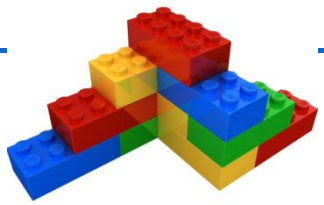
Les types construits

Résumé :

Présentation plus approfondie des types construits de python avec quelques exemples d'emploi utile. En effet apprendre un langage de programmation informatique pour lui-même n'est pas le plus motivant, découvrons donc au travers de la mise en œuvre de ces types comment ils peuvent nous permettre de résoudre un certain nombre de problèmes par l'implémentation d'algorithmes.

Ces travaux débouchent sur l'utilisation de types imbriqués, … autrement dit de types dans les types ! Dans les exercices qui sont décrits dans un autre document.

Les types présentés :

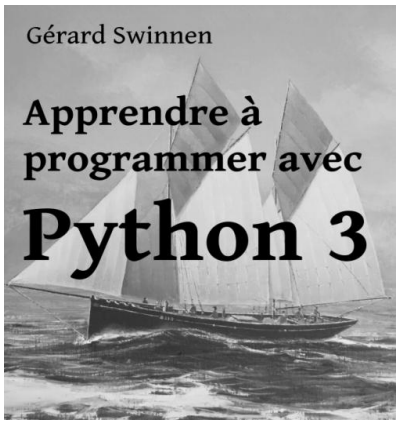
- les listes à plusieurs dimensions.
- les tuples ou n-uplets
- les dictionnaires.
- les ensembles.

Sommaire :

1 Ressources utiles pour aborder les exemples

1.1 A mettre dans sa bibliothèque

- L'excellent travail de Gérard Swinnen :



apprendre_python3_swinnen_2015.pdf
apprendre_python3_swinnen_2010.pdf

- Le site https://www.w3schools.com/python/default.asp

w3schools.com

- La documentation Python https://docs.python.org/fr/3/tutorial/datastructures.html

Python Software Foundation (US)

Python

- Autre site complet : http://www.python-simple.com/

1.2 Les différents types construits disponibles en Python

Ces types sont tous itérables. Résumons leurs caractéristiques dans un tableau :

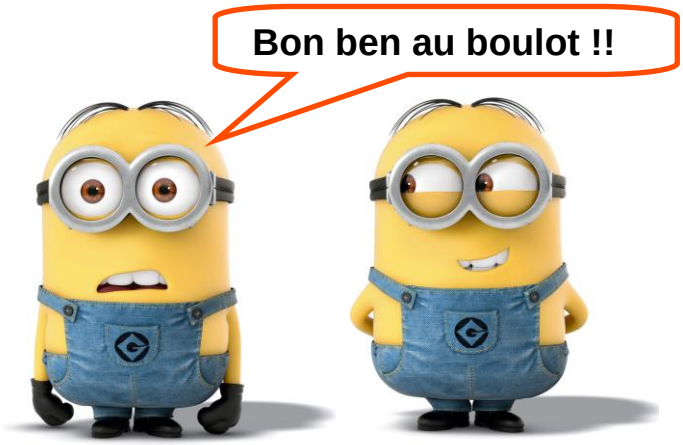
Type de donnée	Nom python	Exemple	Ordonnée	Modifiable	Permet plusieurs fois le même élément
listes	**list**	thislist = ["apple", "banana", "cherry"]	**oui**	**oui**	**oui**
n-uplet	**tuple**	thistuple = ("apple", "banana", "cherry")	**oui**	**non**	**oui**
ensemble	**set**	thisset = {"apple", "banana", "cherry"}	**non**	**non***	**non**
dictionnaire	**dictionnary**	thisdict = { "brand": "Ford", "model": "Mustang", "year": 1964 }	**non**	**oui**	**non**

* Pour la modification des ensembles voir la description dans le texte.

