

IFT-1004 - Introduction à la programmation

Module 4 : Principaux types de données

Honoré Hounwanou, ing.

Département d'informatique et de génie logiciel
Université Laval

Les chaînes de caractères

Les listes

Les dictionnaires

Les fichiers

Lectures et travaux dirigés

Les chaînes de caractères

Objectif

Se familiariser avec le traitement de données textuelles.

- *Type séquentiel ordonné, dénombrable*

- Type *séquentiel ordonné, dénombrable*
- Type *immuable* : une chaîne de caractères ne peut pas être modifiée

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

- 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

- 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

```
phrase_1 = 'Bonjour' # Utilisation d'apostrophes
```

```
phrase_2 = "Bonjour" # Utilisation de guillemets
```

```
phrase_3 = """Bonjour""" # Utilisation de 3 guillemets  
# Surtout utilisé pour la  
# documentation
```

- Caractères spéciaux :

- Caractères spéciaux :
 - `\'` : Pour utiliser une apostrophe dans une chaîne de caractères délimitée par des apostrophes

- Caractères spéciaux :
 - `\'` : Pour utiliser une apostrophe dans une chaîne de caractères délimitée par des apostrophes
 - `\t` : Une tabulation

- 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)

- 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)
 - `\\` : Une barre oblique inversée.

- 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)
 - `\\` : Une barre oblique inversée.
- En utilisant 3 guillemets, on peut écrire des chaînes de caractères sur plusieurs lignes et contenant des apostrophes et guillemets

- 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)
 - `\\` : Une barre oblique inversée.
- En utilisant 3 guillemets, on peut écrire des chaînes de caractères sur plusieurs lignes et contenant des apostrophes et guillemets

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)
 - `\\` : Une barre oblique inversée.
- En utilisant 3 guillemets, on peut écrire des chaînes de caractères sur plusieurs lignes et contenant des apostrophes et guillemets

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

- On peut accéder à des caractères ou des sous-chaînes en utilisant les opérateurs [et]

- On peut accéder à des caractères ou des sous-chaînes en utilisant les opérateurs [et]
- Le premier index de la chaîne est 0

- On peut 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 , puis -2 , etc.

- On peut 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 , puis -2 , etc.

- On peut 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, puis -2, etc.

```
>>> phrase_6 = "Bonjour tout le monde!"
```

```
>>> phrase_6[0]
```

```
'B'
```

```
>>> phrase_6[-1]
```

```
'!'
```

Représentation en mémoire

- Les chaînes de caractères ne sont pas copiées ;
- On accède directement à la mémoire où elles ont été créées
- Les **variables** auxquelles on affecte des chaînes de caractères **contiennent les adresses des blocs mémoires où sont stockées ces chaînes**;
- `phrase_1 = 'Bonjour'` 12934
- `phrase_2 = 'Ok'` 12942
- `phrase_1 = phrase_2`

Nom	Adresse	Contenu	Type
...	6111
...	6112
...	6113
...	6114

Représentation en mémoire

- Les chaînes de caractères ne sont pas copiées ;
- On accède directement à la mémoire où elles ont été créées
- Les **variables** auxquelles on affecte des chaînes de caractères **contiennent les adresses des blocs mémoires où sont stockées ces chaînes**;
- `phrase_1 = 'Bonjour'` 12934
- `phrase_2 = 'Ok'` 12942
- `phrase_1 = phrase_2`

B	o	n	j	o	u	r	\0

Nom	Adresse	Contenu	Type
...	6111
...	6112
...	6113
...	6114

Représentation en mémoire

- Les chaînes de caractères ne sont pas copiées ;
- On accède directement à la mémoire où elles ont été créées
- Les **variables** auxquelles on affecte des chaînes de caractères **contiennent les adresses des blocs mémoires où sont stockées ces chaînes**;
- `phrase_1 = 'Bonjour'` 12934
- `phrase_2 = 'Ok'` 12942
- `phrase_1 = phrase_2`

B	o	n	j	o	u	r	\0

Nom	Adresse	Contenu	Type
...	6111
phrase_1	6112
...	6113
...	6114

Représentation en mémoire

- Les chaînes de caractères ne sont pas copiées ;
- On accède directement à la mémoire où elles ont été créées
- Les **variables** auxquelles on affecte des chaînes de caractères **contiennent les adresses des blocs mémoires où sont stockées ces chaînes**;
- `phrase_1 = 'Bonjour'` 12934
- `phrase_2 = 'Ok'` 12942
- `phrase_1 = phrase_2`

B	o	n	j	o	u	r	\0

Nom	Adresse	Contenu	Type
...	6111
phrase_1	6112	12934	*str
...	6113
...	6114

Représentation en mémoire

- Les chaînes de caractères ne sont pas copiées ;
- On accède directement à la mémoire où elles ont été créées
- Les **variables** auxquelles on affecte des chaînes de caractères **contiennent les adresses des blocs mémoires où sont stockées ces chaînes**;
- `phrase_1 = 'Bonjour'` 12934
- `phrase_2 = 'Ok'` 12942
- `phrase_1 = phrase_2`

B	o	n	j	o	u	r	\0
O	k	\0					

Nom	Adresse	Contenu	Type
...	6111
phrase_1	6112	12934	*str
...	6113
...	6114

Représentation en mémoire

- Les chaînes de caractères ne sont pas copiées ;
- On accède directement à la mémoire où elles ont été créées
- Les **variables** auxquelles on affecte des chaînes de caractères **contiennent les adresses des blocs mémoires où sont stockées ces chaînes**;
- `phrase_1 = 'Bonjour'` 12934
- `phrase_2 = 'Ok'` 12942
- `phrase_1 = phrase_2`

B	o	n	j	o	u	r	\0
O	k	\0					

Nom	Adresse	Contenu	Type
...	6111
phrase_1	6112	12934	*str
phrase_2	6113
...	6114

Représentation en mémoire

- Les chaînes de caractères ne sont pas copiées ;
- On accède directement à la mémoire où elles ont été créées
- Les **variables** auxquelles on affecte des chaînes de caractères **contiennent les adresses des blocs mémoires où sont stockées ces chaînes**;
- `phrase_1 = 'Bonjour'` 12934
- `phrase_2 = 'Ok'` 12942
- `phrase_1 = phrase_2`

