PARADIGMES DE PROGRAMMATION

DIPLÔME DE FORMATION D'INGÉNIEURS EN INFORMATIQUE (FI-A1)

Module 4

Mondher Bouden

2025-2026

Les principales structures de données

Les chaînes de caractères

- Type séquentiel ordonné, dénombrable.
- Type immuable : une chaîne de caractères ne peut pas être modifiée.
- Les éléments d'une chaîne sont de même nature : ce sont des caractères.

Les chaînes de caractères

Caractères spéciaux :

- \' : Pour utiliser une apostrophe dans une chaîne de caractères délimitée par des apostrophes.
- \t : Une tabulation.
- \n : Un "retour de charriot" (touche Entrée).

En utilisant 3 guillemets, on peut écrire des chaînes de caractères sur plusieurs lignes et contenant des apostrophes et des guillemets.

```
phrase_4 = "Bonjour\'\ttout le\nmonde!"
phrase_5 = """ Bonjour " tout le
                monde!" """
```

Les chaînes de caractères

- On peux accéder à des caractères ou des sous-chaînes en utilisant les opérateurs [et].
- Le premier index de la chaîne est 0.
- On peut également indexer à partir de la fin, en commençant par −1.

```
>>> phrase_6 = "Bonjour tout le monde!"
>>> phrase_6[0]
'B'

>>> phrase_6[-1]
'!'
```

Les opérateurs sur les chaînes de caractères

- La concaténation : l'opérateur +
- La conversion en valeur numérique : int(), float()
- La conversion en chaîne : str()
- La longueur d'une chaîne : len()
- L'appartenance d'un élément : in, not in

```
>>> phrase_1 = "Bonjour"
>>> phrase_2 = "Ok"
>>> phrase_3 = phrase_1 + phrase_2
>>> 'o' in phrase_1
True
>>> 'o' in phrase_2
False
```

Les opérateurs sur les chaînes de caractères

- La comparaison de chaînes de caractères : ==
  - Attention à la casse !
  - Les chaînes sont ordonnées :
    - phrase_1 < phrase_2
    - 'A' < 'a' < 'à'
    - Les fonctions ord() et chr() permettent de connaître le code numérique associé à un caractère, et vice-versa.

```
>>> ord('o')
111
>>> ord('O')
79
>>> chr (80)
"P"
```

Exemples

Exemple_Strings.py - C:/Exemples/Exemple_Strings.py (3.7.0)

File Edit Format Run Options Window Help

```
ph1 = 'Bonjour\'\ttout le\nmonde!'
ph2 = """Bonjour ' tout le
                        monde!' """
print(ph1)
print(ph2)
print(len(ph1))
ch = str(15)
print(ch+"20")
print(type(ch))
print(ord('o'))
print(chr(80))
ch1 = "Bonjour"
print('o' in ch1)
print('B' not in ch1)
ch2 = "Salut"
print(id(ch1), id(ch2))
ch1 = ch2
print(id(ch1), id(ch2))
```

Ln: 18 Col: 23

Python 3.7.0 Shell

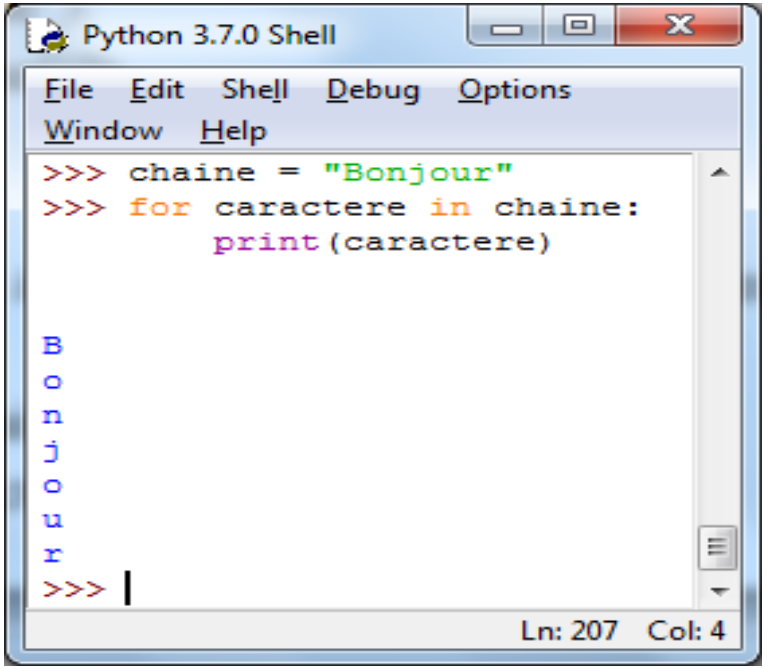
File Edit Shell Debug Options Window Help

```
Bonjour'        tout le
monde!
Bonjour ' tout le
                        monde!'
23
1520
<class 'str'>
111
P
True
False
46722048 46722104
46722104 46722104
>>>
```

