

---

**j4**



# J4: Objectifs

---

- Structures de données – containers
- Séquences
  - chaînes
  - listes
  - tuples
- Opérateurs sur les séquences
- Slicing
- Range
- Enumerate
- Ensembles
- Dictionnaires
- Del
- None
- Compréhensions
- Actions répétées

# Conteneurs

---

- Python distingue les types de données élémentaires et les conteneurs (*containers*)
- Ce sont des structures de données construites sur les types élémentaires
- Elles se distinguent par leurs caractéristiques
  - Ordonnée
  - Indexée
  - Muable ou immuable, *mutable or immutable* (modifiable)
  - Itérable, c'est à dire utilisable dans une boucle *for ... in itérable*
  - Hachable, c'est à dire qui peut servir de clé pour un dictionnaire.

# Hashable

---

- Propriété d'une classe qui la rend apte à servir de clé dans un dictionnaire
- A réserver de préférence aux objets immuables
  - éviter les anomalies de mise à jour des dictionnaires
- Possèdent la méthode `__hash__`

<https://stackoverflow.com/questions/13041352/python-quickly-hash-mutable-object>

# Séquences

---

- Les conteneurs qui sont
  - ordonnés
  - indexés par des entiersconstituent la famille des "séquences".
- Les séquences sont itérables

# str

- Chaînes, *String*
- Non modifiable
- C'est une séquence
- Méthodes
  - isupper(), islower(), capitalize(), swapcase()
  - upper(), lower()
  - zfill(width)
  - strip(), lstrip(), rstrip()
  - find(sous\_chaine, début, fin)
  - replace( s1, s2, n)
  - split (sep, max)
  - join(seq)

# Indexation

- *Index* ou indice, numéro d'un élément d'une séquence. On accède à cet élément par son numéro en utilisant la notation entre crochets, par exemple :
  - `s[2]`
- 1er élément : numéro 0 , désigné par `s[0]`
  - `s[2]` désigne le 3ème élément.
- Dernier élément : numéro -1
  - `s[-1]`
- Index positif pour numéroté à partir du début
  - compris entre 0 et `len(s)-1`
- Index négatif pour numéroté à partir de la fin
  - compris entre -1 et `-len(s)`
- Ceci s'applique à toutes les séquences



# Notation pointée

---

- Les méthodes sont des fonctions qui s'appliquent à un objet
- Elles sont accessibles par la notation pointée
- Ainsi pour une chaîne `s`, on écrit :
  - `s.replace('Pierre', 'Paul')`
  - `s.strip()`les méthodes `replace` et `strip` s'appliquent à la chaîne `s`

# Fonction vs Méthode

## Fonction

```
f(x)           #notation fonctionnelle

f(x, t)        #lorsqu'il y a plus d'un arg

f(g(h(x)))     #composition

#
# Exemple
#
len(x)          #nombre d'éléments
```

len est une fonction

Fonction est un concept mathématique  
Rencontré dans les langages procéduraux

## Méthode

```
x.f()          #notation pointée

x.f(t)         #lorsqu'il y a plus d'un arg

x.h().g().f()  #composition

#
# Exemple
#
x.insert(i,e)  #insérer e dans x en i
```

insert est une méthode pour les listes

Méthode est un concept informatique  
Introduit par les langages orientés objets

# list

- Liste, list <class 'list'>
- C'est une séquence
- Notée entre [ ]
- Éléments séparés par des virgules ,
- Modifiable , non hachable
- Éléments de tous types et pas forcément le même, listes de listes
- littéraux
  - []
- opérateurs
  - del l[3] : détruit le 4<sup>ème</sup> élément
- méthodes (accessibles par la notation pointée)
  - sort, append, remove, index, pop, push, count, extend, index, reverse
  - une slice peut se trouver en partie gauche d'une affectation de liste

# tuple

- En français, on dit aussi : n-uplet
  - C'est la généralisation de : couple, triplet, quadruplet
  - Un élément d'un produit cartésien – par nature, donc : ordonné.
  - Noter qu'un couple est formellement différent d'une paire
    - paire : ensemble à 2 éléments, non ordonné
- En Python, un tuple est noté entre ( ) avec une virgule , séparatrice,
  - la virgule est la vraie marque du tuple
  - les parenthèses servent aussi aux expressions parenthésées
  - tuple à 1 élément : 3.14, ou (3.14,)
- Un tuple est une séquence, non modifiable, hachable
- Il peut être hétérogène

# Opérateurs sur les séquences

---

- appartenance `in`
- concaténation `+`
- répétition `*`
- slicing `s[start: stop: step]`
- longueur `len(s)`
- Plus petit `min(s)`
- Plus grand `max(s)`
- Recherche `s.index(x)`
- Compte `s.count(x)`



