Cours 2: Structures des données

Nguyễn Kim Thắng

kimthang.nguyen@univ-evry.fr

Mails: [L3INF]

bureau 209, IBGBI
IBISC, Univ. Evry, University Paris-Saclay

Chaînes de caractères

Méthodes utiles

```
greetings = "Bonjour L3ASR"

greetings[4]                # => 'o'
"jour" in greetings         # => True
len(greetings)              # => 13
```

Méthodes utiles

```
greetings = "Bonjour L3ASR"

greetings[4]                # => 'o'
"jour" in greetings         # => True
len(greetings)              # => 13
```

```
greetings.find("ASR")          # => 10 (-1 si pas trouvé)
greetings.replace("jour", "ne journée “)
          # => 'Bonne journée L3ASR'
greetings.startswith('Bon')  # => True
greetings.endwith("ASR")     # => True
greetings.isalpha()          # => False
```

Méthodes utiles

```
greetings = "Bonjour L3ASR"

greeting.lower()    # => "bonjour l3asr"
greeting.upper()    # => "BONJOUR L3ASR"
greeting.title()    # => "Bonjour L3Asr”
greeting.strip()    # => "Bonjour L3ASR”
```

Méthode format

```
# les accolades pour réserver les places
'{} belle {}'.format("une", "journée")
# => une belle journée

# affecter les valeurs aux positions
'Les {0} apprennent {1}, {0}'.format("L3ASR", 'Python')
# => L3ASR apprennent Python, L3ASR

"{nom} enseigne le cours {cours}".format(nom = "Je", cours = "Python")
# => Je enseigne le cours Python

# Les valeurs sont converties en string
"{} au carré est {}".format(3, 3**2)
# => 3 au carré est 9
```

Méthode format

```
# format des chiffre style de C
"{:05.2f}".format(3.14159)  # => '03.14'

'{:4}'.format("abc")        # => ‘abc '
'{:*^11}'.format("L3ASR")   # => '***L3ASR***'
```

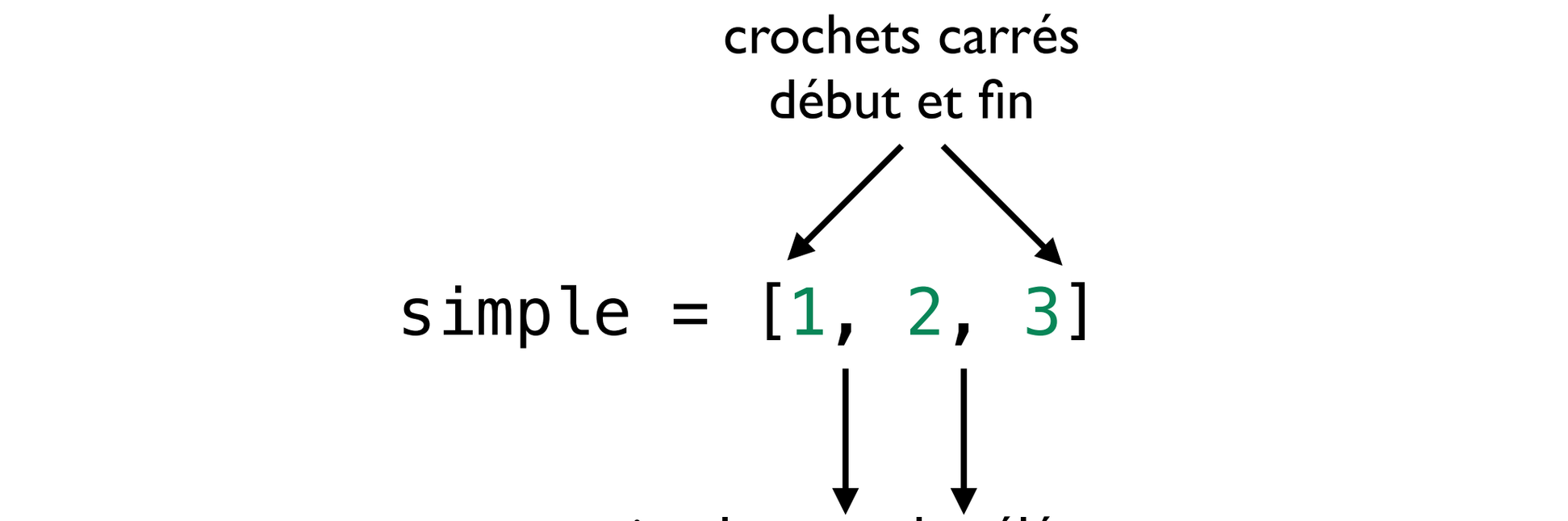
voir https://pyformat.info pour plus de détail