B	o	n	j	o	u	r	\0
O	k	\0					

Nom	Adresse	Contenu	Type
...	6111
phrase_1	6112	12934	*str
phrase_2	6113	12942	*str
...	6114

Représentation en mémoire

- Les chaînes de caractères ne sont pas copiées ;
- On accède directement à la mémoire où elles ont été créées
- Les **variables** auxquelles on affecte des chaînes de caractères **contiennent les adresses des blocs mémoires où sont stockées ces chaînes**;
- `phrase_1 = 'Bonjour'` 12934
- `phrase_2 = 'Ok'` 12942
- `phrase_1 = phrase_2`

B	o	n	j	o	u	r	\0
O	k	\0					

Nom	Adresse	Contenu	Type
...	6111
phrase_1	6112	12934	*str
phrase_2	6113	12942	*str
...	6114

Représentation en mémoire

- Les chaînes de caractères ne sont pas copiées ;
- On accède directement à la mémoire où elles ont été créées
- Les **variables** auxquelles on affecte des chaînes de caractères **contiennent les adresses des blocs mémoires où sont stockées ces chaînes**;
- `phrase_1 = 'Bonjour'` 12934
- `phrase_2 = 'Ok'` 12942
- `phrase_1 = phrase_2`

B	o	n	j	o	u	r	\0
O	k	\0					

Nom	Adresse	Contenu	Type
...	6111
phrase_1	6112	12942	*str
phrase_2	6113	12942	*str
...	6114

Représentation en mémoire

À noter : on a perdu l'accès à la chaîne débutant à l'adresse 12934 : aucune variable n'y fait référence.

12934	B	o	n	j	o	u	r	\0
12942	O	k	\0					

Nom	Adresse	Contenu	Type
...	6111
phrase_1	6112	12942	*str
phrase_2	6113	12942	*str
...	6114

Les opérateurs sur les chaînes de caractères

La concaténation : l'opérateur +

- `phrase_1 = 'Bonjour'`
- `phrase_2 = 'Ok'`
- `phrase_1 = phrase_1 + phrase_2`

12934	B	o	n	j	o	u	r	\0
12942	O	k	\0					

Nom	Adresse	Contenu	Type
...	6111
phrase_1	6112	12934	*str
phrase_2	6113	12942	*str
...	6114

Les opérateurs sur les chaînes de caractères

La concaténation : l'opérateur +

- `phrase_1 = 'Bonjour'`
- `phrase_2 = 'Ok'`
- `phrase_1 = phrase_1 + phrase_2`

12934	B	o	n	j	o	u	r	\0
12942	O	k	\0	B	o	n	j	o
	u	r	O	k	\0			

Nom	Adresse	Contenu	Type
...	6111
phrase_1	6112	12934	*str
phrase_2	6113	12942	*str
...	6114

Les opérateurs sur les chaînes de caractères

La concaténation : l'opérateur +

- `phrase_1 = 'Bonjour'`
- `phrase_2 = 'Ok'`
- `phrase_1 = phrase_1 + phrase_2`

12934	B	o	n	j	o	u	r	\0
12942	O	k	\0	B	o	n	j	o
	u	r	O	k	\0			

Nom	Adresse	Contenu	Type
...	6111
phrase_1	6112	12945	*str
phrase_2	6113	12942	*str
...	6114

Les opérateurs sur les chaînes de caractères

La conversion en valeur numérique : `int()`, `float()`

- `ch = '65'`
- `ch = ch + 1`
- `ch = ch + '1'`
- `incr = int(ch) + 1`

932

940

Nom	Adresse	Contenu	Type
...	450
...	451
...	452
...	453

Les opérateurs sur les chaînes de caractères

La conversion en valeur numérique : `int()`, `float()`

- `ch = '65'`
- `ch = ch + 1`
- `ch = ch + '1'`
- `incr = int(ch) + 1`

932
940

6	5	\0					

Nom	Adresse	Contenu	Type
...	450
...	451
...	452
...	453

Les opérateurs sur les chaînes de caractères

La conversion en valeur numérique : `int()`, `float()`

- `ch = '65'`
- `ch = ch + 1`
- `ch = ch + '1'`
- `incr = int(ch) + 1`

932
940

6	5	\0					

Nom	Adresse	Contenu	Type
...	450
ch	451
...	452
...	453

Les opérateurs sur les chaînes de caractères

La conversion en valeur numérique : `int()`, `float()`

- `ch = '65'`
- `ch = ch + 1`
- `ch = ch + '1'`
- `incr = int(ch) + 1`

932	6	5	\0					
940								

Nom	Adresse	Contenu	Type
...	450
ch	451	932	*str
...	452
...	453

Les opérateurs sur les chaînes de caractères

La conversion en valeur numérique : `int()`, `float()`

- `ch = '65'`
- `ch = ch + 1` # Erreur!
- `ch = ch + '1'`
- `incr = int(ch) + 1`

932	6	5	\0					
940								

Nom	Adresse	Contenu	Type
...	450
ch	451	932	*str
...	452
...	453

Les opérateurs sur les chaînes de caractères

La conversion en valeur numérique : `int()`, `float()`

- `ch = '65'`
- `ch = ch + 1` # Erreur!
- `ch = ch + '1'`
- `incr = int(ch) + 1`

932	6	5	\0	6	5	1	\0	
940								

Nom	Adresse	Contenu	Type
...	450
ch	451	932	*str
...	452
...	453

Les opérateurs sur les chaînes de caractères

La conversion en valeur numérique : `int()`, `float()`

- `ch = '65'`
- `ch = ch + 1` # Erreur!
- `ch = ch + '1'`
- `incr = int(ch) + 1`

932	6	5	\0	6	5	1	\0	
940								

Nom	Adresse	Contenu	Type
...	450
ch	451	935	*str
...	452
...	453

Les opérateurs sur les chaînes de caractères

La conversion en valeur numérique : `int()`, `float()`

- `ch = '65'`
- `ch = ch + 1` # Erreur!
- `ch = ch + '1'`
- `incr = int(ch) + 1`

932
940

6	5	\0	6	5	1	\0	

Nom	Adresse	Contenu	Type
...	450
ch	451	935	*str
...	452
...	453

Les opérateurs sur les chaînes de caractères

La conversion en valeur numérique : `int()`, `float()`

- `incr = int(ch) + 1`

`int(ch)` est évalué à l'entier 651. Y additionner 1 donne donc 652.