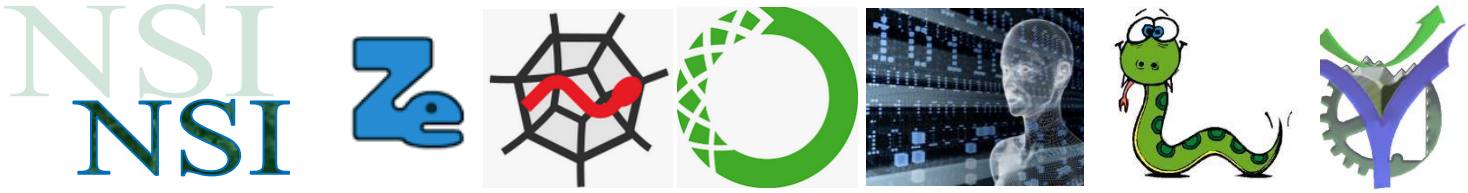
1.3 Emploi de ces différents types de données

L'objectif en soi dans ce cours n'est pas d'aboutir à une connaissance exhaustive sur toutes les subtilités des différents types de données disponibles, d'autant plus que ces types peuvent se combiner. Nous verront des exemples en exercices.

En fait il faut acquérir également le sens de l'utilisation de ces types pour pouvoir choisir dans nos programmes celui qui est le plus adapté pour contribuer à résoudre un problème donné. Nous verrons également cela en exercice.



Acquérir cette compétence ne peut se faire qu'avec une pratique intensive.



2 Fiche de résumé sur les listes

2.1 Création d'une liste

```
thislist = ["apple", "banana", "cherry"]
>>> thislist
['apple', 'banana', 'cherry']
```

Création d'une liste 'vide' :

```
thislist = []
```

2.2 Accès à un élément d'une liste, index, range x : y

Les éléments dans une liste sont indexés. La valeur de l'index débute à 0, les index sont croissants à partir de la valeur vers la droite de la liste :

```
>>> thislist[0]
'apple'
```

On peut également utiliser des index négatifs qui sont décomptés à partir de la fin de la liste :

```
>>> thislist[-1]
'cherry'
```

On peut également donner un intervalle (range) de valeurs :

```
thislist = ["apple", "banana", "cherry",
"orange", "kiwi", "melon", "mango"]
>>> thislist[2:5]
['cherry', 'orange', 'kiwi']
```

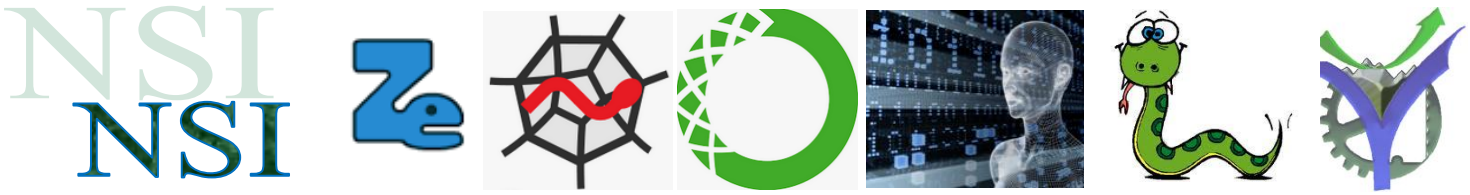
Notez que la borne supérieure de l'intervalle est exclue.

On peut sous entendre le début ou bien la fin de la liste :

```
>>> thislist[:5]
['apple', 'banana', 'cherry', 'orange', 'kiwi']

>>> thislist[2:]
['cherry', 'orange', 'kiwi', 'melon', 'mango']
```

On peut définir un intervalle à partir de la fin de la liste :

```
>>> thislist[-4:-1]
['orange', 'kiwi', 'melon']
```



2.3 Modification d'un élément

Il suffit de donner l'index de l'élément à modifier :

```
>>> thislist[1]='pomme'

>>> thislist
['apple', 'pomme', 'cherry', 'orange', 'kiwi',
'melon', 'mango']
```