Listes

Qu’est-ce une liste

Une séquence d’éléments
finie, ordonnée et modifiable

Qu’est-ce une liste




```
simple_aussi = [‘a’, ‘b’, ‘c’]
```

Listes

```
# créer une liste
empty = []
lettres = ['a','b','c']
nombres = [10, 20 , 30]

# listes contiennet différentes types
mixed = [1, 2, "a", "b"]
```

```
# ajouter les éléments à la fin d'une liste
nombres.append(40)

# l’accès avec les indices
nombres[0]      # => 10
nombres[-1]     # => 40
nombres[1:-1]   # => [20, 30, 40]
```

Listes imbriquées

```
"""
on peut mettre ce qu'on veut
dans une liste
"""


lettres = ['a','b','c']
nombres = [10, 20 , 30]

x = [lettres, nombres]
x             # => [['a', 'b', 'c'], [10, 20, 30, 40]]
x[0]          # => ['a', 'b', 'c']
x[0][2]       # => 'c'
```

Listes récursives

```
"""
on peut mettre ce qu'on veut
dans une liste
"""

x = [1, 2]
x.append(x)
x           # => [1, 2, [...]]
x is x[2]   # => True
```

Méthodes

```
len([])                     # => 0
len("a")                    # => 1
len([1, 2, "abcd"])         # => 3

"ab" in "abcd"              # => True
"abd" in "abcd"             # => False
2 in [1, 2, "ab"]           # => True
"b" in [1, 2, "ab"]         # => False

list("L3ASR")               # => ['L', '3', 'A', 'S', 'R']
list(range(3,6))            # => [3, 4, 5]
```

List et String

```
list("Abc")     # => ['L', 'b', 'c']

# partitioner par un délimiteur
"Je veux faire L3Info".split()
# => ['Je', 'veux', 'faire', 'L3Info']

"1-23-456".split(sep='-')
# => ['1', '23', '456']

# "join" crée un chaine d'une liste
', '.join(["abc", "123", "xyz"])
# => "abc, 123, xyz"
```

Méthodes

```
# compter le nombre d'occurrences
# dans une liste
ma_liste.count(valeur)

# ajouter un élément dans une liste
ma_liste.append(elt)

# ajouter une autre liste
ma_liste.extend(autre_liste)
```

Méthodes

```
# compter le nombre d'occurrences
# dans une liste
ma_liste.count(valeur)

# ajouter un élément dans une liste
ma_liste.append(elt)

# ajouter une autre liste
ma_liste.extend(autre_liste)
```

```
# insérer un élément dans une position
ma_liste.insert(indice, elt)

# trier une liste
ma_liste.sort(key=None, reverse = False)

# inverser une liste
ma_liste.reverse()
```

Méthodes

```
# compter le nombre d'occurrences
# dans une liste
ma_liste.count(valeur)

# ajouter un élément dans une liste
ma_liste.append(elt)

# ajouter une autre liste
ma_liste.extend(autre_liste)
```

```
# insérer un élément dans une position
ma_liste.insert(indice, elt)


# trier une liste
ma_liste.sort(key=None, reverse = False)

# inverser une liste
ma_liste.reverse()
```