				932	6	5	\0	6	5	1	\0	
				940								
Nom	Adresse	Contenu	Type									
...	450									
ch	451	935	*str									
...	452									
...	453									

Les opérateurs sur les chaînes de caractères

La conversion en valeur numérique : `int()`, `float()`

- `incr = int(ch) + 1`

`int(ch)` est évalué à l'entier 651. Y additionner 1 donne donc 652.

				932	6	5	\0	6	5	1	\0	
				940								
Nom	Adresse	Contenu	Type									
...	450									
ch	451	935	*str									
incr	452									
...	453									

Les opérateurs sur les chaînes de caractères

La conversion en valeur numérique : `int()`, `float()`

- `incr = int(ch) + 1`

`int(ch)` est évalué à l'entier 651. Y additionner 1 donne donc 652.

				932	6	5	\0	6	5	1	\0	
				940								
Nom	Adresse	Contenu	Type									
...	450									
ch	451	935	*str									
incr	452	652	int									
...	453									

Les opérateurs sur les chaînes de caractères

À noter : on a perdu l'accès à la chaîne débutant à l'adresse 932 : aucune variable n'y fait référence.

932	6	5	\0	6	5	1	\0	
940								

Nom	Adresse	Contenu	Type
...	450
ch	451	935	*str
incr	452	652	int
...	453

Les opérateurs sur les chaînes de caractères

La conversion en chaîne : `str()`

- `ch = str(incr)`

932	6	5	\0	6	5	1	\0	
940								

Nom	Adresse	Contenu	Type
...	450
ch	451	935	*str
incr	452	652	int
...	453

Les opérateurs sur les chaînes de caractères

La conversion en chaîne : `str()`

- `ch = str(incr)`

`str(incr)` est évalué à la chaîne de caractères '652'. La variable `ch` est ensuite mise à jour.

a	932	6	5	\0	6	5	1	\0	6
	940	5	2	\0					

Nom	Adresse	Contenu	Type
...	450
ch	451	935	*str
incr	452	652	int
...	453

Les opérateurs sur les chaînes de caractères

La conversion en chaîne : `str()`

- `ch = str(incr)`

`str(incr)` est évalué à la chaîne de caractères '652'. La variable `ch` est ensuite mise à jour.

a	932	6	5	\0	6	5	1	\0	6
	940	5	2	\0					

Nom	Adresse	Contenu	Type
...	450
ch	451	939	*str
incr	452	652	int
...	453

Les opérateurs sur les chaînes de caractères

À noter : on a perdu l'accès aux chaînes débutant aux adresses 932 et 935 : aucune variable n'y fait référence.

932	6	5	\0	6	5	1	\0	6
940	5	2	\0					

Nom	Adresse	Contenu	Type
...	450
ch	451	939	*str
incr	452	652	int
...	453

La longueur d'une chaîne : `len()`

```
>>> ch = 'Bonjour'
```

```
>>> len(ch)
```

```
7
```

L'appartenance d'un élément : `in`, `not in`

Les opérateurs sur les chaînes de caractères

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
```

Les opérateurs sur les chaînes de caractères

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

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
```

```
>>> if 'o' in phrase_1 and 'o' not in phrase_2:
...     print("Vrai!")
...
Vrai!
```


La comparaison de chaînes de caractères : ==

- Attention à la *casse* (majuscules / minuscules) !

La comparaison de chaînes de caractères : ==

- Attention à la *casse* (majuscules / minuscules) !
- Les chaînes sont ordonnées :

La comparaison de chaînes de caractères : ==

- Attention à la *casse* (majuscules / minuscules) !
- Les chaînes sont ordonnées :
 - `phrase_1 < phrase_2`

La comparaison de chaînes de caractères : ==

- Attention à la *casse* (majuscules / minuscules) !
- Les chaînes sont ordonnées :
 - `phrase_1 < phrase_2`
 - `'A' < 'a' < 'à'`

La comparaison de chaînes de caractères : ==

- Attention à la *casse* (majuscules / minuscules) !
- 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.

La comparaison de chaînes de caractères : ==

- Attention à la *casse* (majuscules / minuscules) !
- 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.
 - Il est assez rare en pratique d'avoir à utiliser ce type de comparaison (contrairement aux comparaisons entre nombres)

Les opérateurs sur les chaînes de caractères

```
>>> phrase_1 = 'Bonjour'  
>>> phrase_2 = 'Bonjour'  
>>> phrase_1 == phrase_2  
True
```

Les opérateurs sur les chaînes de caractères

```
>>> phrase_1 = 'Bonjour'
```

```
>>> phrase_2 = 'Bonjour'
```

```
>>> phrase_1 == phrase_2
```

```
True
```

```
>>> 'Bonjour' > 'Bonjouq'
```

```
True
```


Les opérateurs sur les chaînes de caractères

```
>>> phrase_1 = 'Bonjour'
```

```
>>> phrase_2 = 'Bonjour'
```

```
>>> phrase_1 == phrase_2
```

```
True
```

```
>>> 'Bonjour' > 'Bonjouq'
```

```
True
```

```
>>> 'BOnjour' > 'Bonjouq'
```

```
False
```

Les opérateurs sur les chaînes de caractères

```
>>> phrase_1 = 'Bonjour'
```

```
>>> phrase_2 = 'Bonjour'
```

```
>>> phrase_1 == phrase_2
```

```
True
```

```
>>> 'Bonjour' > 'Bonjouq'
```

```
True
```

```
>>> 'BOnjour' > 'Bonjouq'
```

```
False
```

```
>>> ord('o')
```

```
111
```

Les opérateurs sur les chaînes de caractères

```
>>> phrase_1 = 'Bonjour'
```

```
>>> phrase_2 = 'Bonjour'
```

```
>>> phrase_1 == phrase_2
```

```
True
```

```
>>> 'Bonjour' > 'Bonjouq'
```

```
True
```

```
>>> 'BOnjour' > 'Bonjouq'
```

```
False
```

```
>>> ord('o')
```

```
111
```

```
>>> ord('0')
```

```
79
```

