



# Index partie A

- > [Qu'est-ce que Python ?](#)
- > [Outils pour commencer](#)
- > [Les bases : \(types, opérateurs, structures, entrées/sorties\)](#)
- > [Les listes](#)
  - > [Boucles for et listes](#)
- > [Débogage avec un IDE](#)
- > [Approfondissement des strings](#)
- > [Fichiers et Répertoires](#)
- > [Travailler avec les CSVs](#)
- > [Les fonctions](#)
- > [Range / scope des variables](#)
- > [Dictionnaire](#)
- > [JSON](#)
- > [API Web \( requêtes GET \)](#)
- > [Modules](#)

# 2N6 Programmation 2



python™



## Rencontre 1:

- Qu'est-ce que python,
- Outils,
- Bases,
- Variables,
- Operateurs,
- Structures de contrôle



# Index

> Qu'est-ce que Python	3
> Outils pour commencer	11
> Les bases en python	19
> Types de données	21
> Opérateurs	26
> Structures de contrôle	29
> Entrée et Sortie de données	31
> Annexe	41





# Qu'est-ce que Python ?

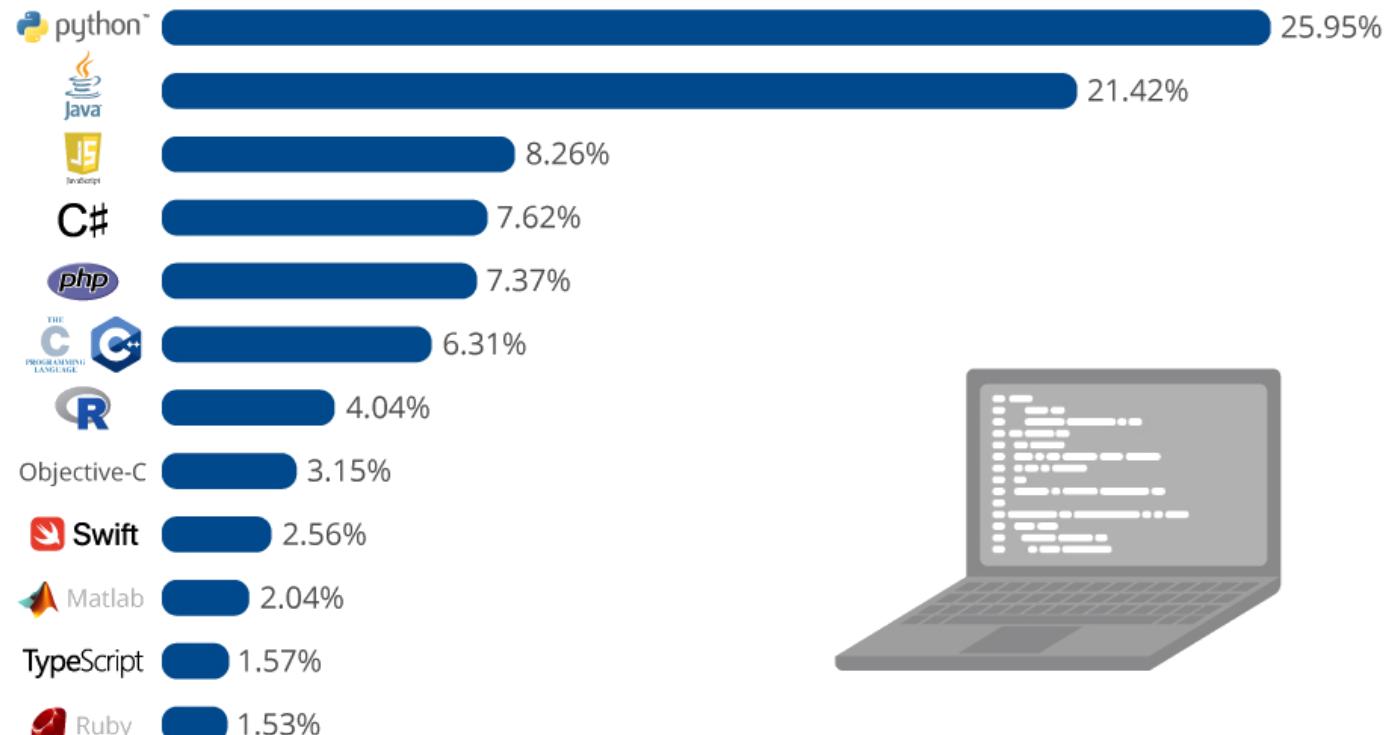


# Python

- > Interprété
- > Multiplateformes
- > Multiparadigme
- > Ouvert
- > Orienté objet
- > Fortement typé
- > Incroyablement populaire

## The Most Popular Programming Languages

Share of the most popular programming languages in the world\*



\* Based on the PYPL-Index, an analysis of Google search trends for programming language tutorials.



Source: PYPL

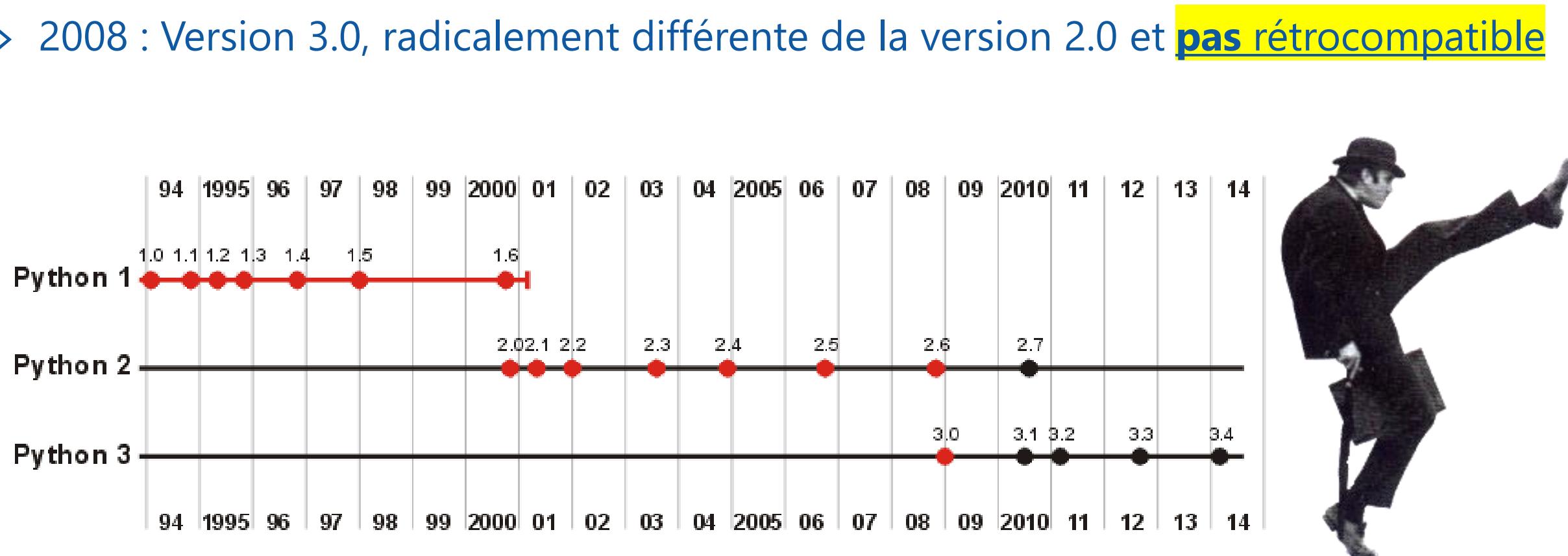
statista

<https://www.statista.com/chart/16567/popular-programming-languages/>



# Historique

- > 1991 : Version 1.0, créée par le néerlandais Guido van Rossum, fan de *Monty Python*.
- > 2000 : Version 2.0, contenant plusieurs améliorations.
- > 2008 : Version 3.0, radicalement différente de la version 2.0 et pas rétrocompatible





# On peut faire quoi?

Python est devenu énormément populaire grâce à sa simplicité et sa versatilité.