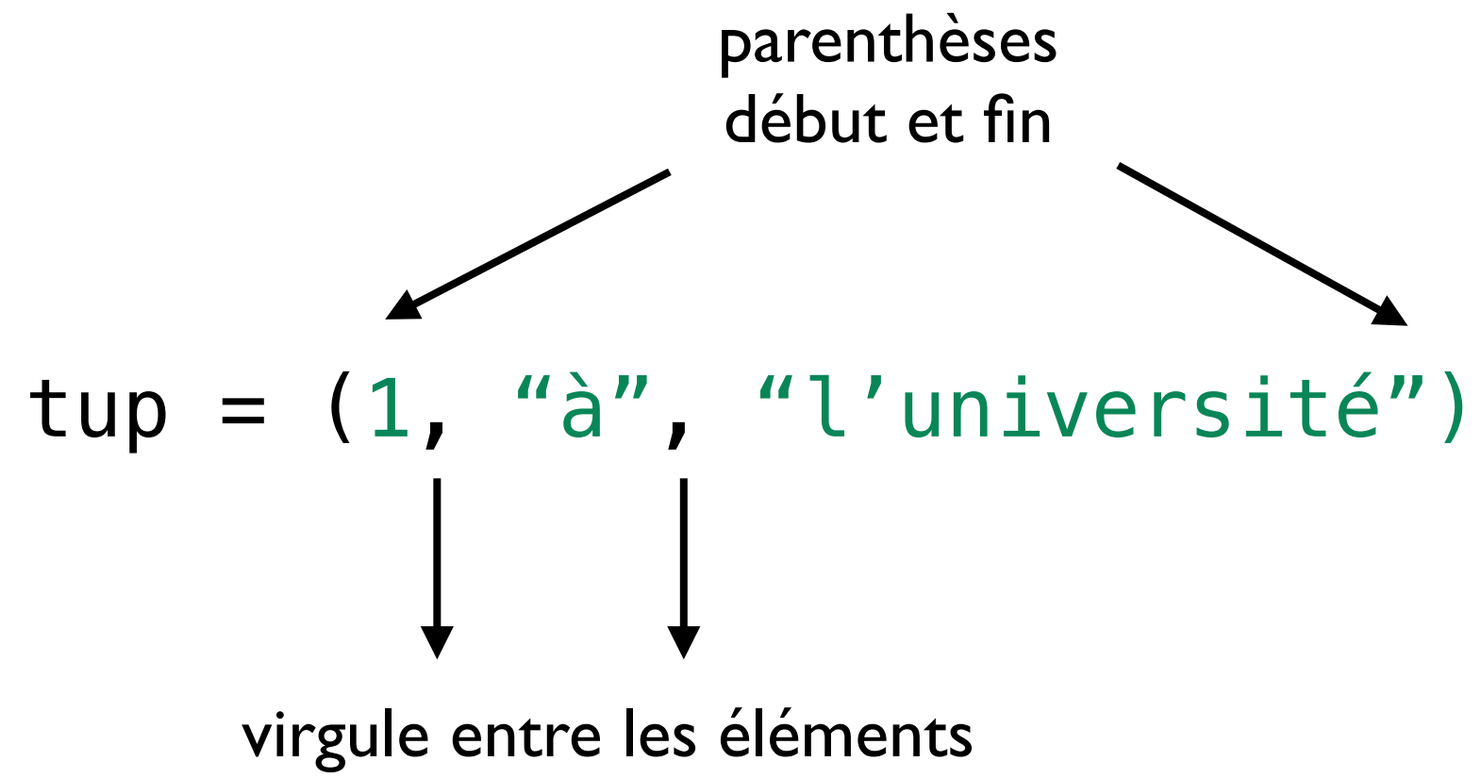
```
# modifier plusieurs éléments dans une liste
del ma_liste[debut:fin:pas]

# supprimer un élément dans une liste et retourner le
ma_liste.pop()      # => supprimer et retourner le dernier elt
ma_liste.pop(i)     # => supprimer et retourner ma_liste[i]

# supprimer un élément par valeur
ma_liste.remove(valeur)
```

Tuples

Qu’est-ce un tuple

Une séquence d’éléments
finie, ordonnée et non-modifiable

Qu’est-ce un tuple



Mais… pourquoi?