Les opérateurs sur les chaînes de caractères

```
>>> phrase_1 = 'Bonjour'
```

```
>>> phrase_2 = 'Bonjour'
```

```
>>> phrase_1 == phrase_2
```

```
True
```

```
>>> 'Bonjour' > 'Bonjouq'
```

```
True
```

```
>>> 'BOnjour' > 'Bonjouq'
```

```
False
```

```
>>> ord('o')
```

```
111
```

```
>>> ord('0')
```

```
79
```

```
>>> chr(80)
```

```
'P'
```

Les opérateurs sur les chaînes de caractères

Le parcours par une *itération* des caractères de l'index 0 à l'index $\text{len}(\text{chaîne}) - 1$

```
i = 0
```

```
while i < len(chaîne):
```

```
    print(chaîne[i])
```

```
    i = i + 1
```

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 opérateurs sur les chaînes de caractères

Le parcours par une *itération* des caractères de l'index 0 à l'index $\text{len}(\text{chaîne}) - 1$

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

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

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

Notez que la variable `caractere` prend, à chaque tour de boucle, la valeur du prochain caractère dans la chaîne. La boucle s'arrête automatiquement lorsque la chaîne a été complètement parcourue !

Les listes

Objectif

Découvrir le potentiel d'une telle structure de données.

- Type séquentiel ordonné, dénombrable

- Type séquentiel ordonné, dénombrable
- Type *mutable* : **on peut modifier une liste**

- Type séquentiel ordonné, dénombrable
- Type *mutable* : **on peut modifier une liste**
- Les éléments peuvent être hétérogènes

- Type séquentiel ordonné, dénombrable
- Type *mutable* : **on peut modifier une liste**
- Les éléments peuvent être hétérogènes
- Une liste peut même contenir d'autres listes !

- Type séquentiel ordonné, dénombrable
- Type *mutable* : **on peut modifier une liste**
- Les éléments peuvent être hétérogènes
- Une liste peut même contenir d'autres listes !
- On peut accéder à des éléments ou des sous-listes en utilisant les opérateurs [et]

- Type séquentiel ordonné, dénombrable
- Type *mutable* : **on peut modifier une liste**
- Les éléments peuvent être hétérogènes
- Une liste peut même contenir d'autres listes !
- On peut accéder à des éléments ou des sous-listes en utilisant les opérateurs [et]
- Le premier index de la liste est 0

- Type séquentiel ordonné, dénombrable
- Type *mutable* : **on peut modifier une liste**
- Les éléments peuvent être hétérogènes
- Une liste peut même contenir d'autres listes !
- On peut 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


```
nombres = [5, 38, 10, 25]
```

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

```
mots = ['jambon', '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])
```

```
mots = ['jambon', 'fromage', 'confiture', 'chocolat']
```

```
mots = ['jambon', 'fromage', 'confiture', 'chocolat']
```

- `len(mots)` donne le nombre d'éléments dans la liste ;

```
mots = ['jambon', '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 = ['jambon', '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]`

```
mots = ['jambon', '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, de même que pour les chaînes de caractères,

```
mots = ['jambon', '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, de même que pour les chaînes de caractères,
 - `mots[+i]` identifie le `(i+1)`-ième élément à partir du début

```
mots = ['jambon', '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, de même que pour les chaînes de caractères,
 - `mots[+i]` identifie le $(i+1)$ -ième élément à partir du début
 - `mots[-i]` identifie le i -ième élément à partir de la fin

- Ajout à la fin (ou concaténation) : l'opérateur +

- Ajout à la fin (ou concaténation) : l'opérateur +
- Comparaison élément par élément : l'opérateur ==

- Ajout à la fin (ou concaténation) : l'opérateur +
- Comparaison élément par élément : l'opérateur ==

- Ajout à la fin (ou concaténation) : l'opérateur +
- Comparaison élément par élément : l'opérateur ==

```
>>> mots = ['jambon', 'fromage', 'confiture', 'chocolat']  
>>> mots = mots + ['oeufs']  
>>> mots == ['jambon', 'fromage', 'confiture', 'chocolat', 'oeufs']  
True
```

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

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

```
>>> mots = ['jambon', 'fromage', 'confiture', 'chocolat']
>>> mots[2:2] = ['miel']
>>> mots
['jambon', 'fromage', 'miel', 'confiture', 'chocolat']
```

Une tranche ?

```
mots = [ 'jambon', 'fromage', 'confiture', 'chocolat' ]  
        ↑         ↑         ↑         ↑         ↑  
        0         1         2         3         4
```

Insertion dans une liste

```
mots = [ 'jambon', 'fromage', 'confiture', 'chocolat' ]  
        ↑         ↑         ↑         ↑         ↑  
        0         1         2         3         4
```

Insertion / remplacement :

```
>>> mots[2:2] = ['miel']
```

```
>>> mots
```

```
['jambon', 'fromage', 'miel', 'confiture', 'chocolat']
```


Remplacement dans une liste

```
mots = [ 'jambon', 'fromage', 'confiture', 'chocolat' ]  
        ↑         ↑         ↑         ↑         ↑  
        0         1         2         3         4
```

Insertion / remplacement :

```
>>> mots[1:3] = ['pain']  
>>> mots  
['jambon', 'pain', 'chocolat']
```

Retrait dans une liste

```
mots = [ 'jambon', 'fromage', 'confiture', 'chocolat' ]  
        ↑         ↑         ↑         ↑         ↑  
        0         1         2         3         4
```

Retrait :

```
>>> mots[2:4] = [] # La liste vide  
>>> mots  
['jambon', 'fromage']
```

Retrait dans une liste

```
mots = [ 'jambon', 'fromage', 'confiture', 'chocolat' ]  
        ↑         ↑         ↑         ↑         ↑  
        0         1         2         3         4
```

Retrait (2) :

```
>>> del mots[1]  
>>> mots  
['jambon', 'confiture', 'chocolat']
```

Bornes d'une tranche

```
mots = [ 'jambon', '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 = [ 'jambon', '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 = [ 'jambon', 'fromage', 'confiture', 'chocolat' ]  
        ↑         ↑         ↑         ↑         ↑  
        0         1         2         3         4
```

```
>>> for element in mots:  
...     print(element)  
...  
jambon  
fromage  
confiture  
chocolat
```