Quelques usages typiques:

- > **Administration systèmes et réseaux** (UNIX/LINUX, Cisco, Ansible, etc.)
- > **Développement Web** (Django, Flask, Pyramid, web2py, etc.)
- > **Scripts** et macros applicatifs (Blender, Inkscape, LibreOffice/OpenOffice, etc.)
  - > Les effets spéciaux de *Star Wars: Episode I – The Phantom Menace* ont été réalisés à l'aide de Python! ([Lien](#))
- > Analyse de données, calcul scientifique, Big Data, *machine-learning*, etc.



# L'avantage de Python

Nous utiliserons Python pour créer des scripts avancés :

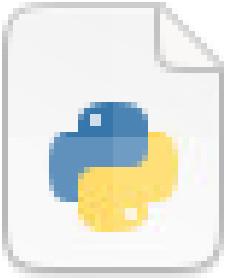
- > Un script est centré sur une tâche précise
- > Interprétés (ne nécessitent pas de compilation)
- > Utilisant la librairie standard
- > Utilisant des librairies spécialisées pour exécuter des tâches complexes
- > Nécessitant l'utilisation de structures de contrôle

Python nous permet d'écrire des scripts utilisant les principes de programmation fonctionnelle ou orientée objet selon le besoin.

Les librairies puissantes disponibles avec Python nous permettent d'écrire rapidement des scripts capables de résoudre des problèmes complexes.



# Fichiers scripts (.py)



```
1 #! /bin/python3
2 # suite de Fibonaci jusqu'a n
3 def fib(n):
4     a, b = 0, 1
5     while a < n:
6         print(a, end=' ')
7         a, b = b, a+b
8     print()
9 fib(1000)
```

```
pierre-paul@pp-vm:~$ python3 fibonacci.py
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
pierre-paul@pp-vm:~$
```



# Origine de python : l'horreur

- > En C#, tous ces scripts sont équivalents :
- > Python est né en partie d'un désir d'avoir un langage qui force le programmeur à bien structurer son code

A)

```
1 string[] fruits = {"pomme", "orange", "patate", "kiwi"};
2 foreach (string i in fruits) {
3     Console.WriteLine(i);
4 }
```

B)

```
1 string[] fruits = {"pomme", "orange", "patate", "kiwi"};
2 foreach(string i in fruits) { Console.WriteLine(i); }
```

C)

```
1 string[] fruits = {"pomme", "orange", "patate", "kiwi"};
2 foreach(string i...
3     in...
4     fruits)...
5     {
6         Console.WriteLine(i);
7     }
...
```

D)

```
1 string[] fruits = {"pomme", "orange", "patate", "kiwi"};
2             foreach (string i...
3             in...
4             fruits)...
5                 Console.WriteLine(i);
6             }
...
```



# Indentation

- > Fait partie de la **syntaxe**.
- > Identifie les niveaux de **profondeur**.
- > Caractères **d'espacement** ou de **tabulation**.
- > Doit être **constante** dans tout le script.

```
1 if n > 100:  
2     print('Valeur excédent le max')  
3  
4 if n < 10:  
5     print('Valeur trop petite')  
6     if n < 0:  
7         print('Valeur négative !')
```

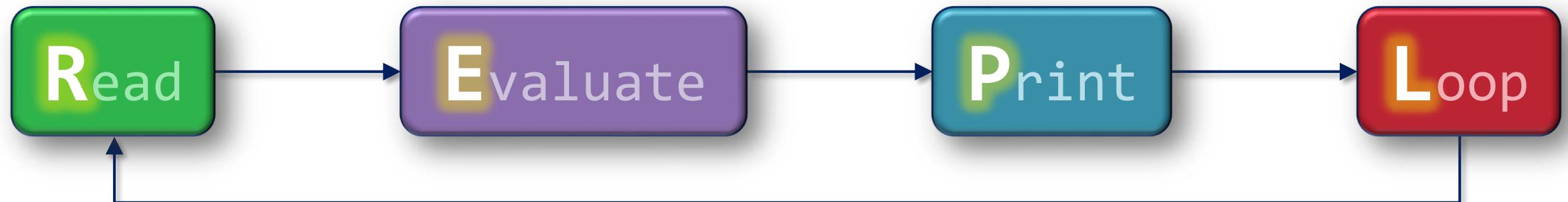




# Outils pour commencer



# Interface interactive (REPL)



```
C:\Users\pierre-paul.gallant>python
Python 3.10.5 (tags/v3.10.5:f377153, Jun  6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> print ("Hello world ! En python !")
Hello world ! En python !
>>>
>>> exit()
C:\Users\pierre-paul.gallant>_
```

The screenshot shows a terminal window titled 'Sélection Invite de commandes'. It displays a Python REPL session. The user types 'python' to start the interpreter, which prints its version and build information. Then, the user enters a multi-line command starting with '>>> print ("Hello world ! En python !")', followed by a carriage return. The interpreter executes the command and prints the output 'Hello world ! En python !'. Finally, the user types '>>> exit()' and presses enter to quit the interpreter.