2.4 Balayer tous les éléments d'une liste avec une boucle

C'est un des points fort de Python, rien à définir il suffit d'une ligne de code pour itérer tous les éléments d'une liste (de toute structure de données itérables) :

```
thislist = ["apple", "banana", "cherry",
"orange", "kiwi", "melon", "mango"]

for x in thislist:
  print(x)
```

```
apple
banana
cherry
orange
kiwi
melon
mango
```

2.5 Tester la présence d'un élément dans une liste

Le mécanisme est similaire au précédent, il suffit d'utiliser l'instruction in :

```
if "apple" in thislist:
  print("Yes, 'apple' is in the fruits list")
```

2.6 Connaitre la longueur d'une liste

La fonction len() s'utilise :

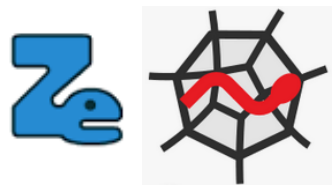
```
>>> len(thislist)
7
```

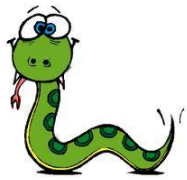
2.7 Ajouter un élément à la fin de la liste

On utilise la méthode append() :

```
thislist = ["apple", "banana", "cherry"]

thislist.append('tomate')

>>> thislist
['apple', 'banana', 'cherry', 'tomate']
```

⇦






Au passage nous vérifions que nous pouvons avoir plusieurs éléments de mêmes valeurs dans une liste :

```
thislist = ["apple", "banana", "cherry"]

thislist.append('tomate')

thislist.append('tomate')

>>> thislist
['apple', 'banana', 'cherry', 'tomate', 'tomate']
```

2.8 Ajouter un élément à une position déterminée

```
thislist = ["apple", "banana", "cherry"]

thislist.insert(1, "orange")

>>> thislist
['apple', 'orange', 'banana', 'cherry']
```

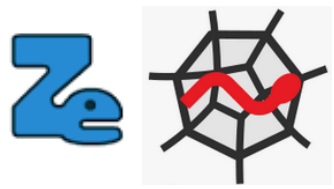
2.9 Enlever un élément

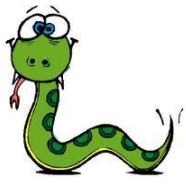
Pour enlever un élément nous pouvons utiliser la méthode remove en spécifiant l'élément à supprimer. La méthode enlève alors le premier élément rencontré depuis l'origine de la liste :

```
thislist = ['apple', 'pomme', 'cherry', 'orange',
'orange', 'melon', 'mango']

thislist.remove('orange')

>>> thislist
['apple', 'pomme', 'cherry', 'orange', 'melon', 'mango']
```

Nous pouvons également accéder à un élément avec son index avec pop. Si aucune valeur n'est précisée c'est la dernière valeur qui est supprimée, exemple pour aboutir au même résultat que précédemment :

```
thislist = ['apple', 'pomme', 'cherry', 'orange',
'orange', 'melon', 'mango']

thislist.pop(3)
```






Troisième méthode utiliser la fonction del sur un élément :

```
thislist = ['apple', 'pomme', 'cherry', 'orange',
'orange', 'melon', 'mango']

del thislist[3]
>>> thislist
['apple', 'pomme', 'cherry', 'orange', 'melon', 'mango']
```

ou sur la liste entière, attention dans ce cas l'objet est supprimé de l'espace des objets il n'existe plus :

```
thislist = ['apple', 'pomme', 'cherry', 'orange',
'orange', 'melon', 'mango']

del thislist

>>> thislist
Traceback (most recent call last):
  File "<console>", line 1, in <module>
NameError: name 'thislist' is not defined
```

2.10 Pour effacer le contenu de la liste

Utiliser la méthode clear :