- `range(n)`

1. Crée en fait un *générateur*, mais pour les besoins de notre cours nous l'utiliserons comme une liste. Pour avoir une vraie liste, utilisez `list(range(n))`

- `range(n)`
 - Crée une liste¹ dont les éléments sont les entiers compris entre 0 et $n - 1$

1. Crée en fait un *générateur*, mais pour les besoins de notre cours nous l'utiliserons comme une liste. Pour avoir une vraie liste, utilisez `list(range(n))`

- `range(n)`
 - Crée une liste¹ dont les éléments sont les entiers compris entre 0 et $n - 1$
 - `n` doit être un entier positif ou nul

1. Crée en fait un *générateur*, mais pour les besoins de notre cours nous l'utiliserons comme une liste. Pour avoir une vraie liste, utilisez `list(range(n))`

- `range(n)`
 - Crée une liste¹ 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

1. Crée en fait un *générateur*, mais pour les besoins de notre cours nous l'utiliserons comme une liste. Pour avoir une vraie liste, utilisez `list(range(n))`

- `range(n)`
 - Crée une liste¹ 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)`

1. Crée en fait un *générateur*, mais pour les besoins de notre cours nous l'utiliserons comme une liste. Pour avoir une vraie liste, utilisez `list(range(n))`

- `range(n)`
 - Crée une liste¹ 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`

1. Créé en fait un *générateur*, mais pour les besoins de notre cours nous l'utiliserons comme une liste. Pour avoir une vraie liste, utilisez `list(range(n))`

- `range(n)`
 - Crée une liste¹ 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`

1. Créé en fait un *générateur*, mais pour les besoins de notre cours nous l'utiliserons comme une liste. Pour avoir une vraie liste, utilisez `list(range(n))`

- `range(n)`
 - Crée une liste¹ 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`**

1. Créé en fait un *générateur*, mais pour les besoins de notre cours nous l'utiliserons comme une liste. Pour avoir une vraie liste, utilisez `list(range(n))`

- `range(n)`
 - Crée une liste¹ 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)`

1. Crée en fait un *générateur*, mais pour les besoins de notre cours nous l'utiliserons comme une liste. Pour avoir une vraie liste, utilisez `list(range(n))`

- `max(liste), min(liste)`

- `max(liste), min(liste)`
 - Retourne le maximum ou le minimum de la liste

- `max(liste), min(liste)`
 - Retourne le maximum ou le minimum de la liste
 - Dépend de la nature des éléments de la liste

- `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 paire à paire

- `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 paire à paire
- `sum(list)`

- `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 paire à paire
- `sum(list)`
 - Retourne la somme des éléments de la liste

- `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 paire à paire
- `sum(list)`
 - Retourne la somme des éléments de la liste
 - Les éléments doivent pouvoir être additionnés, avec un total dont la valeur initiale est 0 (`int`).

Attention !

Une liste est une structure de donnée trop volumineuse pour être copiée par affectation ou passage d'argument. Comme les chaînes de caractères, on travaille avec une *référence*.

- `liste_1 = [1, 2, 3]`
- `liste_2 = liste_1`
- `liste_2[2:3] = [30]`
- `liste_1[0:1] = [10]`

542

660

Nom	Adresse	Contenu	Type
...	13202
...	13203
...	13204
...	13205

Attention !

Une liste est une structure de donnée trop volumineuse pour être copiée par affectation ou passage d'argument. Comme les chaînes de caractères, on travaille avec une *référence*.

- `liste_1 = [1, 2, 3]`
- `liste_2 = liste_1`
- `liste_2[2:3] = [30]`
- `liste_1[0:1] = [10]`

542

660

1	-	2	-	3	\0		

Nom	Adresse	Contenu	Type
...	13202
...	13203
...	13204
...	13205

Attention !

Une liste est une structure de donnée trop volumineuse pour être copiée par affectation ou passage d'argument. Comme les chaînes de caractères, on travaille avec une *référence*.

- `liste_1 = [1, 2, 3]`
- `liste_2 = liste_1`
- `liste_2[2:3] = [30]`
- `liste_1[0:1] = [10]`

542

660

1	-	2	-	3	\0		

Nom	Adresse	Contenu	Type
...	13202
liste_1	13203
...	13204
...	13205

Attention !

Une liste est une structure de donnée trop volumineuse pour être copiée par affectation ou passage d'argument. Comme les chaînes de caractères, on travaille avec une *référence*.

- `liste_1 = [1, 2, 3]`
- `liste_2 = liste_1`
- `liste_2[2:3] = [30]`
- `liste_1[0:1] = [10]`

542

660

1	-	2	-	3	\0		

Nom	Adresse	Contenu	Type
...	13202
liste_1	13203	542	*list
...	13204
...	13205

Attention !

Une liste est une structure de donnée trop volumineuse pour être copiée par affectation ou passage d'argument. Comme les chaînes de caractères, on travaille avec une *référence*.

- `liste_1 = [1, 2, 3]`
- `liste_2 = liste_1`
- `liste_2[2:3] = [30]`
- `liste_1[0:1] = [10]`

542

660

1	-	2	-	3	\0		

Nom	Adresse	Contenu	Type
...	13202
liste_1	13203	542	*list
...	13204
...	13205

Attention !

Une liste est une structure de donnée trop volumineuse pour être copiée par affectation ou passage d'argument. Comme les chaînes de caractères, on travaille avec une *référence*.

- `liste_1 = [1, 2, 3]`
- `liste_2 = liste_1`
- `liste_2[2:3] = [30]`
- `liste_1[0:1] = [10]`

542

660

1	-	2	-	3	\0		

Nom	Adresse	Contenu	Type
...	13202
liste_1	13203	542	*list
liste_2	13204
...	13205

Attention !

Une liste est une structure de donnée trop volumineuse pour être copiée par affectation ou passage d'argument. Comme les chaînes de caractères, on travaille avec une *référence*.

- `liste_1 = [1, 2, 3]`
- `liste_2 = liste_1`
- `liste_2[2:3] = [30]`
- `liste_1[0:1] = [10]`

542

660

1	-	2	-	3	\0		

Nom	Adresse	Contenu	Type
...	13202
liste_1	13203	542	*list
liste_2	13204	542	*list
...	13205

Attention !

