Cours 7 : mercredi 18 novembre

Dictionnaires en Python

Une structure très importante: le dictionnaire

Les dictionnaires sont des *collections non ordonnées d'objets* : il n'y aura pas d'index et l'ordre dans lequel les objets apparaissent n'aura pas vraiment d'importance , cet ordre pourra changer et ne sera pas choisi par l'utilisateur! Le dictionnaire est organisé avec des clés et des valeurs:

deux exemples:

```
dico={2:"deux",1:"un",3:"trois",4:"quatre",5:"cinq"}
stock={"crayons":43,"stylos bleus":23, "stylo rouge":
40,"gommes":12,"regles":25,"compas":10}
```

Les clés sont par exemples “crayons”, “compas” , et les valeurs respectives 43, 10....
Les dictionnaires sont représentés entourés par des accolades {} avec pour chaque terme une donnée sous la forme clé:valeur

Une clé est forcément unique (il ne peut pas y avoir deux valeurs différentes associées).

Une clé n'est pas forcément une chaîne ou un nombre; par contre cela ne peut pas être une donnée mutable ce qui interdit les listes
On peut par contre avoir un tuple comme clé:

```
plan={(0,0): 'gare', (10,100): 'lac', (50,50): 'boulangerie',
(45,25):'mairie'}
```

remarque: contrairement aux exemples donnés, toutes les clés ne sont pas obligatoirement de même type à l'intérieur d'un dictionnaire.

Un dictionnaire est de type dict.

premiers outils/ premiers exemples

La fonction len peut s'appliquer à un dictionnaire: elle renverra alors le nombre de clés ou de couples ce qui revient au même puisqu'une clé n'est présente qu'une fois.

```
>>> len(stock)
5
```

Etant donnée une clé on peut récupérer la valeur correspondante: mais il faut que la clé soit présente dans le dictionnaire (sinon message d'erreur)

```
>>> dico[3]
'trois'
>>> stock["gommes"]
12
>>> stock["feutre"]
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    stock["feutre"]
KeyError: 'feutre'
```

Etant donnée une clé on peut supprimer le couple clé: valeur correspondante (là aussi, il faut que la clé soit présente) avec del.
Il s'agit d'une modification physique.

```
>>> stock
{'regles': 25, 'crayons': 43, 'compas': 10, 'stylo rouge': 40, 'gommes': 12,
'stylos bleus': 23}
>>> del stock["gommes"]   # ne renvoie rien mais modifie stock
>>> stock
{'regles': 25, 'crayons': 43, 'compas': 10, 'stylo rouge': 40, 'stylos bleus': 23}
>>> del stock["feutres"]
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    del stock["feutres"]
KeyError: 'feutres'
```

Pour gérer un dictionnaire:

On peut rajouter un couple ou modifier une valeur avec une instruction du style

nom_du_dico[cle]=valeur

qui soit ajoutera soit modifiera un couple. Cette instruction modifie le dico (modification physique) et renvoie none.

```
>>> stock ["surligneurs"]=10  #là ,on ajoute une clé
>>> stock
{'regles': 25, 'crayons': 43, 'compas': 10, 'stylo rouge': 40, 'surligneurs': 10, 'stylos bleus': 23}

>>> stock["surligneurs"]=6 # là, on modifie la valeur
>>> stock
{'regles': 25, 'crayons': 43, 'compas': 10, 'stylo rouge': 40, 'surligneurs': 6, 'stylos bleus': 23}
```

Pour tester l'appartenance d'une clé au dictionnaire on peut utiliser **in:**

(remarque la méthode has_key présente en version2 ne l'est plus en version3)

```
>>> stock
{'regles': 25, 'crayons': 43, 'compas': 10, 'stylo rouge': 40, 'surligneurs': 10,
'stylos bleus': 23}

>>> “crayons” in stock
True
>>>  “gommes” in stock
False
```

Cela ne marche que pour les clés, pas pour les valeurs

```
>>> 25 in stock
False
```