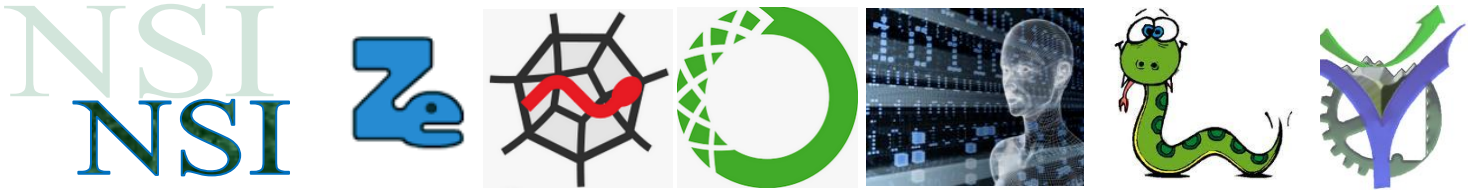
```
thislist = ['apple', 'pomme', 'cherry', 'orange',
'orange', 'melon', 'mango']

thislist.clear()

>>> thislist
[]
```

2.11 Recopier une liste

Attention on ne peut pas utiliser l'affectation. En effet dans ce cas on ne créé qu'une nouvelle référence sur la liste déjà existante en mémoire. Pour créer une nouvelle liste, superficiellement indépendante[1] de la première il faut utiliser la méthode .copy() :

```
thislist = ['apple', 'pomme', 'cherry', 'orange',
'orange', 'melon', 'mango']

copylist = thislist.copy()
```

[1] Voir paragraphe suivant Shallow Copy et Deep Copy



```
>>> copylist
['apple', 'pomme', 'cherry', 'orange', 'orange', 'melon',
'mango']
```

Une autre solution avec la fonction list :

```
thislist = ['apple', 'pomme', 'cherry', 'orange',
'orange', 'melon', 'mango']

copylist = list(thislist)


>>> copylist
['apple', 'pomme', 'cherry', 'orange', 'orange', 'melon',
'mango']
```

Ou avec un slice 'vide' :

```
>>> thislist = ['apple', 'pomme', 'cherry', 'orange', 'mango']

>>> copylist = thislist[:]

>>> copylist
['apple', 'pomme', 'cherry', 'orange', 'mango']
```

2.12 Shallow Copy et Deep Copy[2]

Attention cependant les copies de listes réalisées avec les indications du paragraphe précédent font une copie superficielle, ou shallow copy, de la liste d'origine. Cela signifie que seul le premier niveau est recopié de manière indépendante. Si les listes pointent vers des objets eux-mêmes modifiables à plus d'un seul niveau comme des listes de listes par exemple l'indépendance totale n'est pas assurée. La copie n'est pas complètement indépendante.

Pour assurer une indépendance totale des deux listes copiées quelque soit le contenu de la liste d'origine il faut procéder à une opération de copie profonde appelée deepcopy du module copy :

```
# ESSAIS DE DEEP COPY
import copy

old_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
new_list = copy.deepcopy(old_list)
```

[2] Voir par exemple ici : https://www.programiz.com/python-programming/shallow-deep-copy

2.13 Concaténer deux listes

En utilisant l'opérateur + :

```
liste1 = [1,2,3,4,5,6]
liste2 = ['a','b','c']
liste3 = liste1 + liste2
```

```
>>> liste3
[1, 2, 3, 4, 5, 6, 'a', 'b', 'c']
```

En utilisant la méthode .extend() on peut ajouter une liste à une liste déjà existante et la modifiée en conséquence :

```
liste1 = [1,2,3,4,5,6]
liste2 = ['a','b','c']
liste1.extend(liste2)
```

```
>>> liste1
[1, 2, 3, 4, 5, 6, 'a', 'b', 'c']
```

2.14 Renverser une liste

```
thislist = [1, 2, 3, 4, 5, 6, 'a', 'b', 'c']

thislist.reverse()
```

```
>>> thislist
['c', 'b', 'a', 6, 5, 4, 3, 2, 1]
```

⇦

2.15 La compréhension de liste

Le meilleur pour la fin, remplir automatiquement une liste avec une compréhension de liste, trois exemples :

Le principe de base, utiliser une variable interne x et faire varier cette variable sur un intervalle :

```
thislist = [x**2 for x in range(10)]

>>> thislist
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

On peut y ajouter une condition. On balaye toujours x depuis la valeur 0 jusqu'à la valeur 9 incluse, mais on ne conserve que les valeurs paires :