# Slicing

- S'applique à toutes les séquences
- *Slice* : Tranche. C'est une séquence.
- Exemples de tranches d'une séquence `s`
  - `s[2:4]` : la séquence du 2ème élément inclus au 4ème élément exclus
  - `s[:4]` : séquence du début au 4ème élément exclus
  - `s[2:]` : séquence allant du 2ème élément compris jusqu'à la fin
- Forme générale `s [start:stop:step]`
- Tranches particulières
  - `s[:]` tout
  - `s[::-1]` la même séquence mais à l'envers
  - `s[::3]` prend un élément sur 3

# range

- Crée une liste de nombres entiers en progression arithmétique
- `range(stop)`
  - les nombres de 0 à stop-1
- `range(start, stop [,step])`
  - les nombres de start à stop-1 par pas de step
- Fabrique un objet *range*
- Les éléments contenus sont produits lorsque l'on y accède.

```
range(20)
```

```
list(range(20))
```



# Références

- Les variables qui référencent les mêmes objets muables subissent les mêmes modifications.
- Affectation : référence au même objet
- Copie : référence à une copie
- `import copy`
- `y = copy.copy(x)`
- `z = copy.deepcopy(x)`

Classeur Jupyter Deepcopy.ipynb

# set

- Ensemble
  - non redondant, pas de doublons
  - non ordonné
  - itérable
  - modifiable
    - mais frozenset est non modifiable
- Accolades pour les littéraux
- Opérateur
  - union |
  - intersection &
  - différence –
  - différence symétrique ^
- Méthodes
  - inclusion .issubset(), issuperset()

```
ens_vide = set()
```

```
couleurs = {'Rouge', 'Vert', 'Bleu'}
```

Classeurs Jupyter Sets.ipynb

# dict

- Dictionnaire
- Ensemble de couples clé:valeur
- Noté {clé : valeur, ... }
  - La clé doit être hachable
  - La valeur peut être quelconque
- Il n'est pas ordonné. Mais il est modifiable.
- Opérateur
  - appartenance in
  - del
- Méthodes
  - .keys() regarder quel est le type renvoyé (set ou list)
  - .values()
  - .items() liste de couples clé-valeur
- sorted

```
dict_vide = {}  
  
notes = {'Pierre':18,  
         'Paul':12, 'Marie':20}
```

Classeur Jupyter Dict.ipynb

# del

- Effacer un élément d'un conteneur muable ; ici, une liste
- Effacer un élément de dictionnaire
- Effacer une variable locale  
Après un del, la variable devient non définie
- del est différent de

```
del l[3]
```

```
del note['Pierre']
```

```
del x
```

```
x = None
```

Classeur Jupyter Del.ipynb

# None

- Une constante prédéfinie qui signifie *l'absence de valeur* pour une variable qui est définie
- La variable pointe alors vers un objet None du type NoneType (et c'est le seul de ce type)
- Lorsqu'une fonction ne renvoie pas de résultat, elle renvoie None
- Utile pour les arguments optionnels à des fonctions
  - On compare l'argument qui manque par is

```
if arg is None :
```

```
Classeur Jupyter delnone.ipynb
```

# Actions répétées

---

- While
- For

# while

- Boucle très générale, proche de ce qui se fait dans d'autres langages de programmation
- "Tant que"
- La boucle "Répéter jusqu'à" n'existe pas en Python
- Une clause else ajoute une action qui s'exécute quand on sort de la boucle.
  - savoir que cela existe
  - l'action finale est toujours exécutée
  - sauf en cas de "break" voir plus loin

```
while condition:  
    action
```

```
while condition:  
    action  
else:  
    action_finale
```

# repeat - until

- Dans d'autre langages ,on écrit :
- La condition est une *condition d'arrêt*
- L'action est toujours exécutée au moins une fois
- Mais on peut faire :

```
# N'existe PAS en Python !  
# repeat  
#   action  
# until condition
```

```
while True:  
    action  
    if condition:  
        break
```



# do - while

- Dans d'autre langages ,on écrit :
- La condition est une *condition de continuation*
- L'action est toujours exécutée au moins une fois
- Mais on peut faire :

```
# N'existe PAS en Python !  
# do  
#   action  
# while condition
```

```
while True:  
    action  
    if not condition:  
        break
```