Il ne peut y avoir qu'une seule valeur donc un seul couple pour une même clé, si on en écrit plusieurs à la main, c'est le dernier qui gagne! (avec un programme cela ne peut pas arriver)

```
dico={2:"deux",1:"un",3:"trois",4:"quatre",5:"cinq",3:"tros"}
>>> dico
{1: 'un', 2: 'deux', 3: 'tros', 4: 'quatre', 5: 'cinq'}
```

On a un dictionnaire vide:

```
>>> dico={}
```

L'ordre n'a pas d'importance pour un dictionnaire:

```
>>> stock2={"crayons":13,"stylos verts":3, "stylo rouge":
70,"gommes":2}
>>> stock3={"stylo rouge": 70,"gommes":2,"crayons":13,"stylos
verts":3}
>>> stock2==stock3
True
```

Attention: on ne peut pas additionner deux dicos avec + (ni les multiplier avec *)

```
>>> stock2*2
Traceback (most recent call last):
  File "<pyshell#23>", line 1, in <module>
    stock2*2
TypeError: unsupported operand type(s) for *: 'dict' and 'int'

>>> stock+stock2
Traceback (most recent call last):
  File "<pyshell#24>", line 1, in <module>
    stock+stock2
TypeError: unsupported operand type(s) for +: 'dict' and 'dict'
```

Nous verrons ultérieurement comment “fusionner” deux dictionnaires mais cela dépendra des cas!.

On ne peut pas non plus ajouter un couple clé:valeur avec +

```
>>> stock + {"gommes":4}
Traceback (most recent call last):
  File "<pyshell#80>", line 1, in <module>
    stock + {"gommes":4}
TypeError: unsupported operand type(s) for +: 'dict' and
'dict'
```

C'est d'ailleurs exactement le même problème

Remarque: Comme le dictionnaire n'est pas une séquence, il n'y a pas de “slice” ou d'équivalent à dic[3:6]: pas de “sous-dictionnaire “ extrait automatiquement (mais rien n'empêche d'écrire des fonctions qui renverraient des dictionnaires extraits) et **surtout pas de ième élément**

Quelle différence avec une liste du style:
stock=[['crayon', 13], ['stylo', 24], ['gommes', 14]]

Il faudra parcourir toute la liste éventuellement pour arriver au couple contenant un élément: on peut écrire le programme:

```
def cherche(truc, st):
    for [x,y] in st:
        if x==truc:
            return y
    return False

>>> cherche('gommes',stock)
14
>>> cherche('règles',stock)
False
```

L'interêt du dictionnaire est un accès direct à n'importe quelle clé pour en obtenir la valeur associée ou la modfier.

C'est une structure TRES EFFICACE

Comment faire des programmes avec les dictionnaires?

for s'applique très bien aux dictionnaires
(qui n'est pourtant pas une séquence)
En fait **for gère le parcours des clés .**

```
>>> stock
{'regles': 25, 'crayons': 43, 'compas': 10, 'stylo rouge': 40, 'gommes': 12, 'stylos bleus': 23}

>>> l=[x for x in stock]
>>> l
['regles', 'crayons', 'compas', 'stylo rouge', 'gommes', 'stylos bleus']

>>> for x in stock:
        print x
regles
crayons
compas
stylo rouge
gommes
stylos bleus
```

A retenir: le for travaille sur les clés

(ce qui peut être un moyen de les récupérer ou d'obtenir d'autres résultats....)

→ calculer combien il y a d'objets présents en tout dans le dictionnaire:

```
def total(dic):
   som=0
   for x in dic:   #ou for x in dic.keys()
      som=som+dic[x]
   return som
```

→ pour multiplier toutes les valeurs par un coefficient dans un dictionnaire: (modification physique)

```
def multi(dic,coeff):
   for x in dic:
      dic[x]=coeff*dic[x]  # ou dic[x]*=coeff
```

```
>>> multi(stock, 10)
>>> stock
{'regles': 250, 'crayons': 430, 'compas': 100, 'stylo rouge': 400, 'gommes': 120, 'stylos bleus': 230}
```

les méthodes des dictionnaires

```
>>> plan={(0,0): 'gare', (10,100): 'lac', (50,50): 'boulangerie', (45,25):'mairie'}
```

dico.keys() renvoie les clés présentes dans dico

```
>>> plan.keys()
dict_keys([(45, 25), (0, 0), (10, 100), (50, 50)])
```

dico.values() renvoie les valeurs présentes dans dico

```
>>> plan.values()
dict_values(['mairie', 'gare', 'lac', 'boulangerie'])
```

!! attention ordre quelconque!!!