# help()

- > Dans l'interpréteur, la commande **help()** vous amène dans l'utilitaire d'aide.
- > Il suffit d'écrire une fonction ou un sujet pour obtenir de la documentation sur le sujet.
- > Écrire **str** nous sort une description de la classe string et des méthodes utilisables.

```
Invite de commandes - python
Help on class str in module builtins:

class str(object)
|   str(object='') -> str
|   str(bytes_or_buffer[, encoding[, errors]]) -> str
|
|   Create a new string object from the given object. If encoding or
|   errors is specified, then the object must expose a data buffer
|   that will be decoded using the given encoding and error handler.
|   Otherwise, returns the result of object.__str__() (if defined)
|   or repr(object).
|   encoding defaults to sys.getdefaultencoding().
|   errors defaults to 'strict'.
|
| Methods defined here:
|
| __add__(self, value, /)
|     Return self+value.
|
| __contains__(self, key, /)
|     Return key in self.
|
| __eq__(self, value, /)
|     Return self==value.
|
| __format__(self, format_spec, /)
|     Return a formatted version of the string as described by format_spec.
|
-- Suite --
```



# help()

- > **dir(str)** nous donne toutes les méthodes de la classe str.
- > **help(str.upper)** nous donnent une description de la méthode « upper » de la classe str.

```
Invite de commandes - python

C:\Users\pierre-paul.gallant>python
Python 3.10.5 (tags/v3.10.5:f377153, Jun  6 2022, 16:14:13) [MSC v.1929 64 bit
n win32
Type "help", "copyright", "credits" or "license" for more information.
>>> dir(str)
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__',
 '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__',
 '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__',
 '__lod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
 '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',
 '__casfold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find',
 'format_map', 'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit',
 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper',
 'lstrip', 'lower', 'lstrip', 'maketrans', 'partition', 'removeprefix', 'removesuffix',
 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split',
 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
>>>
>>> help(str.upper)
Help on method_descriptor:

upper(self, /)
    Return a copy of the string converted to uppercase.

>>>
```

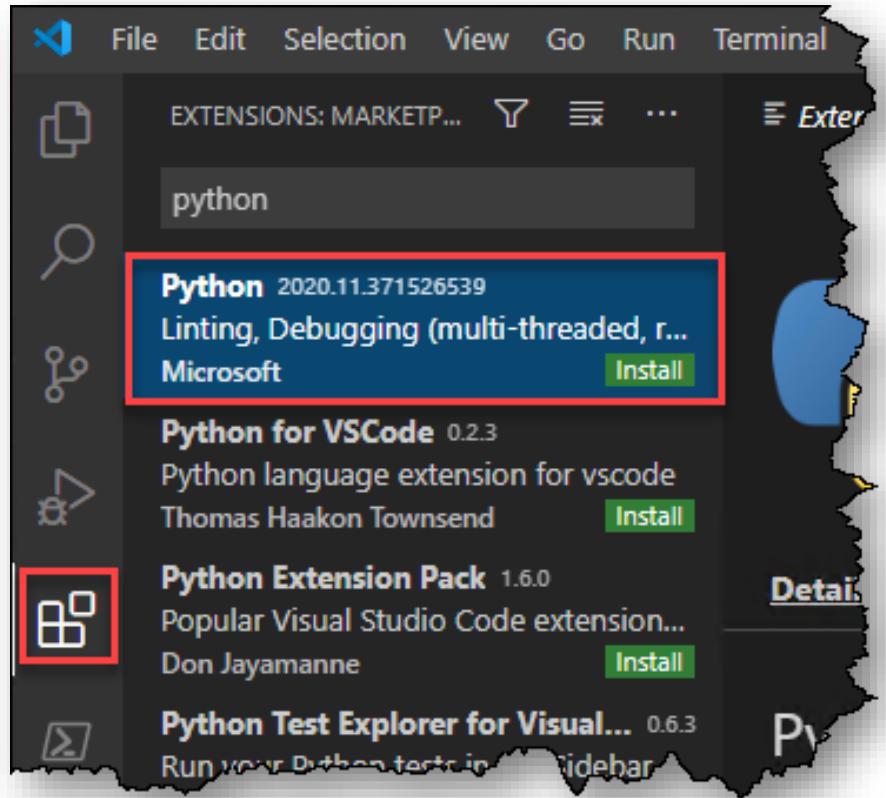


# Ressources en ligne

- > **docs.python.org/3** : Contient les mêmes informations que help() mais avec plus de détails et une interface plus conviviale.
- > **Google.com** : répond à vos questions (essentiel en informatique de nos jours)
- > **w3schools.com/python/** : explications et exemples



# Visual Studio Code (VSC)



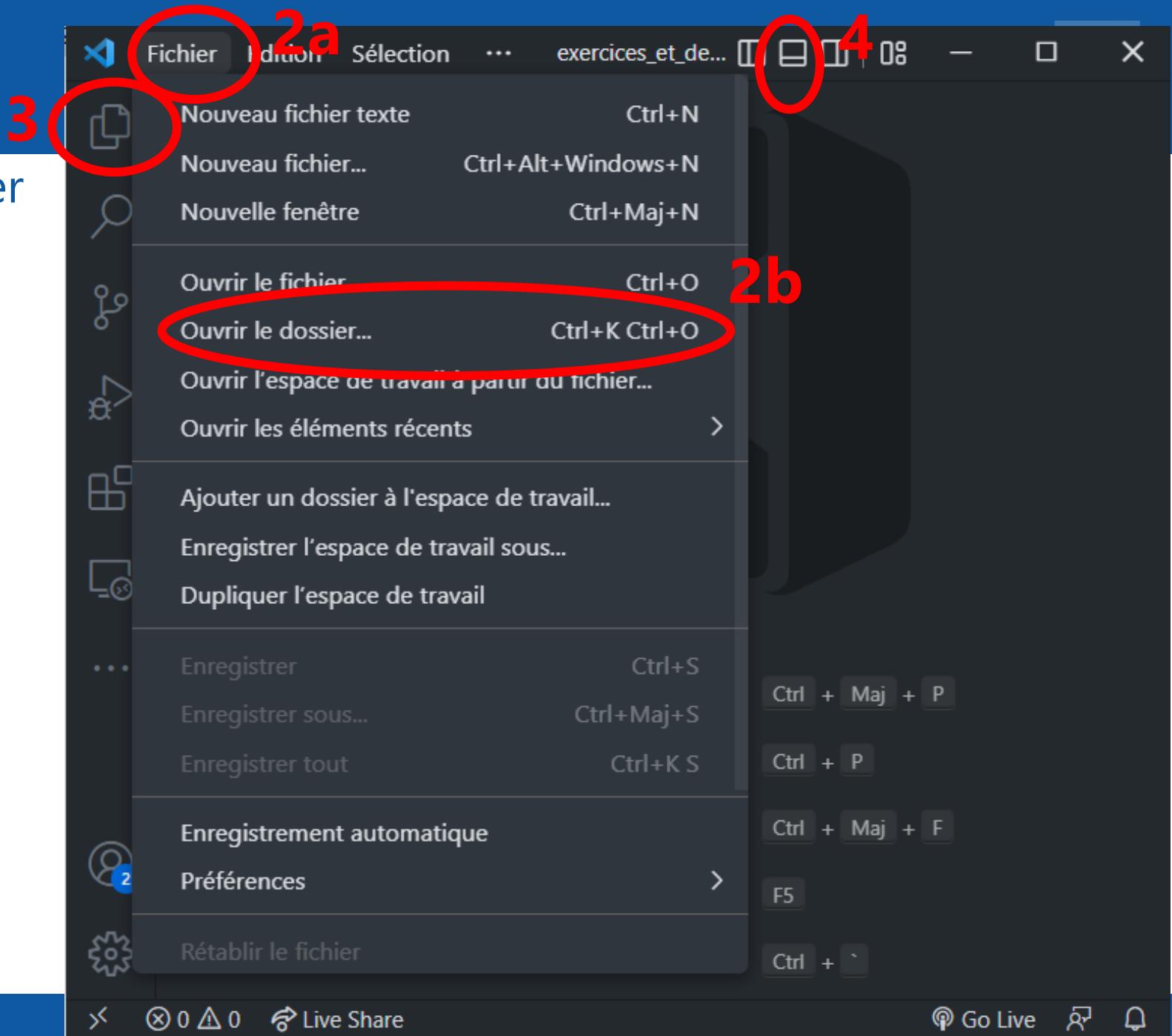
IDE simple et léger

Utilisable sur tous les OS

L'extension Python permet de coder simplement et rapidement.