```
thislist = [x**2 for x in range(10) if x % 2 == 0]

>>> thislist
[0, 4, 16, 36, 64]
```

Ou les valeurs impaires :

```
thislist = [x**2 for x in range(10) if x % 2 == 1]

>>> thislist
[1, 9, 25, 49, 81]
```

On peut imbriquer plusieurs boucles :

```
thislist = [(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]

>>> thislist
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

Beaucoup d'autres possibilités voir la documentation officielle de Python :

https://docs.python.org/fr/3/tutorial/datastructures.html

3 Fiche de résumé sur les tuples

Les tuples représentent en Python une collection d'objets non modifiables et ordonnés. Les tuples sont appelés également n-uplet dans la littérature.

> En mathématiques, si **n** est un entier naturel, alors un **n-uplet** ou **n**-uple est une collection ordonnée de **n** objets, appelés « composantes » ou « éléments » ou « termes » du **n-uplet**.

3.1 Création d'un tuple

```
thistuple = ("apple", "banana", "cherry")
```

```
>>> thistuple
('apple', 'banana', 'cherry')
```

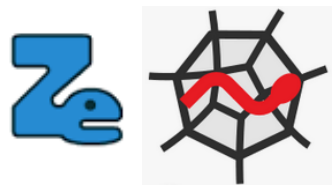
Un tuple n'est pas modifiable essayons quand même :

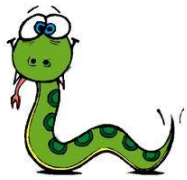
```
>>> thistuple[1]='pomme'
Traceback (most recent call last):
  File "<console>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

3.2 Accès à un élément d'un tuple

```
>>> thistuple[1]
'banana'
```

Nous retrouvons les possibilités offertes par les index positifs, négatifs, définit avec un intervalle de la même manière que celle décrite pour les listes. Nous ne le reprenons donc pas ici.






3.3 Modification d'un élément

Cela n'est pas directement possible avec les tuples. Néanmoins il y a une solution de contournement :

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)
```

```
>>> x
('apple', 'kiwi', 'cherry')
```

Nous observons au passage comment transformer un tuple en liste et une liste en tuple.

3.4 Itérer sur les éléments d'un tuple

Nous retrouvons là encore les techniques classiques permises par le langage Python :

```
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
  print(x)
```

```
apple
banana
cherry
```

3.5 Tester l'existence d'un élément particulier

```
thistuple = ("apple", "banana", "cherry")
if "apple" in thistuple:
  print("Ok, bon appétit")
```

3.6 Le nombre d'éléments d'un tuple

```
>>> len(thistuple)
3
```

3.7 Création d'un tuple avec un seul élément

Pour éviter les confusions de types avec les parenthèses qui peuvent encadrer une valeur pour différentes raisons sans en modifier le type on ajoute une virgule comme ceci :

```
thistuple = ("apple",)
```

3.8 Regrouper plusieurs tuples

C'est possible en utilisant l'opérateur + :

```
tuple1 = ("a", "b" , "c")

tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2
```

```
>>> tuple3
('a', 'b', 'c', 1, 2, 3)
```

⇦

4 Fiche de résumé sur les dictionnaires

Les dictionnaires sont des conteneurs qui sont donc itérables. Ils sont mutables : on peut ajouter, supprimer, modifier leur contenu.

Les dictionnaires ne sont pas des séquences : on ne peut pas accéder à leur contenu en donnant un indice. Les dictionnaires ne sont pas ordonnés : c'est l'interpréteur python qui gère l'organisation interne du dictionnaire. Même si avec les versions récentes de python les éléments sont rangés dans l'ordre de leurs créations.

Les éléments contenus dans les dictionnaires sont référencés par une clé. Cette clé est unique pour chaque élément, elle ne doit donc pas être mutable. En conséquence les éléments constituants une clé peuvent être : un nombre, une chaine de caractères, un tuple, un tuple de tuple etc. … par contre les listes ne sont pas admises car elles sont mutables (modifiables).