Ce n'est pas une liste qui est renvoyée mais une séquence qui se parcourt comme une liste (et qu'on peut transformer en liste)

```
>>> l=plan.keys()
>>> l
dict_keys([(45, 25), (10, 100), (50, 50)])
>>> list(l)
[(45, 25), (10, 100), (50, 50)]
```

dico.clear() vide complètement le dictionnaire:

```
>>> plan
{(45, 25): 'mairie', (10, 100): 'lac', (50, 50): 'boulangerie'}
>>> plan.clear()
>>> plan
{}
```

dico.get(cl, val) renvoie la valeur correspondante à cl dans le dico si la clé est présente et val sinon, val vaut none par défaut.
Là il n'y a pas d'erreur contrairement à dico[cl]

```
>>> plan
{(45, 25): 'mairie', (10, 100): 'lac', (50, 50): 'boulangerie'}
>>> plan.get((0,0))
>>> plan.get((0,0),0)
0
>>> plan.get((0,0)," pas de donnée")
' pas de donnée'
>>> plan.get((50,50))
'boulangerie'
```

dico.items() renvoie les tuples (clé, valeur) correspondant au dico:

```
>>> plan.items()
dict_items([((45, 25), 'mairie'), ((10, 100), 'lac'), ((50, 50), 'boulangerie')])
```

dico.update(autredico) met à jour dico en tenant compte du dictionnaire autredico :
attention ce n'est pas une somme, c'est une mise à jour: c'est les valeurs de autredico qui seront prises en compte s'il y a des clés communes.... il n'y a pas d'addition

```
>>> dico={'un':'one','deux':'too','trois':'three'}
>>> dico2={'cinq':'five','deux':'two','trois':'three'}

>>> dico.update(dico2)

>>> dico  # dico a été mis à jour
{'un': 'one', 'trois': 'three', 'deux': 'two', 'cinq': 'five'}

>>> dico2  # pas modifié
{'trois': 'three', 'deux': 'two', 'cinq': 'five'}
```

Les dictionnaires sont des données mutables

Attention: les dictionnaires sont des données mutables (comme les listes) donc il faudra faire éventellement une "vraie" copie (puisque utiliser = ne peut pas convenir).

```
>>> d=dico
>>> d['un']='ein'
>>> d
{'un': 'ein', 'trois': 'tres', 'deux': 'two', 'cinq': 'five'}
>>> dico
{'un': 'ein', 'trois': 'tres', 'deux': 'two', 'cinq': 'five'}
```

On a pour cela la méthode dico.copy()

Dictionnaires et compréhensions

On peut utiliser le for en compréhension pour fabriquer un dictionnaire:

Si on a une liste:

```
>>> l=['france','espagne','italie']
>>> dic={x:0 for x in l}
>>> dic
{'france': 0, 'italie': 0, 'espagne': 0}
```

Autre exemple:

```
>>> couples=[["a",23],["b",0],["c", 234]]
>>> d={x:v for x, v in couples if v!=0}
>>> d
{'a': 23, 'c': 234}
```

→ Une addition de dictionnaires:

On veut “fusionner” deux dictionnaires en ***additionnant*** les articles commun.
Rappel: + ne convient pas au type dict
et update ne donne pas les résultats voulus

```
def add_dico(d1,d2): # on modifie d1
   for x in d2:  # parcours des clés de d2
      if x in d1:     # test pour que d1[x] ne plante pas
         d1[x]=d1[x]+d2[x]
      else:
         d1[x]=d2[x]


stock={'o1':12,'o2':12,'o3':11,'o4':12,'o11':10,'o8':12,'o7':1,'o6':12,'o5':8}
stock2={'o11': 1, 'o8': 2, 'o6': 2,  'o4': 5, 'o3': 7,'o14':5}
>>>add_dico(stock,stock2)
>>> stock
{'o14': 5, 'o11': 11, 'o8': 14, 'o7': 1, 'o6': 14, 'o5': 8, 'o4': 17, 'o3': 18, 'o2': 12, 'o1':
12}
```

variante

On peut aussi créer un nouveau dictionnaire pour ne modifier aucun des dictionnaires de départ:

```
def add_dico(d1,d2): # on crée un nouveau dictionnaire
   d3=d1.copy()  # ici pas besoin de deepcopy()
   for x in d2:  # parcours des clés de d2
      if x in d1:      # test pour que d1[x] ne plante pas
         d3[x]=d1[x]+d2[x]
      else:
         d3[x]=d2[x]
   return d3  # indispensable
```

→ Travail sur des mots ou des textes

On veut compter le nombre d'occurences de chacune des lettres présentes dans un texte. On pourrait bien sûr compter les a puis les b ..... mais cela demanderait plusieurs passages.... ou alors avoir 26 variables ou une liste à 26 variables..... mais c'est lourd..... sans compter les caractères spéciaux.....

Avec un dictionnaire on peut le faire en un seul passage de façon assez simple.

```
montexte="Il existe un grand nombre de langages de programmation, chacun avec ses avantages et ses inconvenients. L'ideal serait d'en utiliser plusieurs. "

def compte(tex):
    dic={}
    for x in tex:
        if x in dico:  # si x est déjà dans le dico on rajoute 1
                dic[x]=1+ n
        else:
                dic[x]=1 # si x n'y est pas on l'ajoute
    return dic


>>> compte(montexte)
{' ': 20, '"': 2, ',': 1, '.': 1, 'I': 1, 'L': 1, 'a': 12, 'c': 4, 'b': 1, 'e': 18, 'd': 5, 'g': 5, 'i': 9, 'h': 1, 'm': 3, 'l': 5, 'o': 4, 'n': 12, 'p': 2, 's': 12, 'r': 7, 'u': 5, 't': 7, 'v': 3, 'x': 1}
```

On pourrait ensuite retravailler pour supprimer les espaces, les apostrophes, uniformiser l et L.....

Avec dico.get()

```
montexte="Il existe un grand nombre de langages de programmation, chacun avec ses avantages et ses inconvenients. L'ideal serait d'en utiliser plusieurs. "

def compte(tex):
    dic={}
    for x in tex:
        n = dic.get(x,0) # renvoie 0 si x pas dans  dico
        dic[x]=1+ n
    return dic
# l'emploi de dic.get évite de faire un test sur
# la présence de la clé x dans le dictionnaire

>>> compte(montexte)
{' ': 20, '"': 2, ',': 1, '.': 1, 'I': 1, 'L': 1, 'a': 12, 'c': 4, 'b': 1, 'e': 18, 'd': 5, 'g': 5, 'i': 9, 'h': 1, 'm': 3,
'l': 5, 'o': 4, 'n': 12, 'p': 2, 's': 12, 'r': 7, 'u': 5, 't': 7, 'v': 3, 'x': 1}
```

On pourrait ensuite retravailler pour supprimer les espaces, les apostrophes, uniformiser l et L.....

→ des listes de dictionnaires

Il peut être très intéressant de faire des listes de dictionnaires: par exemple:

```
l1={"auteur":"victor hugo ","num":"0005670078","disponible":1,"titre":
"les miserables", "cote":"R MIS HUG","genre":"classique"}
l2={"auteur":"agatha christie", "titre" :"les dix petits negres",
"genre":"policier","num":"0006780078","disponible":0,"cote":"RP MIS
HUG" }
l3={"auteur":"dan brown", "titre" :"da vinci code",
"num":"0005340078","disponible":0, "cote":"RP DAV BRO ", "genre":
"thriller"}

bibli=[l1,l2,l3]
```

