

2N6 Programmation 2



pythonTM

Travailler avec des fichiers et CSVs



Rappel

f-strings



retour



```
1 salutation = "Bonjour"
2 nom = "Gallant"
3 prenom = "Pierre-Paul"
4
5 print(f"{salutation} Mr.{prenom} {nom} au cours 2N6 pour réseautique")
```

Le f-string commence par un **f** suivie du début du string (" ou ')

La valeur des variables peut être utilisé directement en mettant le nom de la variable entre { }

On peut y mettre du texte comme dans un string normal

Le f-string termine par le même guillemet (" ou ')

Ex de f-string



retour



```
/* ex_f-strings.py > ...
```

```
1 salutation = "Bonjour et bienvenue"
```

```
2 nom = "gallant"
```

```
3 prenom = "Pierre-Paul"
```

```
4
```

```
5 print(f"\n{salutation} Mr.{prenom} {nom.capitalize()} au cours 2N6 pour réseautique.\n")
```

PROBLÈMES

SORTIE

CONSOLE DE DÉBOGAGE

TERMINAL

.NET INTERACTIVE

JUPYTER

AZURE

COMMENTS

```
PS C:\> & "C:/Program Files/Python310/python.exe" "c:/Users/pierre-paul.gallant/OneDrive - Cégep Édouard-Montpeti
```

```
Bonjour et bienvenue Mr.Pierre-Paul Gallant au cours 2N6 pour réseautique.
```

```
PS C:\> █
```



Boucles for et listes



retour



- > Itère sur la liste. La variable `cours` prend la valeur de chacun des objets dans la liste les uns après les autres.
- > Itère sur la liste.
La fonction `range()` avec `len()` nous donnent une liste numérique de 0 à la valeur de la longueur de la liste.
- > Ici, `index` prends les valeurs 0,1,2,3 puis 4

```
liste_cours = ['Programmation 1', 'Math',  
|...|...|...|...'Bureautique', 'Réseau 1', 'Math']
```

```
for cours in liste_cours:  
|...|print(cours)
```

```
Programmation 1  
Math  
Bureautique  
Réseau 1  
Math
```

```
for index in range(len(liste_cours)):  
|...|print(index, liste_cours[index])
```

```
0 Programmation 1  
1 Math  
2 Bureautique  
3 Réseau 1  
4 Math
```

La boucle for standard



retour



Variable créée dans la boucle for. Sa valeur change dans chaque itération de la boucle.

```
1 liste_cours = ["Prog 1", "Math", "Bureautique"]
2
3 for cours in liste_cours :
4     print("Bienvenue au cours :")
5     print(cours)
```

Liste sur laquelle on itère.

- > Lorsque qu'on arrive à ligne 3 dans cet exemple. La variable "cours" devient équivalente à la première valeur contenue dans la liste "liste_cours". (for cours in liste_cours)
- > Dans chaque itération de la boucle, la valeur de "cours" change et devient la prochaine valeur dans la liste.



La boucle for sur l'index



retour



C#

```
1 var les_cours= new List<string>{"Prog","Math","Res"};  
2  
3 for (int i=0; i < les_cours.length(); i++) {  
4     les_cours[i] = "cours " + les_cours[i] ;  
5     Console.WriteLine(les_cours[i]) ;  
6 }
```

`les_cours.length() == 3`

`i` commence à `i == 0`

`i++` → `i == 1`

`i++` → `i == 2`

Python

```
1 les_cours = ["Prog 1", "Math", "Res"]  
2  
3 for i in range(len(les_cours)) :  
4     les_cours[i] = "cours " + les_cours[i]  
5     print(les_cours[i])  
6
```

`len(les_cours) == 3`

`range(3)` → `[0,1,2]`

`i` prend chaque valeur dans la liste donc :

`i == 0`

Puis, `i == 1`

Puis, `i == 2`





Fichiers et Répertoires

Comment interagir avec les fichiers avec python



La fonction open()

- > Permet facilement d'interagir avec des fichiers
- > Peut créer des fichiers ou ouvrir des fichiers existants
- > Deux paramètres essentiels :

```
file = open("status.conf", "w")
```

Mode d'ouverture

Fichier à ouvrir

Variable contenant un objet correspondant au fichier créé. Permet d'interagir facilement avec ce fichier.