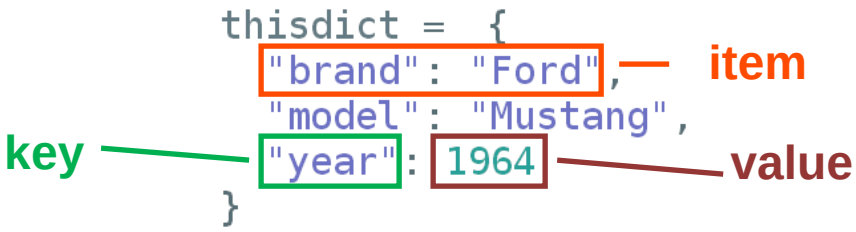
Voir https://www.w3schools.com/python/python_dictionaries.asp w3schools.com

4.1 Création d'un dictionnaire

Création d'un dictionnaire vide :

```
mondict = {}
```

Création d'un dictionnaire avec trois éléments : brand, model, year



On remarque les noms associés aux différentes parties d'un dictionnaire.

4.2 Accès aux différents éléments

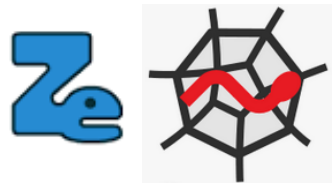
Sur la base du dictionnaire précédent on peut y accéder via les clés, les valeurs ou bien les items :

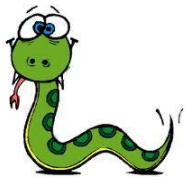
```
>>> thisdict.keys()
dict_keys(['year', 'model', 'brand'])

>>> thisdict.values()
dict_values([1964, 'Mustang', 'Ford'])

>>> thisdict.items()
dict_items([('year', 1964), ('model', 'Mustang'),
('brand', 'Ford')])
```

Accès à une valeur via la clé :

```
>>> thisdict["model"]
'Mustang'

>>> thisdict.get("model")
'Mustang'
```

⇦






4.3 Combien y a-t-il de valeurs ?

```
>>> len(thisdict)
3
```

4.4 Ajouter une valeur

```
>>> thisdict['power']=125

>>> thisdict
{'year': 1964, 'model': 'Mustang', 'brand': 'Ford', 'power': 125}
```

4.5 Modifier une valeur

```
>>> thisdict['power']=250

>>> thisdict
{'year': 1964, 'model': 'Mustang', 'brand': 'Ford', 'power': 250}
```

4.6 Tester l'existence d'un élément

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
if "model" in thisdict:
  print("Yes, 'model' is one of the keys in the thisdict dictionary")
```

4.7 Supprimer des valeurs

Avec leur clé :

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
```

```
>>> thisdict
{'year': 1964, 'brand': 'Ford'}
```

4.8 Supprimer complètement un dictionnaire

```
del thisdict["model"]
```

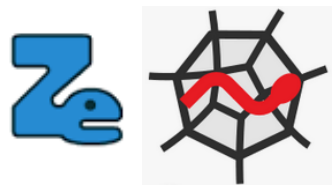
On peut supprimer complètement tout le dictionnaire en une seule commande :

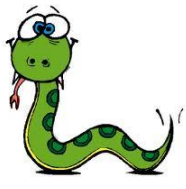
```
del thisdict
```

Il n'existe plus en mémoire.

```
>>> thisdict
Traceback (most recent call last):
  File "<console>", line 1, in <module>
NameError: name 'thisdict' is not defined
```

⇦






4.9 On peut simplement vouloir vider le dictionnaire :

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

thisdict.clear()
```

```
>>> thisdict
{}
```

4.10 Lister un dictionnaire avec une boucle

Par exemple imprimer toutes les clés :

```
print('Les clés du dictionnaire')
for x in thisdict:
    print(x)
```

```
Les clés du dictionnaire
year
model
brand
```

Ou bien imprimer toutes les valeurs :

```
print('Les valeurs du dictionnaire')
for x in thisdict:
  print(thisdict[x])
```