# Utiliser VSC

- 1) Créer ou télécharger le dossier d'exercices
- 2) Dans VSC, aller dans Fichier > ouvrir le dossier... et sélectionner le dossier d'exercices
- 3) Sélectionner le menu explorateur
- 4) Activer le panneau terminal



# Utiliser VSC

Exécution du code  
/ du débuggeur



Explorateur de fichiers

The screenshot shows the Visual Studio Code interface with the following elements:

- Explorateur de fichiers (File Explorer):** Located on the left, it shows a tree view of files and folders. A red bracket on the left groups this pane with the terminal.
- Editor:** The main central area displays a Python script named `demo1.py`. The code contains variables `nombre_un` and `nombre_deux`, an if/elif/else block for comparison, and print statements. A red circle highlights the play button icon (`>`) in the top right corner of the editor tab, which is used for running the script.
- Terminal:** At the bottom, the terminal window shows the command `python .\R01\Demo\exemple.py` being run, followed by the output "petit". A red bracket on the right groups this pane with the file explorer.

Le script en cours

Terminal



# Les bases :

- > Types de données
- > Opérateurs
- > Structures de données
- > Structures de contrôles
- > Entrées/sorties de données



# Les noms de variables dans Python

- > Sensibles à la case (majuscule et minuscule)
- > Doivent commencer par une lettre ou un « underscore » ( \_ )
- > Peuvent contenir lettres, chiffres, et \_, **PAS** d'accents
- > Le standard : noms en minuscules, mots séparés par \_

~~scoreDuJoueur~~

score\_du\_joueur 

- > MAIS ! Les underscores ( \_ ) au début et à la fin d'une variable ont une signification particulière que l'on verra plus tard.



# Types de données

Variables et leurs utilisations



# Principaux types de données

Type	Description	Example
int	Nombre entier	varint = 42
float	Nombre réel (point flottant)	varfloat = 3.14
bool	Valeur booléenne (True et False, avec la majuscule)	varbool = True
str	Chaîne de caractères	varstr = "Spam"
liste	Plusieurs objets stockés séquentiellement dans une variable	varliste = ["tomate", "celeri" , "concombre" ]
dictionnaire	Plusieurs objets stockés avec une clef correspondant dans une variable	vacDic = {"legume" : "concombre", "fruit" : "pomme" }

Vérifier le type d'une variable avec la fonction **type()** :

```
>>> type(varstr)
<class 'str'>
>>>
```





# Typage dynamique

Les variables dans Python ont toutes un type, mais ce type n'est vérifié que lorsque le code est interprété. Il peut changer dans le temps.

```
pierre-paul@pp-vm:~/scripts$ python3
Python 3.10.6 (main, Nov 14 2022, 16:10:14) [GCC 11.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 9
>>> type(a)
<class 'int'>
>>>
>>> a = a / 2
>>> print (a)
4.5
>>> type(a)
<class 'float'>
>>> a = "oranges"
>>> type(a)
<class 'str'>
>>> █
```

Le type des variables n'est donc pas déclaré lors de leur initialisation.

On utilise les variables de la façon qui nous convient le mieux.





# Sous-strings

- > Facile d'accéder à différentes portions des « strings » ou de créer des « sous-strings »
- > En spécifiant la sous-chaine recherchée
- > Ou même en commençant de la fin avec des index négatifs
- > On appelle ces types de manipulation : « slicing »

```
>>> liste1 = "Bonjour le monde"
>>> print(liste1)
Bonjour le monde
```

```
Bonjo
>>> print(liste1[0:5])
Bonjo
>>> print(liste1[:5])
Bonjo
```

```
Bonjo
>>> print(liste1[2:])
njour le monde
```

```
e
>>> print(liste1[-1])
e
>>> print(liste1[-5:])
monde
```

# f-strings

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

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

La valeur des variables peut être utilisée 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 ')





- > Listes de caractères formatés
- > Permet de changer dynamiquement les chaînes de caractères.
- > Facilite la lecture et la maintenance du code.

```
↳ formated_strings.py > ...
1  prenom = "Pierre-Paul"
2  nom_famille = "Gallant"
3
4  print("Bonjour,", prenom, nom_famille, "!")
5  print("Bonjour, {} {}".format(prenom, nom_famille))
6  print(f"Bonjour, {prenom} {nom_famille} !")
7
8  # Mais si on décide de changer le "!" pour un "."
9  print("Bonjour, ", prenom, " ", nom_famille, ".", sep="")
10 print("Bonjour, {} {}".format(prenom, nom_famille))
11 print(f"Bonjour, {prenom} {nom_famille}.")
```

```
1  prenom = "pierre-paul"
2
3  print(f"Bonjour {prenom.capitalize()}.")
```



# Operateurs



# Opérateurs arithmétiques

Operateurs	Nom	Utilisation	Exemple (x=5 ; y =2)
+	Addition	$x + y$	$x + y == 7$
-	Soustraction	$x - y$	$x - y == 3$
*	Multiplication	$x * y$	$x * y == 10$
/	Division	$x / y$	$x / y == 2.5$
%	Modulus	$x \% y$	$x \% y == 1$
**	Exponentiation	$x ** y$	$x ** y == 25$
//	Division plancher	$x // y$	$x//y == 2$



# Opérateurs d'assignation

Operateur	Exemple	Équivalent
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x // 3	x = x // 3
**=	x **= 3	x = x ** 3

Notez : Les opérateurs **++** et **--** n'existent **pas** dans python.

Il faut plutôt utiliser **x += 1** et **x -= 1**



# Operateurs logiques

- > Dans C# on utilisait &&, ||, et !
- > Dans python, on utilisera les « keywords » :

« and », « or », et « not »

a=1

b=2

c=3

```
print (a==1 and b== 2)
```

```
print (a==1 or b==4)
```

```
print (not c == 2)
```



# Structures de contrôle



# Structures de contrôle

- > Les structures de contrôles telles que les conditionnelles et les boucles sont commencées par le deux-points : « : »
- > Leur contenu est ensuite déterminé par l'indentation.
- > La tabulation va déterminer ce qui fait partie d'une structure de contrôle et ce qui n'en fait pas partie.

```
1 if (n > 100):  
2     print('Valeur excédent le max')  
3  
4 if (n < 10):  
5     print('Valeur trop petite')  
6  
7 (( reste du code ))  
8
```

The diagram illustrates the structure of the provided Python code. Vertical dashed lines define the scope of each conditional block. A red arrow points to the first 'if' statement at line 1, indicating its start. A double-headed horizontal arrow at the bottom indicates the continuation of the code block, spanning from the end of the first 'if' block to the end of the code at line 8.