- Nous avons déjà des listes, pourquoi a-t-on besoin des tuples?

  - Stocker les données hétérogènes d'un objet
    (penser au struct ou sql objets)

  - Renforcer la non-modification

  - Bloquer les séquences pour hachage

Travailler avec les tuples

```
# vous pouvez pas modifier les éléments

objet = (1, 2, "trois")
objet[0]                    # => 1
objet[0] = "un"             # => TypeError

# ... mais autres opérations marchent normallement
len(objet)                  # => 3
objet[:-1]                  # => [1, 2]
"trois" in objet            # => True
```

Travailler avec les tuples

```
# vous pouvez pas modifier les éléments

objet = (1, 2, "trois")
objet[0]                    # => 1
objet[0] = "un"             # => TypeError

# ... mais autres opérations marchent normallement
len(objet)                  # => 3
objet[:-1]                  # => [1, 2]
"trois" in objet            # => True
```

```
# emballage et déballage

pre_chiffres = 1 , "deux", "trois"
print(pre_chiffres)                    # => (1, 'deux', 'trois')
type(pre_chiffres)                     # => tuple

# nous pouvons déballer
x, y, z = pre_chiffres
x                                 # => 1
y                                 # => 'deux'
z                                 # => 'trois'
```

Echange des valeurs

```
a, b  = b, a
```

```
a = 1
b = 2

# échanger les valeurs de a et b
a, b = b, a

a    # => 2
b    # => 1
```

Echange des valeurs

```
a, b  = b, a
```

```
a = 1
b = 2

# échanger les valeurs de a et b
a, b = b, a
```

```
a    # => 2
b    # => 1
```

- Question: afficher les 10 nombres Fibonacci

Echange des valeurs

```
a, b  = b, a
```

```
a = 1
b = 2

# échanger les valeurs de a et b
a, b = b, a

a   # => 2
b   # => 1
```

- Question: afficher les 10 nombres Fibonacci

```
a, b = 0, 1

for i in range(10)
    print(a)
    a, b = b, a + b
```

Emballage et déballage en fonctions

```
def rapid(a,b,c)
    return a + b - c

seq = (1, 2, 3)
rapid(*seq)      # => 0
```


*: déballage le tuple
exécute rapid(1, 2, 3)