Ln: 340 Col: 4

Les opérateurs sur les chaînes de caractères

Le parcours par une *itération* des caractères de l'index 0 à l'index len(chaine) – 1




```
i = 0
while i < len (chaine ):
        print (chaine[i])
        i = i + 1
```

On peut également *itérer élément par élément*, avec une boucle **for** !

```
for caractere in chaine:
        print (caractere)
```

Les listes

- Type séquentiel ordonné, dénombrable.
- Type mutable : on peut modifier une liste.
- Les éléments peuvent être hétérogènes.
- Une liste peut contenir des sous-listes.
- On peux accéder à des éléments ou des sous-listes en utilisant les opérateurs [et].
- Le premier index de la liste est 0.
- On peut également indexer à partir de la fin, en commençant par −1.

Les listes

nombre = [5, 38, 10, 25]

suites = [[5, 38], [10, 25]]

mots = ["lait", "fromage ", "confiture ", "chocolat "]

trucs = [5000 , "Brigitte ", 3.1416 , ["Albert ", "René", 1947]]

jours = ["dimanche ", "lundi ", "mardi ", "mercredi ", "jeudi ", "vendredi ", "samedi "]

print (jours [0], jours [1], jours [-1])

Les opérateurs sur les listes

mots = ["lait ", "fromage ", "confiture ", "chocolat "]

- len(mots) donne le nombre d'éléments dans la liste;
- Tous les éléments sont indexés de 0 à len(mots) - 1
- mots[i] donne accès à un élément de la liste mots si i est dans l'intervalle [−len(mots), len(mots) − 1]
- Par convention,
  - mots[+i] identifie le (i-1)-ième élément à partir du début.
  - mots[-i] identifie le i-ième élément à partir de la fin.

Les opérateurs sur les listes

- Ajout à la fin : l'opérateur +
- Comparaison élément par élément : l'opérateur ==

```
>>> mots = ["lait", "fromage ", "confiture ", "chocolat "]
>>> mots = mots + ["oeufs "]
>>> mots == ["lait", "fromage ", "confiture ", "chocolat ", "oeufs "]
True
```

Les opérateurs sur les listes

- Insertion/remplacement : utilisation des opérateurs [], en spécifiant une *tranche*.

```
>>> mots = ["lait", "fromage", "confiture ", "chocolat "]
>>> mots [2:2] = ["miel"]
>>> mots
["lait", "fromage", "miel", "confiture", "chocolat "]
```

Une tranche ?

- mots = [" lait", "fromage ", "confiture ", "chocolat "]

↑	↑	↑	↑	↑
0	1	2	3	4

Insertion dans une liste

- mots = [" lait", "fromage ", "confiture ", "chocolat "]

↑	↑	↑	↑	↑
0	1	2	3	4

Insertion / remplacement :

```
>>> mots [2:2] = ["miel"]
>>> mots
["  lait", "fromage ", "miel", "confiture ", "chocolat "]
```

Insertion dans une liste

- mots = [" lait", "fromage ", "confiture ", "chocolat "]

↑	↑	↑	↑	↑
0	1	2	3	4

Insertion / remplacement :

```
>>> mots [1:3] = ["pain"]
>>> mots
[" lait ", "pain", "chocolat "]
```

Retrait dans une liste

- mots = [" lait", "fromage ", "confiture ", "chocolat "]

↑	↑	↑	↑	↑
0	1	2	3	4

Retrait :

```
>>> mots [2:4] = [] # La liste vide
>>> mots
[" lait", "fromage "]
```

Retrait dans une liste

- mots = [" lait", "fromage ", "confiture ", "chocolat "]