La fonction open()

> Les modes d'ouverture :

```
file = open("status.conf", "w")
```

- r Ouvre pour lecture seulement.
- w Ouvre pour écriture, crée le fichier si nécessaire.
Écrase fichier existant. (Ne garde pas ce qui est déjà dans le fichier.)
- x Crée un nouveau fichier et l'ouvre pour écriture.
Échoue si le fichier existe déjà.
- a Ouvre pour écriture MAIS concatène à la fin si le fichier existe.



La fonction open()

- > open() retourne un objet correspondant au fichier ouvert avec ses propres méthodes.

```
1  with open("status.conf","r") as fichier :
2      ....print( fichier.read() ).....#lis le reste du contenu du fichier
3      ....print( fichier.read(10) )...#lis les 10 caract`res suivants dans le fichier
4      ....print( fichier.readline() ) # lis la prochaine ligne
5      ....print( fichier.readlines() )# lis toutes les lignes du fichier et retourne une liste
6      ....print( fichier.readable() ) # retourne booléen
7      ....
8  with open("status.conf","w") as fichier :
9      ....fichier.write("texte").....#écrit "texte" dans le fichier
10     ....print( fichier.writable() ) #retourne booléen
11     ....
12     ....# peu importe comment on crée l'objet file on peut :
13     ....fichier.seek(7).....#place le pointeur au 7ième caractère du fichier
14     ....fichier.seek(0).....#retourne le pointeur au début du fichier
15     ....fichier.tell().....#indique la position
16     ....
17     ....fichier.close().....#ferme le fichier
```



L'instruction "with"

- > Un nouveau contrôle de flux propre à python
- > Très utilisé pour l'ouverture / traitement de fichiers

```
with_test.py > ...  
1 file = open("status.conf", "w")  
2 file.write("Mise à jour en cours\n")  
3 file.close()
```

- > L'ouverture de fichiers nécessite sa fermeture
- > Sinon risque de **perte de données**

Excellent, sécuritaire

```
with_test.py > ...  
1 with open("status.conf","w") as file :  
2     file.write("Mise à terminer\n")  
3
```

- > Fermeture automatique du fichier ouvert
- > Assignment d'une variable d'une portée limité.
- > Clarté du code



L'instruction "with"

- > À moins de circonstances extraordinaires, on utilise tout le temps l'instruction « with » lorsqu'on utilise la fonction « open() »
- > Ici, on crée un nouveau fichier vide avec open() et on le ferme immédiatement en sortant de l'instruction "with"

Excellent, sécuritaire

```
2  with open("test.txt", "w") as file:  
3      ...pass
```



- > Fait partie de la librairie standard
- > Permet d'accéder aux données et d'interagir avec l'os

os_exemple.py > ...

```
1 import os
2 info_env = os.environ
3 home = info_env.get("HOME")
4 os.mkdir(f"{home}/scripts2")
5 os.chdir(f"{home}/scripts2")
6 os.makedirs("resultats/recursif")
7 print(os.getcwd())
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
• pierre-paul@pp-vm:~/scripts$ /bin/python3 /home/
/home/pierre-paul/scripts2
• pierre-paul@pp-vm:~/scripts$ tree ../scripts2
../scripts2
├── resultats
│   └── recursif
```

2 directories, 0 files

- > `os.environ` : retourne un objet contenant les variables d'environnement (qu'on peut obtenir avec `get()`)
- > `os.mkdir()` : créer un répertoire
- > `os.chdir()` : change le répertoire courant
- > `os.makedirs()` : crée plusieurs répertoires de façon récursive
- > `os.getcwd()` : retourne un str indiquant le répertoire courant
- > Voir cours précédent pour plus de détails sur les fonctions du module os



Le sous-module path

- Contient de nombreuses fonctions spécifiques à la manipulations de path

```
19 # différentes fonctions du sous-module path. Le fichier dem
20 print(os.path.basename(f'{chemin}/demo4.txt'))--# 'demo4.txt'
21 print(os.path.dirname(f'{chemin}/demo4.txt'))---# 'c:\\Users\\.....\\R04_demo'
22 print(os.path.split(f'{chemin}/demo4.txt'))-----# ('c:\\Users\\.....\\R04_demo', 'demo4.txt')
23 print(os.path.exists(f'{chemin}/demo4.txt'))----# True
24 print(os.path.isfile(f'{chemin}/demo4.txt'))----# True
25 print(os.path.isdir(f'{chemin}/demo4.txt'))-----# False
26 print(os.path.splitext(f'{chemin}/demo4.txt'))--#('c:\\Users\\.....\\R04_demo', '.txt')#
```