# Structures de contrôle (exemple)

- > Deux structures de contrôles.
- > Une à l'intérieur de l'autre.
- > La tabulation indique la profondeur et quand une est section finie.
- > Le nombre d'espaces doit être uniforme à travers le script.

```
with open('contributorsNomPrenom.csv', 'r', encoding='utf-8') as csv_file
    csv_reader = csv.reader(csv_file)
    # skip the first line #
    next(csv_reader)
    for line in csv_reader:
        print(line)
```



# if, elif, else



- > Même principe quand C#
- > Le « if » est suivi d'un test logique dont la valeur est évaluée à True ou False lors de l'exécution.
- > Le « else if » de C# devient elif en Python

```
1 # exemple if, elif, else
2 x = float(input("\nInserez la valeur de x : "))
3
4 if x < 0 :
5     print("La valeur de x est négative")
6 elif x > 0 :
7     print("La valeur de x est positif")
8 else:
9     print("La valeur est 0")
10 print("")
```

PROBLÈMES 3 SORTIE CONSOLE DE DÉBOGAGE TERMINAL .NET INTERA

PS C:\Users\pierre-paul.gallant\Documents\2n6-Python\exercices\_exercices\_et\_demos\R01/Demo/demo6.py

Inserez la valeur de x : 5  
La valeur de x est positif





# Structures de contrôle ( les boucles)

- > La boucle « while » fonctionne de la même manière que dans les autres langages
- > La boucle « for » utilisée avec la fonction « range() » fonctionne de façon similaire à la boucle « for » en C#
- > Dans Python, la boucle « for » est principalement utilisée pour itérer sur chaque élément d'une liste ou groupe de données.

```
1 while a < n:  
2     print(a, end=' ')  
3     a = a + 2
```

```
PS C:\Users\pierre-paul.gall  
0 2 4 6 8
```

```
1 mot = 'Montpetit'  
2 for i in range(4,len(mot)):  
3     print(mot[i])
```

```
PS C:\Users\pierre-paul.gal  
p  
e  
t  
i  
t
```

```
1 mot = 'Montpetit'  
2 for lettre in mot :  
3     print(lettre)
```

```
pierre-paul@pp-vm:~/scripts$ ./loop.py  
M  
o  
n  
t  
p  
e  
t  
i  
t
```





# Structures de contrôle (boucles)

- > Deux mots-clés sont importants pour l'utilisation des boucles : « break » et « continue »
- > « break » va interrompre une boucle et en sortir sans exécuter le reste du code dans l'itération

```
1 mot = 'Montpetit'  
2 for lettre in mot :  
3     if lettre == 'e' :  
4         break  
5     print(lettre, end=' ')  
6 print('fin !')  
7
```

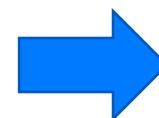
```
pierre-paul@pp-vm:~/scripts$ ./loop.py  
M  
o  
n  
t  
p  
fin !
```



# Structures de contrôle (boucles)

- > Deux mots-cléfs sont importants pour l'utilisation des boucles : « break » et « continue »
- > « continue » va interrompre une boucle et passé à la prochaine itération sans passer par le code restant dans la présente itération.

```
1 mot = 'Montpetit'  
2 for lettre in mot :  
3     if lettre == 'e' :  
4         continue  
5     print(lettre, end=' ')  
6 print('fin !')  
7
```



```
pierre-paul@pp-vm:~/scripts$ ./loop.py  
M  
o  
n  
t  
p  
t  
i  
t  
fin !
```



# Entrée et Sortie de données



# Input

- > Fonction de base pour passer une chaîne de caractère à un programme lors de l'exécution.

```
nom = input("Écrire votre nom : ")
```

Variable à laquelle on attribue la valeur renvoyée.

Message imprimé lors de l'exécution.

```
nom = input("Écrire votre nom : ")  
print(f"Bonjour {nom}.")
```

```
C:\Users\pierre-paul.gallant\Documents\2n6-Python\  
exercices_et_demos\R01\Demo>python exemple.py  
Écrire votre nom : Pierre-paul  
Bonjour Pierre-paul.
```

- > Input() retourne un **str**, si on veut un **int** ou **float** il faut faire la conversion.

```
age = int(input("Écrire votre age :"))
```





# PRINT

- > Fonction de base pour afficher des chaînes de caractères. (ou d'autres objets)

Help on built-in function print in module builtins:

```
print(...)  
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)  
  
    Prints the values to a stream, or to sys.stdout by default.  
    Optional keyword arguments:  
    file:  a file-like object (stream); defaults to the current sys.stdout.  
    sep:   string inserted between values, default a space.  
    end:   string appended after the last value, default a newline.  
    flush: whether to forcibly flush the stream.  
(END)
```

- > `sep` : string séparant les valeurs
- > `end` : string inséré après la dernière valeur
- > `file` : où on sort le résultat (le `sys.stdout` par défauts ; i.e. le terminal)

```
>>> print ("Bonjour", "le", "Monde", "!", sep=" ")  
Bonjour le Monde !  
>>> print ("Bonjour", "le", "Monde", "!", sep="_")  
Bonjour_le_Monde_!  
>>> print ("Bonjour", "le", "Monde", "!", sep="\n")  
Bonjour  
le  
Monde  
!  
>>> █
```



# Annexe



# Installer Python selon l'OS

- > Windows:
  - > Package: <https://www.python.org/downloads/windows/>
  - > Microsoft Store: <https://www.microsoft.com/fr-ca/search?q=python>
  - > Nuget: <https://www.nuget.org/packages/python/>
  - > Chocolatey: <https://chocolatey.org/packages?q=python>
- > Linux:
  - > Apt (Debian, Ubuntu): `sudo apt install python3`
  - > Yum (RedHat, CentOS): `sudo yum install python3`
  - > Autres: <https://www.python.org/downloads/source/>
- > Mac OS X
  - > Package: <https://www.python.org/downloads/mac-osx/>
  - > Homebrew: <https://link.medium.com/7ZputWaFDbb>



# Pour en savoir plus...

- > [Python Command Line Arguments – Real Python](#)
- > [Common Python Data Structures \(Guide\) – Real Python](#)

# 2N6 Programmation 2



python™



Rencontre 2:

- Listes dans python (l'équivalent des *arrays*)
- Débogage avec VSC



# Les listes

Les possibilités qu'elles offrent



# Listes

Les listes sont des structures de données qui permettent de stocker plusieurs objets dans une même variable.