↑	↑	↑	↑	↑
0	1	2	3	4

Retrait (2) :

```
>>> del (mots [1])
>>> mots
[" lait", "confiture ", "chocolat]
```

Retrait dans une liste

- mots = [" lait", "fromage ", "confiture ", "chocolat "]

↑	↑	↑	↑	↑
0	1	2	3	4

Lorsqu'on veut une tranche à partir du début ou bien jusqu'à la fin, on peut omettre le premier et/ou le dernier terme autour du :

```
mots [:3] # équivalent à mots [0:3]
mots [1:] # équivalent à mots [1:4]
mots [:] # équivalent à mots [0:4]
```

Appartenance à une liste

- mots = [" lait", "fromage ", "confiture ", "chocolat "]

↑	↑	↑	↑	↑
0	1	2	3	4

Appartenance d'un élément : in, not in

```
>>> "fromage " in mots
True
>>> "patate " in mots
False
>>> 8 in mots:
False
```

Itération sur les éléments

- mots = [" lait", "fromage ", "confiture ", "chocolat "]

↑	↑	↑	↑	↑
0	1	2	3	4

```
for element in mots:
        print (element)
```

lait
fromage
confiture
chocolat

Fonctions utiles reliées aux listes

Rappel

- range(n)
  - Crée une liste dont les éléments sont les entiers compris entre 0 et n-1
  - n doit être un entier positif ou nul
  - range(0) retourne une liste vide
- range(from, to, step)
  - from est la valeur initiale, de type int
  - to est une valeur d'arrêt, exclue de l'intervalle, de type int
  - step est l'incrément, de type int
  - range(n) équivaut à range(0, n, 1)

Fonctions utiles reliées aux listes

Exemple_ForRange2.py - C:\Exemples\Exemple_ForRange2.py (3.7.0)

File Edit Format Run Options Window Help

```
#intervalle = list(range(5))
#intervalle = list(range(-1, 5))
intervalle = list(range(5, -1, -1))

for nb in intervalle:
    print(nb)
print("terminé")
```

Ln: 7 Col: 16

Python 3.7.0 Shell

File Edit Shell Debug Options Window Help

```
5
4
3
2
1
0
terminé
>>>
```

Ln: 164 Col: 4

Fonctions utiles reliées aux listes

- max(liste), min(liste)
  - Retourne le maximum ou le minimum de la liste
  - Dépend de la nature des éléments de la liste
  - Les éléments doivent être comparables
- sum(list)
  - Retourne la somme des éléments de la liste
  - Les éléments doivent pouvoir être additionnés

Exemples

Exemple_Listes.py - C:/Exemples/Exemple_Listes.py (3.7.0)

File Edit Format Run Options Window Help

```
liste1 = [1,2,3]
liste2 = [4,5,6]
liste2 = liste1 * 2
print(liste2)
liste3 = [[1,2,3],[5,7,9],[6,4,8]]
print(liste3[1][1:3])
lst = [1,2,3,4,5,6,7,8,9,10]
print(sum(lst))
print(max(lst))
print(min(lst))
print(sum(range(max(range(101)))))
liste1 = [1,2,3,4,5]
liste2 = liste1
print(liste1,liste2)
liste1[3:] = []
print(liste1,liste2)
```

Ln: 16 Col: 20

Python 3.7.0 Shell

File Edit Shell Debug Options Window Help

```
[1, 2, 3, 1, 2, 3]
[7, 9]
55
10
1
4950
[1, 2, 3, 4, 5] [1, 2, 3, 4, 5]
[1, 2, 3] [1, 2, 3]
>>>
```

Ln: 390 Col: 4

Copie d'une liste

- On peut utiliser le *constructeur* de la structure de donnée **list**, ou bien une tranche, qui fait également une copie.

```
liste_1 = [1, 2, 3, 4]
liste_2 = list(liste_1)  # Une copie sera créée
liste_3 = liste_1[:]  # Une tranche allant du début
                      # à la fin de la liste.
```

Les dictionnaires

- Type non ordonné, dénombrable
- Type mutable : on peut modifier un dictionnaire
- Chaque élément est associé à une clé, on a donc des *couples* <clé, élément>
- On accède aux éléments par la clé au lieu d'utiliser un index
- Les éléments peuvent être hétérogènes
- Un dictionnaire peut contenir des dictionnaires !