Une liste est une structure de donnée trop volumineuse pour être copiée par affectation ou passage d'argument. Comme les chaînes de caractères, on travaille avec une *référence*.

- `liste_1 = [1, 2, 3]`
- `liste_2 = liste_1`
- `liste_2[2:3] = [30]`
- `liste_1[0:1] = [10]`

542

660

1	-	2	-	3	\0		

Nom	Adresse	Contenu	Type
...	13202
liste_1	13203	542	*list
liste_2	13204	542	*list
...	13205

Attention !

Une liste est une structure de donnée trop volumineuse pour être copiée par affectation ou passage d'argument. Comme les chaînes de caractères, on travaille avec une *référence*.

- `liste_1 = [1, 2, 3]`
- `liste_2 = liste_1`
- `liste_2[2:3] = [30]`
- `liste_1[0:1] = [10]`

542

660

1	-	2	-	30	\0		

Nom	Adresse	Contenu	Type
...	13202
liste_1	13203	542	*list
liste_2	13204	542	*list
...	13205

Attention !

Une liste est une structure de donnée trop volumineuse pour être copiée par affectation ou passage d'argument. Comme les chaînes de caractères, on travaille avec une *référence*.

- `liste_1 = [1, 2, 3]`
- `liste_2 = liste_1`
- `liste_2[2:3] = [30]`
- `liste_1[0:1] = [10]`

542

660

1	-	2	-	30	\0		

Nom	Adresse	Contenu	Type
...	13202
liste_1	13203	542	*list
liste_2	13204	542	*list
...	13205

Attention !

Une liste est une structure de donnée trop volumineuse pour être copiée par affectation ou passage d'argument. Comme les chaînes de caractères, on travaille avec une *référence*.

- `liste_1 = [1, 2, 3]`
- `liste_2 = liste_1`
- `liste_2[2:3] = [30]`
- `liste_1[0:1] = [10]`

542	10	-	2	-	30	\0		
660								

Nom	Adresse	Contenu	Type
...	13202
liste_1	13203	542	*list
liste_2	13204	542	*list
...	13205

Attention !

Une liste est une structure de donnée trop volumineuse pour être copiée par affectation ou passage d'argument. Comme les chaînes de caractères, on travaille avec une *référence*.

- `liste_1 = [1, 2, 3]`
- `liste_2 = liste_1`
- `liste_2[2:3] = [30]`
- `liste_1[0:1] = [10]`

542

660

10	-	2	-	30	\0		

Nom	Adresse	Contenu	Type
...	13202
liste_1	13203	542	*list
liste_2	13204	542	*list
...	13205

Si on a besoin d'une copie d'une liste, programmons une fonction !

Copie d'une liste

Si on a besoin d'une copie d'une liste, programmons une fonction !

```
def copie_liste(liste_entree):  
    liste_sortie = []  
    i = 0  
    for element in liste_entree:  
        liste_sortie[i:i] = [element]  
        # Ou bien : liste_sortie[i:] = [element]  
        # Ou bien : liste_sortie.append(element)  
        #             on y reviendra!  
        i = i + 1  
  
    return liste_sortie
```

Ou mieux, utilisons le *constructeur* de la structure de donnée `list` (nous y reviendrons !), ou bien une tranche, qui fait également une copie.

Ou mieux, utilisons le *constructeur* de la structure de donnée `list` (nous y reviendrons !), 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

Objectif

Ajouter une autre structure de données utile à notre coffre à outils.

- Type **non ordonné**, dénombrable

- Type **non ordonné**, dénombrable
- Type mutable : on peut modifier un dictionnaire

- 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>

- 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

- 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

- 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 d'autres 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
```


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'
```

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
```

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'}
```

Les opérateurs sur les dictionnaires

```
t = {'mouse': 'souris', 'computer': 'ordinateur',  
     'keyboard': 'clavier'}
```

Obtenir une liste des clés :

```
>>> list(t.keys())  
['mouse', 'computer', 'keyboard']
```

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']
```

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

Objectif

Se familiariser avec les fichiers, c'est à dire les ensembles de données stockées sur un support externe, contenant soit des caractères (pour la représentation de documents textuels), soit des données diverses.

- Permettent de conserver de l'information, tel les bases de données

- Permettent de conserver de l'information, tel les bases de données
- Permettent d'échanger de l'information

- Permettent de conserver de l'information, tel les bases de données
- Permettent d'échanger de l'information
- On peut modifier un fichier

- 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

- 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 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 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')
```

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...
```

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 d'un fichier

```
f_1 = open('nom.txt', 'r')
```

```
f_2 = open('nom_2.txt', 'w')
```

Support externe :

116052	B	o	n	j	o	u	r	\n
116060	O	k	\n	B	o	n	j	o
	u	r	O	k	\n	EOF		
118000								
118008								

Nom	Adresse	Contenu	Type
...	1202
...	1203
...	1204
...	1205

Ouverture d'un fichier

```
f_1 = open('nom.txt', 'r')
```

```
f_2 = open('nom_2.txt', 'w')
```

Support externe :

116052	B	o	n	j	o	u	r	\n
116060	O	k	\n	B	o	n	j	o
	u	r	O	k	\n	EOF		
118000								
118008								

Nom	Adresse	Contenu	Type
...	1202
...	1203
...	1204
...	1205

Ouverture d'un fichier

```
f_1 = open('nom.txt', 'r')
```

```
f_2 = open('nom_2.txt', 'w')
```

Support externe :

116052	B	o	n	j	o	u	r	\n
116060	O	k	\n	B	o	n	j	o
	u	r	O	k	\n	EOF		
118000								
118008								

Nom	Adresse	Contenu	Type
...	1202
f_1	1203
...	1204
...	1205

Ouverture d'un fichier

```
f_1 = open('nom.txt', 'r')
```

```
f_2 = open('nom_2.txt', 'w')
```

Support externe :

116052	B	o	n	j	o	u	r	\n
116060	O	k	\n	B	o	n	j	o
	u	r	O	k	\n	EOF		
118000								
118008								

Nom	Adresse	Contenu	Type
...	1202
f_1	1203	116052	*str
...	1204
...	1205

Ouverture d'un fichier