Les listes sont des structures ordonnées. Les objets stockés dans la liste ont un ordre et sont accessibles par leur indice / position qu'on nomme souvent index.

```
liste = ['Ford Mustang 1964', 'Reliant Robin 1988', 'Toyota Tercel 1991']

print(liste)
# ['Ford Mustang 1964', 'Reliant Robin 1988', 'Toyota Tercel 1991']
print(liste[1])
# 'Reliant Robin 1988'
print(liste[2])
# 'Toyota Tercel 1991'
```

# Accéder aux méthodes

Comme pour toute autre classe :

« dir() » nous donnent la liste des méthodes disponibles

Méthodes spéciales (dunder)

Méthodes des objets « list »

On lance l'interpréteur Python

```
C:\Users\pierre-paul.gallant>python
Python 3.10.5 (tags/v3.10.5:f377153, Jun  6 2022, 16:14:1
3) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more
information.

>>>
>>> dir(list)
['__add__', '__class__', '__class_getitem__', '__contains__',
 '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__',
 '__format__', '__ge__', '__getattribute__', '__getitem__',
 '__gt__', '__hash__', '__iadd__', '__imul__', '__init__',
 '__init_subclass__', '__iter__', '__le__', '__len__',
 '__lt__', '__mul__', '__ne__', '__new__', '__reduce__',
 '__reduce_ex__', '__repr__', '__reversed__', '__rmul__',
 '__setattr__', '__setitem__', '__sizeof__', '__str__',
 '__subclasshook__', 'append', 'clear', 'copy', 'co
unt', 'extend', 'index', 'insert', 'pop', 'remove', 'reve
rse', 'sort']
>>>
```

# Listes (méthodes)



> `append()`: ajoute un objet à la fin de la liste

```
>>> cours = ["prog 1", "reseau 1"]
>>> cours.append("prog 2")
>>> print(cours)
['prog 1', 'reseau 1', 'prog 2']
>>>
```

> `clear()`: Vide une liste

```
>>> cours.clear()
>>> print(cours)
[]
```

> `count()`: Compte le nombre de fois qu'une valeur apparaît dans la liste

```
>>> cours.count("prog 1")
0
>>> cours = ["prog 1", "res 1", "prog 2", "prog 1"]
>>> cours.count("prog 1")
2
```



# Listes (méthodes)



- > **extend():** Ajoute les valeurs d'une liste à la liste existante (fonctionne différemment de append() )
- > **index():** Retourne l'index (i.e; la position) de la valeur passée
- > **insert() :** Ajoute un objet à la position indiquée

```
>>> cours = ["prog 1", "res 1", "prog 2", "prog 1"]
>>> cours2 = ["philosophie", "anglais", "français"]
>>> cours.extend(cours2)
>>> print(cours)
['prog 1', 'res 1', 'prog 2', 'prog 1', 'philosophie', 'anglais', 'français']
>>> -
```

```
>>> print(cours.index("res 1"))
1
>>>
```

```
>>> print(cours)
['prog 1', 'res 1', 'prog 2', 'prog 1', 'philosophie', 'anglais', 'français']

>>> cours.insert(2, "prog 3")
>>> print(cours)
['prog 1', 'res 1', 'prog 3', 'prog 2', 'prog 1', 'philosophie', 'anglais', 'français']
>>> -
```



# Listes (méthodes)

- > `remove()` : retire le premier objet correspondant à la valeur passée
- > `pop()` : retire ET retourne l'objet à l'index indiqué. Par défaut : le dernier objet
- > `reverse()` : inverse l'ordre des objets dans la liste
- > `sort()` : trie la liste en ordre croissant

```
>>> cours.remove("prog 1")
>>> print(cours)
['res 1', 'prog 3', 'prog 2', 'prog 1', 'philo', 'anglais', 'francais']
>>>
```

```
>>> cours_retirer = cours.pop(1)
>>> print(cours)
['res 1', 'prog 2', 'prog 1', 'philo', 'anglais', 'francais']
>>> print(cours_retirer)
prog 3
>>> -
```

```
>>> cours.reverse()
>>> print(cours)
['francais', 'anglais', 'philo', 'prog 1', 'prog 2', 'res 1']
>>>
```

```
>>> cours.sort()
>>> print(cours)
['anglais', 'francais', 'philo', 'prog 1', 'prog 2', 'res 1']
>>> -
```

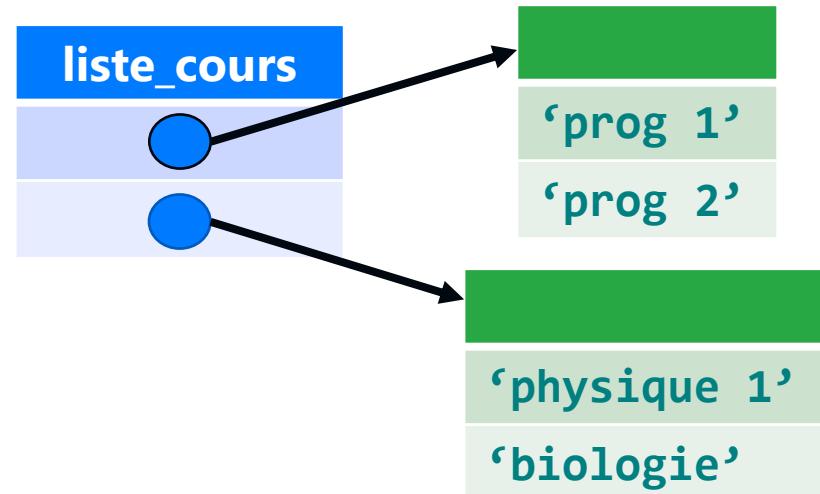
# Listes (méthode copy)

> `copy()` : retourne une copie « peu profonde » de la liste.

Génère une nouvelle liste contenant les mêmes références aux mêmes objets que la première liste.

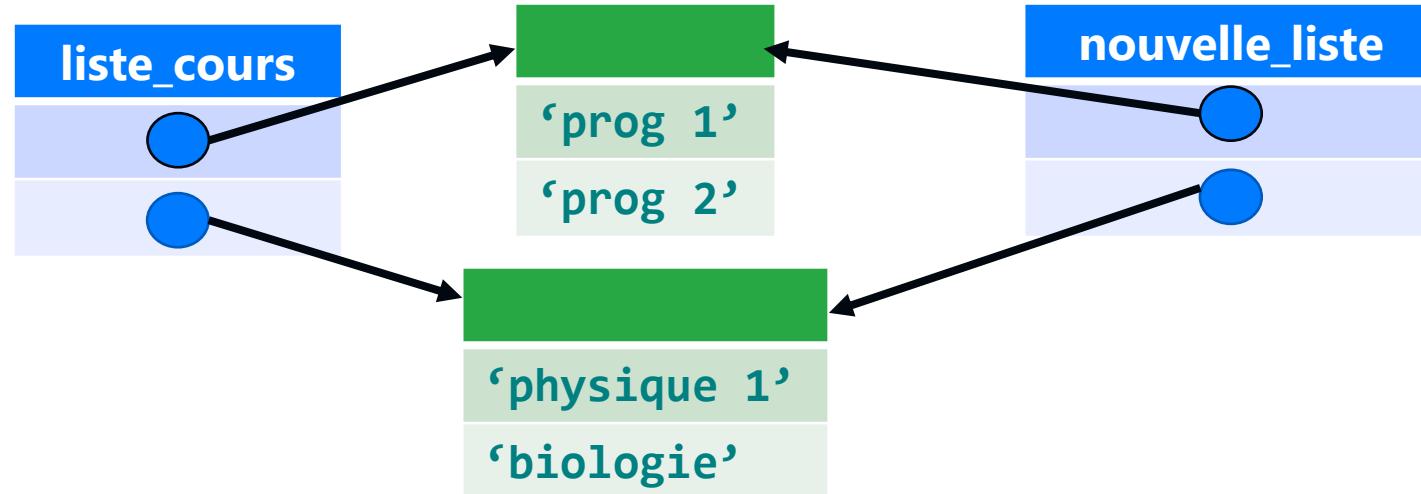
```
liste_cours = [[‘prog 1’, ‘prog 2’], [‘physique 1’, ‘biologie’]]
```

L'objet « `liste_cours` »  
contient des pointeurs vers  
deux autres objets listes.