Un astuce en déballage

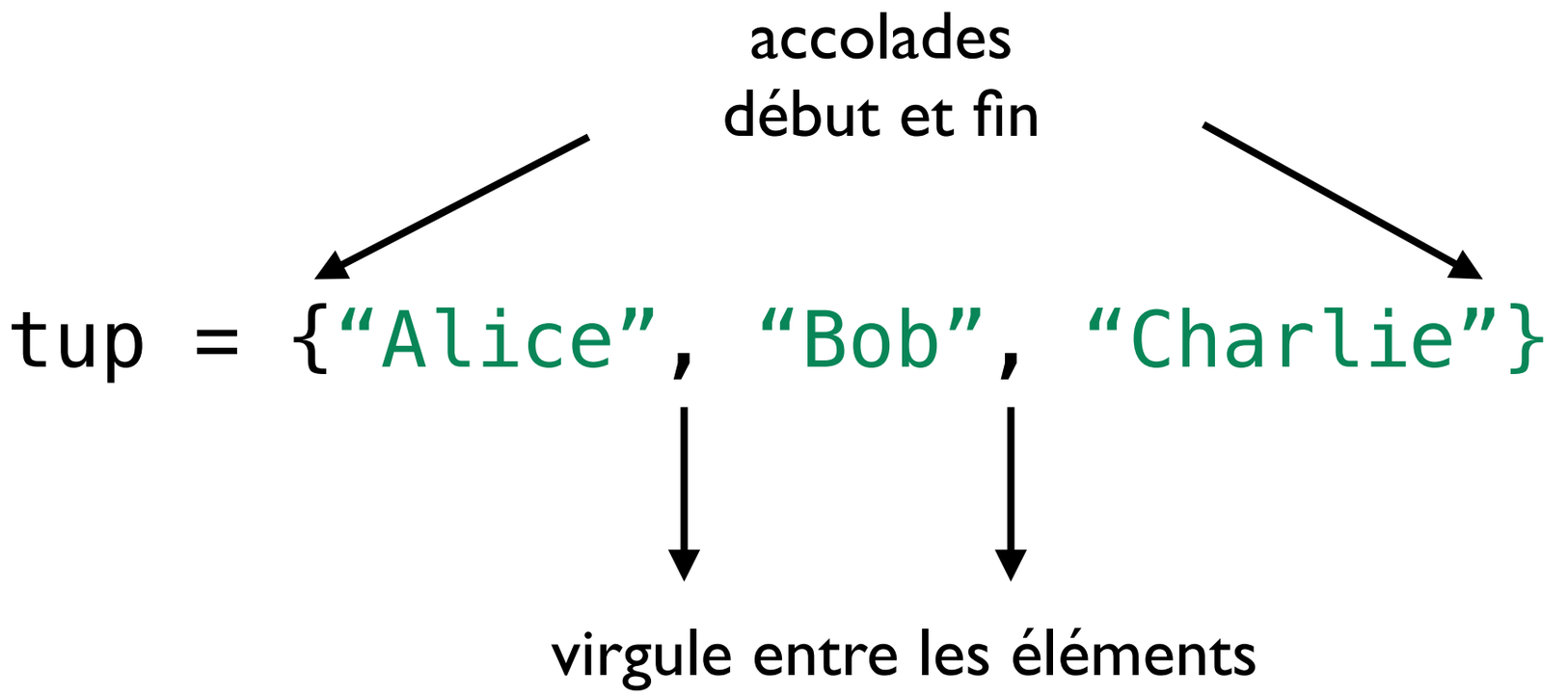
```
enumerate(["Alice", "Bob", "Charlie"])
# => ((0, 'Alice'), (1, 'Bob'), (2, 'Charlie'))

for i, nom in enumerate(["Alice", "Bob", "Charlie"])
    print(i, nom)

# 0 Alice
# 1 Bob
# 2 Charlie
```

Ensembles

Qu’est-ce un ensemble

Une collection des éléments distincts
et non-ordonnés

Qu’est-ce un ensemble



Mais… pourquoi?

- Nous avons déjà des listes, tuples, pourquoi a-t-on besoin des ensembles?
  - Vérification rapide de l'occurence
  - Elimination les doublons
  - Operations simplifiées (intersection, union, etc)

Travailler avec les ensembles

```
empty_set = set()
set_from_list = set([1, 3, 2, 3])   # => {1, 3, 2}

len(set_from_list)      # => 3
1 in set_from_list      # => True
4 in set_from_list      # => False
```

Travailler avec les ensembles

```
empty_set = set()
set_from_list = set([1, 3, 2, 3])   # => {1, 3, 2}

len(set_from_list)      # => 3
1 in set_from_list      # => True
4 in set_from_list      # => False
```

```
a = set('mississippi')  # => {'m', 'i', 's', 'p'}
a.add('asr')            # => {'m', 'i', 's', 'p', 'asr'}
a.remove('i')           # => KeyError si 'i' n'est pas dans a
a.discard('i')          # => comme remove mais pas erreur

a.pop()                 # => 'm' ou 'i' ou 's' ou 'p' ou 'asr'
a.clear()               # => supprimer tout
```

Travailler avec les ensembles

```
a = set('abccdaeb')     # => {'a', 'b', 'c', 'd', 'e'}
b = set('cdeffedg')     # => {'c', 'd', 'e', 'f', 'g'}

# Difference
a - b                   # => {'a', 'b'}
# Union
a | b                   # => {'a', 'b', 'c', 'd', 'e', 'f', 'g'}
# Intersection
a & b                   # => {'c', 'd', 'e'}
# Difference symétrique
a ^ b                   # => {'a', 'b', 'f', 'g'}
```