```
f_1 = open('nom.txt', 'r')
```

```
f_2 = open('nom_2.txt', 'w')
```

Support externe :

116052	B	o	n	j	o	u	r	\n
116060	O	k	\n	B	o	n	j	o
	u	r	O	k	\n	EOF		
118000								
118008								

Nom	Adresse	Contenu	Type
...	1202
f_1	1203	116052	*str
...	1204
...	1205

Ouverture d'un fichier

```
f_1 = open('nom.txt', 'r')
```

```
f_2 = open('nom_2.txt', 'w')
```

Support externe :

116052	B	o	n	j	o	u	r	\n
116060	O	k	\n	B	o	n	j	o
	u	r	O	k	\n	EOF		
118000								
118008								

Nom	Adresse	Contenu	Type
...	1202
f_1	1203	116052	*str
f_2	1204
...	1205

Ouverture d'un fichier

```
f_1 = open('nom.txt', 'r')
```

```
f_2 = open('nom_2.txt', 'w')
```

Support externe :

116052	B	o	n	j	o	u	r	\n
116060	O	k	\n	B	o	n	j	o
	u	r	O	k	\n	EOF		
118000								
118008								

Nom	Adresse	Contenu	Type
...	1202
f_1	1203	116052	*str
f_2	1204	118000	*str
...	1205

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

Lecture et écriture de chaînes de caractères

```
# 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)
```

Lecture et écriture de chaînes de caractères

```
# 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!")
```

Lecture et écriture de chaînes de caractères

```
# 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 !

```
ma_chaine = f_1.readline()
```

```
mon_car = f_1.read(1)
```

Mémoire :

904

910

Support externe :

116052	B	o	n	j	o	u	r	\n
116060	O	k	\n	B	o	n	j	o
	u	r	O	k	\n	EOF		
118000								
118008								

Nom	Adresse	Contenu	Type
f_1	1203	116052	*str
f_2	1204	118000	*str
...	1205
...	1206


```
ma_chaine = f_1.readline()
```

```
mon_car = f_1.read(1)
```

Mémoire :

904

910

Support externe :

116052	B	o	n	j	o	u	r	\n
116060	O	k	\n	B	o	n	j	o
	u	r	O	k	\n	EOF		
118000								
118008								

Nom	Adresse	Contenu	Type
f_1	1203	116052	*str
f_2	1204	118000	*str
...	1205
...	1206

```
ma_chaine = f_1.readline()
```

```
mon_car = f_1.read(1)
```

Mémoire :

904

910

Support externe :

116052

116060

B	o	n	j	o	u	r	\n
O	k	\n	B	o	n	j	o
u	r	O	k	\n	EOF		

118000

118008

Nom	Adresse	Contenu	Type
f_1	1203	116052	*str
f_2	1204	118000	*str
...	1205
...	1206

```
ma_chaine = f_1.readline()
```

```
mon_car = f_1.read(1)
```

Mémoire :

904	B	o	n	j	o	u
910	r	\n	\0			

Support externe :

116052	B	o	n	j	o	u	r	\n
116060	O	k	\n	B	o	n	j	o
	u	r	O	k	\n	EOF		
118000								
118008								

Nom	Adresse	Contenu	Type
f_1	1203	116052	*str
f_2	1204	118000	*str
...	1205
...	1206

```
ma_chaine = f_1.readline()
```

```
mon_car = f_1.read(1)
```

Mémoire :

904	B	o	n	j	o	u
910	r	\n	\0			

Support externe :

116052	B	o	n	j	o	u	r	\n
116060	O	k	\n	B	o	n	j	o
	u	r	O	k	\n	EOF		
118000								
118008								

Nom	Adresse	Contenu	Type
f_1	1203	116052	*str
f_2	1204	118000	*str
ma_chaine	1205
...	1206

```
ma_chaine = f_1.readline()
```

```
mon_car = f_1.read(1)
```

Mémoire :

904	B	o	n	j	o	u
910	r	\n	\0			

Support externe :

116052	B	o	n	j	o	u	r	\n
116060	O	k	\n	B	o	n	j	o
	u	r	O	k	\n	EOF		
118000								
118008								

Nom	Adresse	Contenu	Type
f_1	1203	116052	*str
f_2	1204	118000	*str
ma_chaine	1205	904	*str
...	1206

```
ma_chaine = f_1.readline()
```

```
mon_car = f_1.read(1)
```

Mémoire :

904	B	o	n	j	o	u
910	r	\n	\0			

Support externe :

116052	B	o	n	j	o	u	r	\n
116060	O	k	\n	B	o	n	j	o
	u	r	O	k	\n	EOF		
118000								
118008								

Nom	Adresse	Contenu	Type
f_1	1203	116052	*str
f_2	1204	118000	*str
ma_chaine	1205	904	*str
...	1206

```
ma_chaine = f_1.readline()
```

```
mon_car = f_1.read(1)
```

Mémoire :

904	B	o	n	j	o	u
910	r	\n	\0	O	\0	

Support externe :

116052	B	o	n	j	o	u	r	\n
116060	O	k	\n	B	o	n	j	o
	u	r	O	k	\n	EOF		
118000								
118008								

Nom	Adresse	Contenu	Type
f_1	1203	116052	*str
f_2	1204	118000	*str
ma_chaine	1205	904	*str
...	1206

```
ma_chaine = f_1.readline()
```

```
mon_car = f_1.read(1)
```

Mémoire :

904	B	o	n	j	o	u
910	r	\n	\0	O	\0	

Support externe :

116052	B	o	n	j	o	u	r	\n
116060	O	k	\n	B	o	n	j	o
	u	r	O	k	\n	EOF		
118000								
118008								

Nom	Adresse	Contenu	Type
f_1	1203	116052	*str
f_2	1204	118000	*str
ma_chaine	1205	904	*str
mon_car	1206


```
ma_chaine = f_1.readline()
```

```
mon_car = f_1.read(1)
```

Mémoire :

904	B	o	n	j	o	u
910	r	\n	\0	O	\0	

Support externe :

116052	B	o	n	j	o	u	r	\n
116060	O	k	\n	B	o	n	j	o
	u	r	O	k	\n	EOF		
118000								
118008								

Nom	Adresse	Contenu	Type
f_1	1203	116052	*str
f_2	1204	118000	*str
ma_chaine	1205	904	*str
mon_car	1206	913	*str

```
f_2.write('Désolé!')
```

```
f_2.write(ma_chaine)
```

```
f_2.write('\nfini!')
```