```
Les valeurs du dictionnaire
1964
Mustang
Ford
```

```
for x in thisdict.values():
  print(x)
```

Autre possibilité :

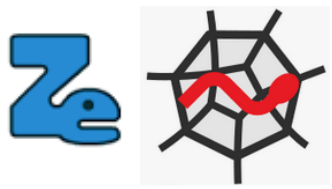
```
for x in thisdict.values():
```

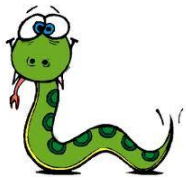
Ou bien lister tous les items :

```
print('Lister les items')
for key, value in thisdict.items():
    print ("Clé : %s, valeur : %s." % (key, value))
```

```
Lister les items
Clé : year, valeur : 1964.
Clé : model, valeur : Mustang.
Clé : brand, valeur : Ford.
```

⇦






4.11 Copie de dictionnaires

Attention au mauvais réflexe pour la copie de dictionnaires

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

thisdictfaussecopie = thisdict
```

```
>>> thisdictfaussecopie
{'year': 1964, 'model': 'Mustang', 'brand': 'Ford'}
```

Le signe = créé simplement une nouvelle référence vers le dictionnaire thisdict avec un nouveau nom thisdictfaussecopie. Si on modifie l'une l'autre est automatiquement modifiée car les deux pointent vers le même objet en mémoire, démonstration :

```
thisdict['year'] = 2020
```

```
>>> thisdictfaussecopie
{'year': 2020, 'model': 'Mustang', 'brand': 'Ford'}
```

Pour faire une vrai copie, qui créée un nouvel objet en mémoire indépendant du premier, il faut utiliser la méthode .copy() :

```
thisdictvraiecopie = thisdict.copy()

thisdict['year'] = 2020
```

```
>>> thisdictvraiecopie
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}

>>> thisdict
{'model': 'Mustang', 'brand': 'Ford', 'year': 2020}
```

Maintenant thisdict et thisdictvraiecopie sont bien indépendants.

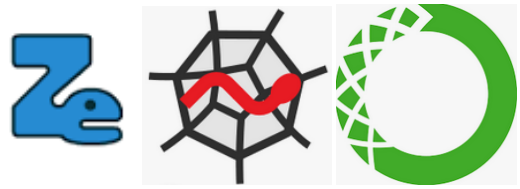
4.12 Dictionnaire en compréhension

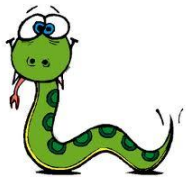
La compréhension de dictionnaires est possible il suffit de préciser la paire clé-valeur :

```
dict = { k : k*2 for k in range (2, 8) }
```

```
>>> dict
{2: 4, 3: 6, 4: 8, 5: 10, 6: 12, 7: 14}
```

⇦





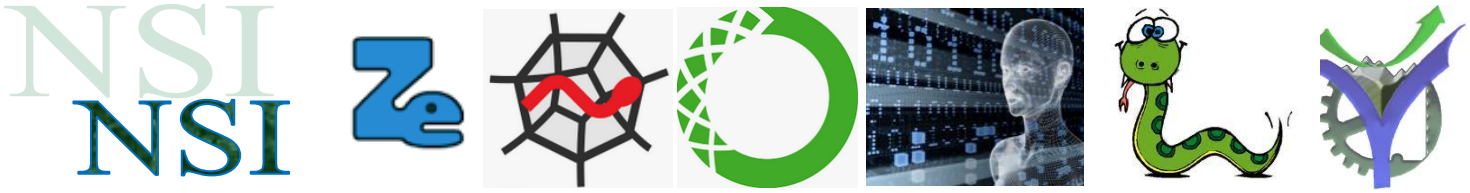

4.13 Dictionnaire de dictionnaires

Un dictionnaire peut contenir un ensemble hétérogène de clés-valeurs et également un autre dictionnaire exemple :

```
child1 = { "name" : "Emil", "year" : 2004 }
child2 = { "name" : "Tobias","year" : 2007 }
child3 = { "name" : "Linus", "year" : 2011 }

myfamily = {
  "child1" : child1,
  "child2" : child2,
  "child3" : child3
}
```

L'accès aux éléments :

```
>>> myfamily['child1']
{'year': 2004, 'name': 'Emil'}

>>> myfamily['child1']['name']
'Emil'
```



5 Les ensembles

Un ensemble est une collection d'objets ou d'éléments. Une relation d'appartenance définit si un élément appartient ou non à un ensemble. Dans un ensemble chaque élément est unique, il n'y a pas de doublons ni de relation d'ordre.