On peut alors travailler sur un livre (= un dico) ou sur une bibliothèque (liste de dicos)

fonctions “génériques”

```
def affichage(l,valeur_cle):
    for x in l:
        print x[valeur_cle]

>>> affichage (bibli,"auteur")
victor hugo
agatha christie
dan brown


>>> affichage(bibli,"titre")
les miserables
les dix petits negres
da vinci code
```

Recherches avec des compréhensions

exemples: chercher les livres disponibles, les romans, ...

```
def dispo(bib):
    return [liv for liv in bib if liv["disponible"]==1]

def roman(bib):
   return [liv for liv in bib if liv["cote"][0:2]=="R "]

def recherche_nom(bib,nom):
    return [liv for liv in bib if nom in liv["auteur"]]

def recherche_titre(bib,titre):
    return [liv for liv in bib if titre in liv["titre"]]

def recherche_origine(bib,orig):
   return [liv for liv in bib if orig== liv["num"][-4:]]
```

Version générique mais qui ne fait pas tout

```
def recherche(bibli, cle, valeur):
    return [liv for liv in bibli if liv[cle]==valeur]
```

Utilisation de update

On peut modéliser un emprunt grace à update qui va modifier les valeurs qu'on veut mettre à jour:

```
emprunt={“dispo”:False, “retour”:”4/04/2008”, “emprunteur”:”2345121”}
l1.update(emprunt)
>>> bibli
[{'retour': '4/04/2008', 'num': '0005670078', 'disponible': 1, 'dispo': False,
'emprunteur': '345121', 'cote': 'R MIS HUG', 'auteur': 'victor hugo ', 'genre':
'classique', 'titre': 'les miserables'}, {'num': '0006780078', 'disponible': 0, 'cote':
'RP MIS HUG', 'auteur': 'agatha christie', 'genre': 'policier', 'titre': 'les dix petits
negres'}, {'num': '0005340078', 'disponible': 0, 'cote': 'RP DAV BRO ', 'auteur':
'dan brown', 'genre': 'thriller', 'titre': 'da vinci code'}]
>>> dispo(bibli)
[{'retour': '4/04/2008', 'num': '0005670078', 'disponible': 1, 'dispo': False,
'emprunteur': '345121', 'cote': 'R MIS HUG', 'auteur': 'victor hugo ', 'genre':
'classique', 'titre': 'les miserables'}]
```

donner une “mémoire des calculs” à Python

Un dictionnaire peut servir de mémoire pour conserver des résultats de calculs au fur et à mesure pour ne pas les exécuter plusieurs fois

exemple: calcul efficace des termes de la suite de Fibonacci

La suite de fibonacci est définie par
fo=f1=1
$f_n = f_{n-1} + f_{n-2}$

D’où une idée du programme

```
def fibo(n):
    if n ==0 or n==1:
        return 1
    else:
        return fibo(n-2)+fibo(n-1)
```

Problème?

Ce calcul n'est pas efficace….. à partir de n=25/30 cela rame….

Pourquoi? c'est facile à comprendre si on voit ce que fait réellement la machine:

fibo(6):

fibo(5) + fibo(4)

fibo (4) + fibo(3) fibo(3) + fibo (2)

fibo(3) + fibo(2) fibo(2)+fibo(1) fibo(2)+fibo(1) fibo(1) + fibo(0)

fibo(2)+fibo(1) fibo(1) + fibo(0) fibo(1) + fibo(0) fibo(1) + fibo(0)

fibo(1) + fibo(0)

Beaucoup de calculs sont faits plusieurs fois.....

Il faut donc soit faire mémoriser les valeurs à la machine (avec les dictionnaires- voir plus loin) soit programmer différemment

donner une “mémoire des calculs” à Python

Un dictionnaire peut servir de mémoire pour conserver des résultats de calculs au fur et à mesure pour ne pas les exécuter plusieurs fois

```
fib={0:1,1:1}  # sert d'initialisation  veut dire fibo(0)=1 et fibo(1)=1

def fibo(n):
    if n in fib:
        return fib[n]
    else:
        x=fibo(n-2)+fibo(n-1)
        fib[n]=x  # on rajoute une valeur au dico dès le calcul
        return x
```

Cela devient très efficace !