# for

- Parcourir un intervalle de nombres entiers
- mais l'on peut parcourir d'autres itérables, par ex: une liste
- ou une chaîne
- ou un ensemble
- Pour parcourir un dictionnaire, on boucle sur ses éléments constitutifs
- Peut prendre une clause else, exécutée en fin de boucle

```
for i in range (a, b, step):  
    action
```

```
for x in ['Riri', 2, 3.14]:  
    action
```

```
for c in "Nabuchodonosor":
```

```
for c in { 'R', 'G', 'B', 'Y', M', 'C' }:
```

```
for x in dict.keys():
```

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

```
for k,v in dict.items():
```

# Compréhension

- Maths

- Définir un ensemble *en extension* : citer tous ses éléments
- Définir un ensemble *en compréhension* :
  - à l'aide d'autres ensembles,
    - par ex "sous ensemble de"
  - en caractérisant les éléments par une propriété.

```
l = [ n**2 for n in primes] # liste
```

```
noms_propres = {mot for mot in dico \  
if mot[0].isupper()} # ensemble
```

- Python:

- à l'aide de for et if
- S'applique aux tuples, listes, ensembles, dictionnaires – tout objet itérable

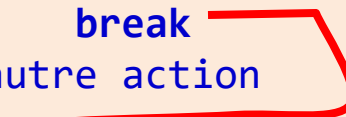
```
from random import randint  
N1=1000  
N2=9999  
pseudo = {nom:randint(N1,N2) for \  
nom in noms_propres} # dictionnaire
```

Classeurs Jupyter Compréhension.ipynb  
Extension vs compréhension.ipynb

# Ruptures

- Sortir de la boucle et passer à la suite avec `break`
- En cas de `break`, l'action finale introduite par `else`: n'est pas exécutée
- Rester dans la boucle mais passer à l'itération suivante avec `continue`

```
for i in range (a, b, step):  
    action  
    if condition:  
        break  
    autre action  
suite
```



```
while condition1:  
    action  
    if condition2:  
        continue  
    autre action  
suite
```

# sort et sorted

---

- Si l est une liste
  - l.sort() réarrange le contenu de l sur place
  - sorted(l) crée une nouvelle liste triée

# enumerate

- S'applique à tout container itérable
- affecte un numéro à chaque élément parcouru
  - donner le numéro de départ
- renvoie un tuple
  - numéro
  - élément

Classeur Jupyter Enumerate.ipynb

# Exercice $n!$ itératif

---

- Ecrire la fonction factorielle en utilisant un algorithme itératif

# Exercice Recherche dans une chaîne

---

- Rechercher si la sous-chaîne "pattern" apparaît dans au moins l'un des mots de la liste "words"
- ```
def contient(words, pattern):  
    pass
```



# Exercice Vecteurs

- Représenter un vecteur de  $n$  composantes :
  - liste avec  $n$  éléments

$v = [1, 12, 30]$  , un exemple de vecteur avec 3 composantes

def somme(v):

pass #TODO à vous de faire (le total des nombres contenus dans le vecteur)

- Lire un vecteur de dimension  $N$
- Afficher un vecteur
- Faire la somme de 2 vecteurs
- Multiplication d'un vecteur par un scalaire (un nombre flottant)
- Faire le produit scalaire de 2 vecteurs

# Exercice Matrices

---

- Lire une matrice  $N \times N$
- Afficher une matrice  $N \times N$
- Ajouter deux matrices
- Multiplier un vecteur par une matrice
- Multiplier une matrice par une matrice

# Exercice Nombres premiers

---

- Trouver les nombres premiers inférieurs à  $N$
- On rappelle qu'un nombre est premier si et seulement s'il a deux diviseurs : 1 et lui-même.
- Selon cette définition, 1 n'est pas premier (car un seul diviseur : 1)
- La méthode du crible d'Eratosthène consiste à éliminer de l'espace de recherche les nombres qui sont multiples des nombres premiers déjà trouvés.
- Afficher les nombres premiers trouvés dans l'ordre croissant

# Exercice PGCD itératif

---

- Ecrire la fonction PGCD en évitant l'algorithme récursif déjà visité et en utilisant une boucle

# Les mots-clés ajoutés

|        |          |       |        |          |        |
|--------|----------|-------|--------|----------|--------|
| False  | None     | True  | and    | as       | assert |
| async  | await    | break | class  | continue | def    |
| del    | elif     | else  | except | finally  | for    |
| from   | global   | if    | import | in       | is     |
| lambda | nonlocal | not   | or     | pass     | raise  |
| return | try      | while | with   | yield    |        |

# Merci !

- Restons en contact :
  - Georges Georgoulis – [ggeorgoulis@alteractifs.org](mailto:ggeorgoulis@alteractifs.org) – 06 12 68 40 06



Coopérative d'activité et d'entrepreneurs [www.alteractifs.org](http://www.alteractifs.org)