En python l'objet ensemble existe et est défini par l'instruction set :

5.1 Création d'un ensemble en python

```
thisset = {"apple", "banana", "cherry"}
```

```
>>> thisset
{'cherry', 'banana', 'apple'}
```

Ou bien par le constructeur set :

```
thisset = set(("apple", "banana", "cherry"))
```

5.2 Ajout d'un élément dans un ensemble

```
thisset.add("orange")
```

```
>>> thisset
{'orange', 'cherry', 'banana', 'apple'}
```

Il n'y a pas d'accumulation d'un même élément dans un ensemble démonstration :

```
thisset = {"apple", "banana", "cherry"}

thisset.add("orange")

thisset.add("apple")
```

```
>>> thisset
{'orange', 'cherry', 'banana', 'apple'}
```

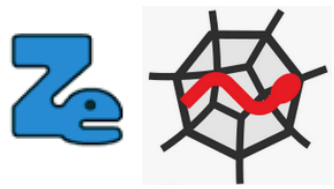
L'élément 'apple' n'apparait pas deux fois !

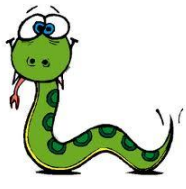
5.3 Ajout de plusieurs éléments

```
thisset = {"apple", "banana", "cherry"}

thisset.update(["orange", "mango", "grapes"])

>>> thisset
{'mango', 'orange', 'banana', 'grapes', 'apple', 'cherry'}
```

5.4 Combien y a t-il d'éléments dans un ensemble ?

```
>>> thisset
{'mango', 'orange', 'banana', 'grapes', 'apple', 'cherry'}

>>> len(thisset)
6
```






5.5 Enlever un élément d'un ensemble

```
thisset = {"apple", "banana", "cherry"}

thisset.remove("banana")

thisset.discard("apple")
```

```
>>> thisset
{'cherry'}
```

5.6 Supprimer complètement un ensemble

```
thisset = {"apple", "banana", "cherry"}

del thisset
```

```
>>> thisset
Traceback (most recent call last):
  File "<console>", line 1, in <module>
NameError: name 'thisset' is not defined
```

5.7 Ou bien vider un ensemble de tous ses éléments

```
thisset = {"apple", "banana", "cherry"}

thisset.clear()
```

```
>>> thisset
set()
```

5.8 Deux moyens de réunir deux ensembles

```
set1 = {"a", "b" , "c"}

set2 = {1, 2, 3}

set3 = set1.union(set2)
```

```
>>> set3
{1, 2, 3, 'b', 'c', 'a'}
```

L'ensemble set3 est un nouvel ensemble indépendant de set1 et set2.

Il y a une autre méthode qui ne crée pas de nouvel ensemble avec .update() :

```
set1 = {"a", "b" , "c"}

set2 = {1, 2, 3}

set1.update(set2)
```

```
>>> set1
{1, 2, 'a', 3, 'b', 'c'}
```

Rappel : il n'y a pas de doublons ni de relation d'ordre dans les ensembles.

5.9 Création d'un ensemble vide

On ne peut pas créer un ensemble vide directement car il y a une ambigüité avec les dictionnaires :

```
>>> ensemble_vide={}

>>> ensemble_vide
{}

>>> type(ensemble_vide)
<class 'dict'>
```

Il faut procéder comme ci-dessous :

```
>>> ensemble_vide = set()

>>> ensemble_vide
set()

>>> ensemble_vide.add('orange')

>>> ensemble_vide
{'orange'}

>>>
```