Mémoire :

904	B	o	n	j	o	u
910	r	\n	\0	O	\0	

Support externe :

116052	B	o	n	j	o	u	r	\n
116060	O	k	\n	B	o	n	j	o
	u	r	O	k	\n	EOF		
118000								
118008								

Nom	Adresse	Contenu	Type
f_1	1203	116052	*str
f_2	1204	118000	*str
ma_chaine	1205	904	*str
mon_car	1206	913	*str

```
f_2.write('Désolé!')
```

```
f_2.write(ma_chaine)
```

```
f_2.write('\nfini!')
```

Mémoire :

904	B	o	n	j	o	u
910	r	\n	\0	O	\0	

Support externe :

116052	B	o	n	j	o	u	r	\n
116060	O	k	\n	B	o	n	j	o
	u	r	O	k	\n	EOF		
118000								
118008								

Nom	Adresse	Contenu	Type
f_1	1203	116052	*str
f_2	1204	118000	*str
ma_chaine	1205	904	*str
mon_car	1206	913	*str

```
f_2.write('Désolé!')
```

```
f_2.write(ma_chaine)
```

```
f_2.write('\nfini!')
```

Mémoire :

904	B	o	n	j	o	u
910	r	\n	\0	0	\0	

Support externe :

116052	B	o	n	j	o	u	r	\n
116060	0	k	\n	B	o	n	j	o
	u	r	0	k	\n	EOF		
118000	D	é	s	o	l	é	!	
118008								

Nom	Adresse	Contenu	Type
f_1	1203	116052	*str
f_2	1204	118000	*str
ma_chaine	1205	904	*str
mon_car	1206	913	*str

```
f_2.write('Désolé!')
```

```
f_2.write(ma_chaine)
```

```
f_2.write('\nfini!')
```

Mémoire :

904	B	o	n	j	o	u
910	r	\n	\0	O	\0	

Support externe :

116052	B	o	n	j	o	u	r	\n
116060	O	k	\n	B	o	n	j	o
	u	r	O	k	\n	EOF		
118000	D	é	s	o	l	é	!	B
118008	o	n	j	o	u	r	\n	

Nom	Adresse	Contenu	Type
f_1	1203	116052	*str
f_2	1204	118000	*str
ma_chaine	1205	904	*str
mon_car	1206	913	*str

```
f_2.write('Désolé!')
```

```
f_2.write(ma_chaine)
```

```
f_2.write('\nfini!')
```

Mémoire :

904	B	o	n	j	o	u
910	r	\n	\0	O	\0	

Support externe :

116052	B	o	n	j	o	u	r	\n
116060	O	k	\n	B	o	n	j	o
	u	r	O	k	\n	EOF		
118000	D	é	s	o	l	é	!	B
118008	o	n	j	o	u	r	\n	\n
	f	i	n	i	!			

Nom	Adresse	Contenu	Type
f_1	1203	116052	*str
f_2	1204	118000	*str
ma_chaine	1205	904	*str
mon_car	1206	913	*str

```
f_1.close()
```

```
f_2.close()
```

Mémoire :

904	B	o	n	j	o	u
910	r	\n	\0	O	\0	

Support externe :

116052	B	o	n	j	o	u	r	\n
116060	O	k	\n	B	o	n	j	o
	u	r	O	k	\n	EOF		
118000	D	é	s	o	l	é	!	B
118008	o	n	j	o	u	r	\n	\n
	f	i	n	i	!			

Nom	Adresse	Contenu	Type
f_1	1203	116052	*str
f_2	1204	118000	*str
ma_chaine	1205	904	*str
mon_car	1206	913	*str

`f_1.close()`

`f_2.close()`

Mémoire :

904	B	o	n	j	o	u
910	r	\n	\0	O	\0	

Support externe :

116052	B	o	n	j	o	u	r	\n
116060	O	k	\n	B	o	n	j	o
	u	r	O	k	\n	EOF		
118000	D	é	s	o	l	é	!	B
118008	o	n	j	o	u	r	\n	\n
	f	i	n	i	!			

Nom	Adresse	Contenu	Type
<code>f_1</code>	1203	116052	<code>*str</code>
<code>f_2</code>	1204	118000	<code>*str</code>
<code>ma_chaine</code>	1205	904	<code>*str</code>
<code>mon_car</code>	1206	913	<code>*str</code>


```
f_1.close()
```

```
f_2.close()
```

Mémoire :

904	B	o	n	j	o	u
910	r	\n	\0	O	\0	

Support externe :

116052	B	o	n	j	o	u	r	\n
116060	O	k	\n	B	o	n	j	o
	u	r	O	k	\n	EOF		
118000	D	é	s	o	l	é	!	B
118008	o	n	j	o	u	r	\n	\n
	f	i	n	i	!	EOF		

Nom	Adresse	Contenu	Type
f_1	1203	116052	*str
f_2	1204	118000	*str
ma_chaine	1205	904	*str
mon_car	1206	913	*str

```
f_1.close()
```

```
f_2.close()
```

Mémoire :

904	B	o	n	j	o	u
910	r	\n	\0	O	\0	

Support externe :

116052	B	o	n	j	o	u	r	\n
116060	O	k	\n	B	o	n	j	o
	u	r	O	k	\n	EOF		
118000	D	é	s	o	l	é	!	B
118008	o	n	j	o	u	r	\n	\n
	f	i	n	i	!	EOF		

Nom	Adresse	Contenu	Type
f_1	1203	116052	*str
f_2	1204	118000	*str
ma_chaine	1205	904	*str
mon_car	1206	913	*str

- Les chaînes de caractères et les fichiers textes seront utiles pour permettre à nos applications de conserver et de s'échanger de l'information

- Les chaînes de caractères et les fichiers textes seront utiles pour permettre à nos applications de conserver et de s'échanger de l'information
- Les listes et les dictionnaires sont des outils extrêmement puissants que nous allons utiliser afin de développer des applications relativement complexes assez rapidement

Lectures et travaux dirigés

- Chapitres 9 et 10 de G. Swinnen
- Travaux dirigés : Utilisation de ces structures de données dans des applications plus complexes

Questions ?