e3510>), ('-', <function difference at 0x101e4cbf8>)])
```