# Listes (méthode copy)

```
nouvelle_liste = liste_cours.copy()  
  
print(nouvelle_liste)  
[['prog 1', 'prog 2'], ['physique 1', 'biologie']]
```



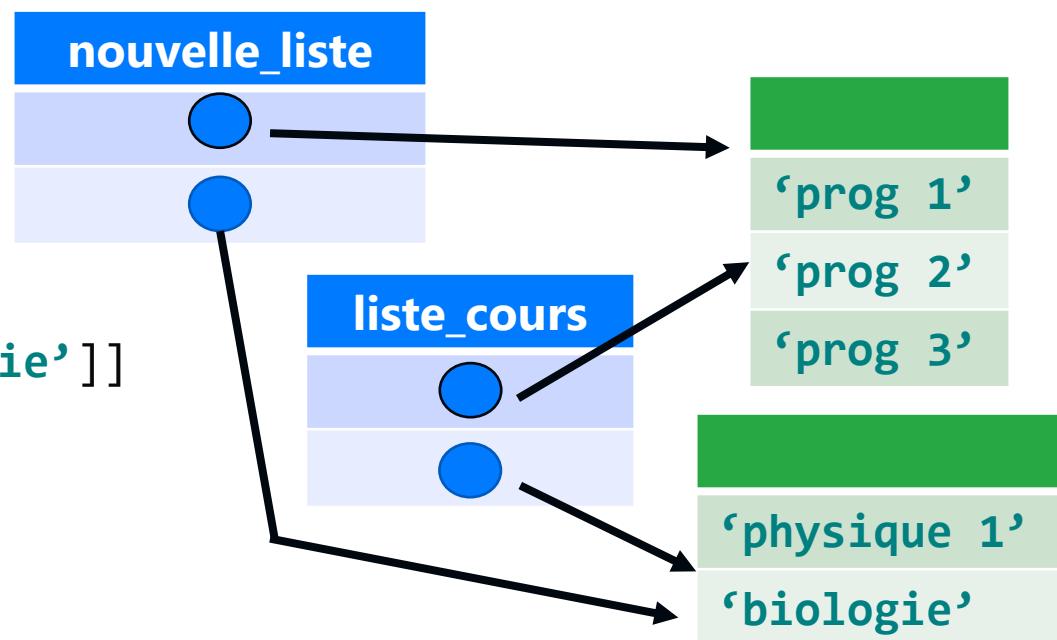


# Listes (méthode copy)

- Les listes « liste\_cours » et « nouvelle\_liste » font référence aux mêmes objets. Donc des modifications effectuées sur un objet dans une des listes seront visibles à partir de l'autre liste.

```
liste_cours[0].append('prog 3')
```

```
print(nouvelle_liste)  
[['prog 1', 'prog 2', 'prog 3'], ['physique 1', 'biologie']]
```





# Sous-Listes

- > Les listes peuvent être facilement divisées en sous-listes avec des opérations de « slicing »

```
cours = ['prog 1', 'prog 2', 'physique 1', 'biologie', 'anglais', 'philosophie']
```

- > cours[x] retourne la valeur à l'indice donné

```
>>> print(cours[3])
biologie
```

- > cours[x:y] retourne une sous liste contenant les valeurs se trouvant de l'index [x] jusqu'à l'index [y-1]

```
>>> print(cours[0:2])
['prog 1', 'prog 2']
```

- > cours[-1] retourne la valeur à la dernière position.

```
>>> print(cours[-1])
philosophie
```





# Sous-Listes

```
cours = ['prog 1', 'prog 2', 'physique 1', 'biologie', 'anglais', 'philosophie']
```

- > cours[:x] retourne une sous-liste contenant les valeurs du début jusqu'à l'index [x] (non inclus)
- > cours[x:] retourne une sous-liste contenant les valeurs de l'index [x] jusqu'à la fin
- > cours[-x:-y] retourne une sous-liste avec les valeurs de -x jusqu'à -y (non inclus)

```
>>> print(cours[:2])  
['prog 1', 'prog 2']
```

```
>>> print(cours[4:])  
['anglais', 'philosophie']
```

```
>>> print(cours[-5:-2])  
['prog 2', 'physique 1', 'biologie']
```



# Fonctions utiles avec les listes

- > Les fonctions « min() », « max() », et « sum() » sont souvent utilisées avec les listes

```
notes_gr10 = [ 34, 87, 96, 67.4, 72, 99.2, 59 ]
```

- > min() retourne la valeur la plus basse

```
>>> print(min(notes_gr10))  
5
```

- > max() retourne la valeur la plus élevée

```
>>> print(max(notes_gr10))  
99.2
```

- > sum() retourne la sommation des valeurs

```
>>> print(sum(notes_gr10))  
460.5999999999997
```

```
>>> moyenne = sum(notes_gr10) / len(notes_gr10)  
>>> print(moyenne)  
65.8
```

# Boucles for et listes



# retour

- > Itère sur la liste, la variable `cours` prend la valeur de chacun des objets dans la liste un après l'autre.
  - > Itère sur la liste, la variable `index` prend les valeurs numériques de 0 jusqu'à la valeur de la longueur de la liste.

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

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





# Débogage avec un IDE

Principe et outils dans Visual Studio Code



# Messages d'erreur

- > Important pour la maintenance de code complexe.
- > Contiens plusieurs outils de débogage
- > Problème avec le code :
  1. Regarder le message d'erreur dans la console

```
pierre-paul@pp-vm:~/scripts$ /bin/python3 /home/pierre-paul/scripts/zzz_factoriel_broken.py
Traceback (most recent call last):
  File "/home/pierre-paul/scripts/zzz_factoriel_broken.py", line 10, in <module>
    print (factoriel(a))
NameError: name 'a' is not defined
```

L'erreur qui a eu lieu. Ici la variable « a » n'est pas définie.