Travailler avec les ensembles

```
a = set('abccdaeb')     # => {'a', 'b', 'c', 'd', 'e'}
b = set('cdeffedg')     # => {'c', 'd', 'e', 'f', 'g'}

# Difference
a - b                   # => {'a', 'b'}
# Union
a | b                   # => {'a', 'b', 'c', 'd', 'e', 'f', 'g'}
# Intersection
a & b                   # => {'c', 'd', 'e'}
# Difference symétrique
a ^ b                   # => {'a', 'b', 'f', 'g'}


a < b                   # sous-ensemble strict
a <= b                  # sous-ensemble
a > b                   # supper-ensemble strict
a >= b                  # supper-ensemble
```

Un exemple

```
mot_long = 'L3ASRcilsMiage'

def faire_quoi(mot):
    for lettre in mot:
        if lettre not in mot_long:
            return False
    return True
```

- Le programme fait quoi?

Un exemple

```
mot_long = 'L3ASRcilsMiage'

def faire_quoi(mot):
    for lettre in mot:
        if lettre not in mot_long:
            return False
    return True
```

- Le programme fait quoi?
- Une simplification

```
mot_set = set('L3ASRcilsMiage')

def faire_quoi(mot):
    return set(mot) <= mot_set
```

Objets

Objets

Tout est un objet

```
isinstance(1, object)                   # => True
isinstance("Alice", object)             # => True
isinstance([1,2,"Bob"], object)         # => True
isinstance(None, object)                # => True

isinstance(str, object)                 # => True
isinstance(object, object)              # => True
```

Objets

Tout est un objet

```
isinstance(1, object)                   # => True
isinstance("Alice", object)             # => True
isinstance([1,2,"Bob"], object)         # => True
isinstance(None, object)                # => True

isinstance(str, object)                 # => True
isinstance(object, object)              # => True
```

Chaque objet a une identité, un type et une valeur

Objets

- Dès qu'un objet est créé, il est donné une identité qui reste inchangée

```
id(22)       # => 4323728688 (par exemple)
```

Objets

- Dès qu'un objet est créé, il est donné une identité qui reste inchangée

```
id(22)       # => 4323728688 (par exemple)
```

- Chaque objet a un type, le type d'un objet détermine ce qu'on peut faire avec l'objet

```
type(1)          # => int
type("abc")      # => str
```

  - Type est également un object

```
isinstance(type(3.0), object)        # => True
```

Objets

- Dès qu'un objet est créé, il est donné une identité qui reste inchangée

```
id(22)       # => 4323728688 (par exemple)
```

- Chaque objet a un type, le type d'un objet détermine ce qu'on peut faire avec l'objet

```
type(1)          # => int
type("abc")      # => str
```

  - Type est également un object

```
isinstance(type(3.0), object)        # => True
```

- Chaque objet a une valeur

Dictionnaires

Qu’est-ce un dictionnaire

Une affectation (modifiable) des valeurs aux objets

couples (clé, valeur)

Travailler avec les dictionnaires

```
empty = {}
type(empty)      # => dict

a = {"un":1, "deux":2, "trois": 3}
b = dict(un = 1, deux = 2, trois = 3)
c = dict([('un', 1), ('deux', 2), ('trois', 3)])
a == b == c          # => True
```

Travailler avec les dictionnaires

```
empty = {}
type(empty)      # => dict

a = {"un":1, "deux":2, "trois": 3}
b = dict(un = 1, deux = 2, trois = 3)
c = dict([('un', 1), ('deux', 2), ('trois', 3)])
a == b == c          # => True


len(a)               # => 3
key in a.keys()
key in a             # équivalent à "key in d.keys()"
value in a.values()
a.copy()
a.clear()            # rendre a vide
```