Les dictionnaires

```
>>> t = {} # dictionnaire vide
>>> t["computer "] = "ordinateur "
>>> t["mouse "] = "souris "
>>> t["keyboard "] = "clavier "
>>> t
{"mouse ": "souris ", "computer ": "ordinateur ",
"keyboard ": "clavier "}
```

Les opérateurs sur les dictionnaires

t = {"mouse ": "souris ", "computer ": "ordinateur ",

"keyboard ": "clavier "}

len(t) donne le nombre d'éléments (ou de couples <clé, valeur>)

>>> **len** (t)

3

Ajout ou remplacement :

>>> t["cloud "] = "nuage "

>>> t["mouse "] = "souris optique "

Retrait :

>>> **del** (t["mouse "])

Les opérateurs sur les dictionnaires

t = {"mouse ": "souris ", "computer ": "ordinateur ",
"keyboard ": "clavier "}

Appartenance (fonctionne avec les clés) :

>>> "computer " **in** t

True

>>> "ordinateur " **in** t

False

Itération sur les éléments d'un dictionnaire, par les clés :

```
for cle in t:
        print (cle)
        print (t[cle]) # On accède à une valeur en
                       # utilisant sa clé
```

Les opérateurs sur les dictionnaires

t = {"mouse ": "souris ", "computer ": "ordinateur ",
"keyboard ": "clavier "}

Obtenir une liste des clés :

>>> **list** (t.keys ())

["mouse ", "computer ", "keyboard "]

Obtenir une liste des valeurs :

>>> **list** (t.values ())

["souris ", "ordinateur ", "clavier "]

Attention : Les clés et valeurs peuvent être énumérées dans n'importe quel ordre !

Les fichiers

- Permettent de conserver de l'information, tel les bases de données.
- Permettent d'échanger de l'information.
- On peut modifier un fichier.
- Dans le cadre de ce cours, nous ne traiterons que les fichiers contenant du texte.
- Nous utiliserons un ensemble de fonctions prédéfinies pour interagir avec un fichier.

Ouverture et fermeture d'un fichier

Il faut toujours ouvrir un fichier avant de s'en servir, en *lecture* ou en *écriture*.
On utilise ensuite le fichier, et on le ferme par la suite.

```
# Ouverture en lecture
f_1 = open('nom_du_fichier_existant.txt', 'r')

# Ouverture en écriture
f_2 = open('nom_du_nouveau_fichier.txt', 'w')

# Utilisation du fichier...

# Fermeture
f_1.close()
f_2.close()
```

Ouverture et fermeture d'un fichier

```
# Lecture d'une ligne (jusqu'au prochain caractère
# spécial \n)
ma_chaine = f_1.readline()

# Lecture d'un caractère. Si read() renvoie une chaîne
# vide, on est rendu à la fin du fichier.
mon_car = f_1.read(1)

# Écriture d'une chaîne
f_2.write("Désolé!")
f_2.write(chaine)
f_2.write("\nfini!")
```

Ces opérations ne sont possibles que si les fichiers sont ouverts dans le bon mode (lecture et écriture), et il ne faut pas oublier de fermer les fichiers à la fin !

Exemple: Les fichiers

Exemple_Fichiers.py - C:/Exemples/Exemple_Fichiers.py (3.7.0)

File Edit Format Run Options Window Help

```
fichier = open('mon_fichier.txt', 'w')
fichier.write("Bonjour, ceci est un texte.")
fichier.write(" Encore du texte.")
fichier.write("\nTest de retour de chariot.")
fichier.close()

fichier = open('mon_fichier.txt')
ma_chaine = fichier.readline()
print(ma_chaine)
fichier.close()

fichier = open('mon_fichier.txt')
for line in fichier:
    print(line)
```

Ln: 14 Col: 15

mon_fichier.txt - Bloc-notes

Fichier Edition Format Affichage ?

```
Bonjour, ceci est un texte. Encore du texte.
Test de retour de chariot.
```

Python 3.7.0 Shell

File Edit Shell Debug Options Window Help

```
Bonjour, ceci est un texte. Encore du texte.

Bonjour, ceci est un texte. Encore du texte.

Test de retour de chariot.
>>>
```

Ln: 185 Col: 4