Interopérabilité



« **L'interopérabilité** est la capacité que possède un produit ou un système, dont les interfaces sont intégralement connues, à fonctionner avec d'autres produits ou systèmes existants ou futurs et ce sans restriction d'accès ou de mise en œuvre. »

Source: [Wikipédia](#)



Travailler avec les CSVs

- CSV : un fichier texte dont les données sont délimitées par un caractère spécial (virgule par défaut).
- La première ligne contient les en-têtes de colonnes.

```
No étudiant;Groupe;Nom de l'étudiant;Prénom de l'étudiant;Prog.  
2273383;1010;Carrier;Alexandre;420.BU  
2222119;1010;Dion;Paul;420.BB  
2229304;1010;Douida;Wissale;420.BA  
2218593;1010;El Ammari;Amar;420.BA
```

- Exemple où le 'délimiter' est un ;

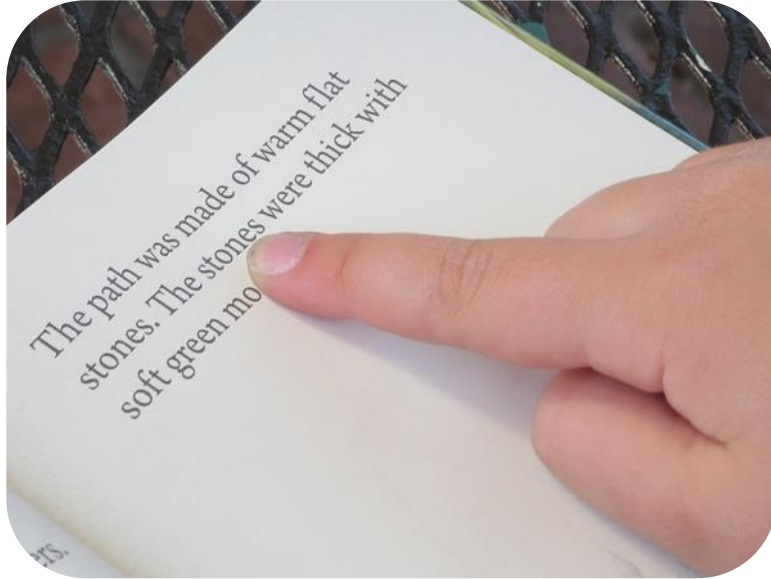


Travailler avec les CSVs

- › On utilise le module csv, conçu spécifiquement pour travailler de façon efficace avec des fichiers CSVs

```
3  import csv
4
5  with open('etudiants.csv', 'r', encoding='utf-8') as csv_file:
6      ....csv_reader = csv.reader(csv_file)
7      ....# saute la première ligne #
8      ....next(csv_reader)
9      ....for line in csv_reader:
10         ....|....print(line)
11         ....|....
```

Lire les CSVs



- Pour lire un csv. On commence par instancier un objet à partir de la classe « reader » avec le fichier qu'on désire lire en paramètre.
- Cet objet est itérable et peut donc être utilisé avec une boucle for.

```
3 import csv
4
5 with open('etudiants.csv', 'r', encoding='utf-8') as csv_file:
6     csv_reader = csv.reader(csv_file)
7     # saute la première ligne #
8     next(csv_reader)
9     for line in csv_reader:
10         print(line)
11
```

- On peut le comparer à un pointeur qui indique où nous sommes rendu dans le fichier qu'on lis

Écrire un CSV



- Le module csv permet aussi créer facilement des CSVs à partir de nos données
- On crée un objet à partir de la classe « reader » avec le fichier dans lequel on veut écrire en paramètre.
- Ici, on itère sur la structure de données contenant l'information à écrire.

```
1 import csv
2 liste_employers = [{"steve", "E211"}, {"Ana", "A809"}, {"Jon", "B32"}, {"Sophie", "N\\A"}]
3
4 with open('listeEmployes.csv', 'w', encoding='utf-8') as csv_file:
5     .... csv_writer = csv.writer(csv_file, delimiter = ';', lineterminator="\n")
6     .... csv_writer.writerow(["Employé", "Bureau"])
7     .... for line in liste_employers:
8     ....     csv_writer.writerow(line)
```