Travailler avec les dictionnaires

```
empty = {}
type(empty)      # => dict

a = {"un":1, "deux":2, "trois": 3}
b = dict(un = 1, deux = 2, trois = 3)
c = dict([('un', 1), ('deux', 2), ('trois', 3)])
a == b == c          # => True


len(a)               # => 3
key in a.keys()
key in a             # équivalent à "key in d.keys()"
value in a.values()
a.copy()
a.clear()            # rendre a vide



# l'accès
a['un']          # => 1
a['quatre']      # => KeyError

# modifications
a['deux'] = 22  # => modifier la valeur
a['quatre'] = 4 # ajouter une nouvelle clé
```

Travailler avec les dictionnaires

```python
d = {"Alice" : [1, 2, 3], "Bob" : 4, 5, 6}

# méthode get
d.get("Alice")       # => [1, 2, 3]
d.get("Charlie")     # => None (pas KeyError)

# si None est une valeur valide du dictionnaire?
res = d.get("Bob", [])      # return [] si la clé "Bob" n'existe pas
```

Travailler avec les dictionnaires

```python
d = {"Alice" : [1, 2, 3], "Bob" : 4, 5, 6}

# méthode get
d.get("Alice")       # => [1, 2, 3]
d.get("Charlie")     # => None (pas KeyError)

# si None est une valeur valide du dictionnaire?
res = d.get("Bob", [])      # return [] si la clé "Bob" n'existe pas
```

```python
# supprimer les éléments

del d.['Alice']
del d.['Danny']     # KeyError

# récupérer les éléments
d.pop("Bob", None)  # => [4, 5, 6] et return None si la clé n'exsite pas
```

Travailler avec les dictionnaires

```
d = {"Alice" : [1, 2, 3], "Bob" : 4, 5, 6}

d.keys()      # => <"Alice", "Bob">
d.values()    # => <[1, 2, 3], [4, 5, 6]>
d.items()     # => <("Alice", [1, 2, 3]), ("Bob", [4, 5, 6])>
```

Travailler avec les dictionnaires

```
d = {"Alice" : [1, 2, 3], "Bob" : 4, 5, 6}

d.keys()       # => <"Alice", "Bob">
d.values()     # => <[1, 2, 3], [4, 5, 6]>
d.items()      # => <("Alice", [1, 2, 3]), ("Bob", [4, 5, 6])>
```

```
for clé, valeur in d.items():
    print(clé, valeur)

val_liste = list(d.keys())
```

Compréhensions

Qu’est-ce une compréhension

Une compréhension fournit un moyen
de construction très concise

Questions réponses

```
questions = ['nom', 'age', 'couleur préférée']
réponses = ['Jean', 22, 'bleu']

for q, r in zip(questions, réponses):
    print('Quel(le) est votre {0}? Mon (Ma) {0} est {1}.format(q,r))
```

Questions réponses

```
questions = ['nom', 'age', 'couleur préférée']
réponses = ['Jean', 22, 'bleu']

for q, r in zip(questions, réponses):
    print('Quel(le) est votre {0}? Mon (Ma) {0} est {1}.format(q,r))
```

```
# =>  Quel(le) est votre nom? Mon (Ma) nom est Jean
# =>  Quel(le) est votre age? Mon (Ma) age est 22
# =>  Quel(le) est votre couleur préférée? Mon (Ma) couleur préférée est bleu
```

Courses

```
courses = ["oranges", "pommes", "poires",
"oranges", "bananes"]

for c in sorted(set(courses)):
    print(c)
```

Courses

```
courses = ["oranges", "pommes", "poires",
"oranges", "bananes"]

for c in sorted(set(courses)):
    print(c)
```

```
# bananes
# poires
# pommes
# oranges
```

Nombres carrés

```
carrés = []

for x in range(10):
    carrés.append(x ** 2)

carrés           # => [0, 1, 4, ...]
```

Nombres carrés

```
carrés = []

for x in range(10):
    carrés.append(x ** 2)

carrés          # => [0, 1, 4, ...]
```

- Plus simple

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

Nombres carrés

```
carrés = []

for x in range(10):
    carrés.append(x ** 2)

carrés          # => [0, 1, 4, ...]
```

- Plus simple

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

```
[f(x) for x in collection if cond(x)]
```