La ligne qui a causé une erreur

Le fichier qui a causé une erreur



# débogueur

## > Problème avec le code :

2. Ajouter un ou des breakpoint(s) avec un clic dans l'espace à gauche des lignes de code

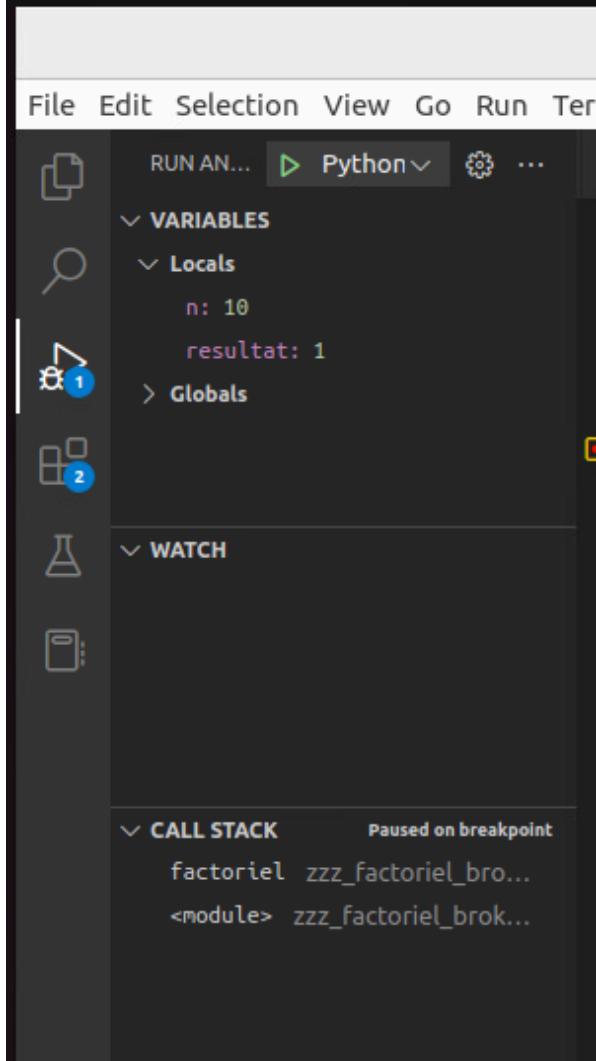
Quand on lance le script en mode Debug : le script va passer à travers les lignes de code et s'arrêter au premier breakpoint rencontré.

```
zzz_factoriel_broken.py M
zzz_factoriel_broken.py > ⌂ factoriel
1  #! /bin/python3
2
3  def factoriel (n):
4      resultat = 1
5      while n > 0 :
6          resultat = resultat * n
7          n = n-1
8
9  print("Le factoriel est : ")
10 print (factoriel(10))
11 print("fin du factoriel")
```

The screenshot shows a code editor window with a dark theme. On the left, there's a file tree with 'zzz\_factoriel\_broken.py' selected. In the main editor area, there's a Python script with several lines of code. A red arrow points from the left margin of the code area to the top toolbar. Another red arrow points from the top toolbar to a context menu that is open. The context menu has two items: 'Run Python File' and 'Debug Python File'. The 'Debug Python File' item is highlighted with a red box.



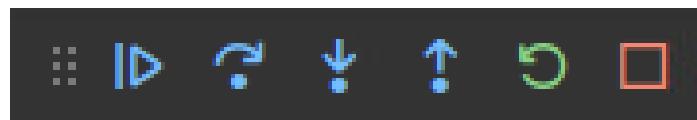
# débogueur



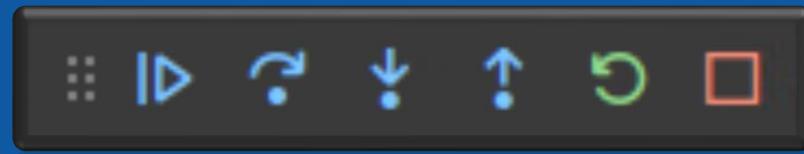
> Problème avec le code :

3. Une fois arrêté, on peut exécuter le code ligne par ligne avec le panneau de contrôle qui apparait dans le mode débogage.

On peut voir les variables et leurs valeurs dans le panneau de débogage à gauche à mesure que le code est exécuté.



# Naviguer le débogueur



➤ Continue : Le code s'exécute jusqu'au prochain breakpoint



➤ Step over : Exécute la prochaine ligne de code SANS entrer dans les fonctions



➤ Step into : Exécute la prochaine ligne de code INCLUANT entrer dans les fonctions s'il s'agit de la prochaine ligne exécutée



➤ Step out : Recule d'une ligne dans le code NE MODIFIE PAS LES VARIABLES



➤ Restart : Relance le script (encore en mode débogage)



➤ Stop : Quitte le mode débogage

# Exécution de code spécifique

- > Lorsqu'on développe un script, on peut décider d'exécuter uniquement certaines parties pour des fins de débogage.
- > Dans le cas où une partie du code prend longtemps à exécuter ou n'est pas nécessaire

```
R02_Demo > /* R2_demo4 copy.py
1 #execution de lignes spécifiques
2
3 while True:
4     print("Boucle infinie")
5
6 cartes_graphiques_en_stock = ['GeForce RTX 3070Ti',
7                                 'Radeon RX 6950 XT',
8                                 'Radeon RX 6900 XT']
9 for carte in cartes_graphiques_en_stock:
10    print(f"La {carte} est disponible en magasin.")

PROBLÈMES 1 SORTIE CONSOLE DE DÉBOGAGE TERMINAL .NET INTERACTI
Copyright (C) Microsoft Corporation. Tous droits réservés.

Testez le nouveau système multiplateforme PowerShell https://aka.ms/pscore-dotnet-get-started

PS C:\Users\pierre-paul.gallant\Cégep Édouard-Montpetit\CMT-420_I> & "C:/Program Files/Python310/python.exe"
Python 3.10.5 (tags/v3.10.5:f377153, Jun  6 2022, 16:14:13) [MSC v.1932 64 bit (AMD64)]
Type "help", "copyright", "credits" or "license" for more information
>>> cartes_graphiques_en_stock = ['GeForce RTX 3070Ti',
...                                'Radeon RX 6950 XT',
...                                'Radeon RX 6900 XT']
...
>>> for carte in cartes_graphiques_en_stock:
...     print(f"La {carte} est disponible en magasin.")
...
La GeForce RTX 3070Ti est disponible en magasin.
La Radeon RX 6950 XT est disponible en magasin.
La Radeon RX 6900 XT est disponible en magasin.
>>>
```



# Exécution de code spécifique

1. Sélectionner les lignes à exécuter.
2. Presser « F1 » pour voir toutes les commandes.
3. Trouver l'option « Run Selection/Line in python »

The screenshot shows a code editor with two tabs: R2\_demo2.py and R2\_demo12.py. The R2\_demo12.py tab is active, displaying the following Python code:

```
/* R2_demo2.py */ /* R2_demo12.py */
R02_Demo > /* R2_demo4 copy.py
1 #execution de lignes
2
3 while True:
4     print("Boucle infinie")
5
6 cartes_graphiques_en_stock = []
7
8
9 for carte in cartes_graphiques_en_stock:
10    print(f"La {carte} est disponible en magasin.")
11
12
13 while True:
14     print("Boucle infinie")
15
16 cartes_graphiques_rupture_de_stock = ['GeForce GTX 750Ti']
17 carte_en_rupture = cartes_graphiques_en_stock.pop()
18 cartes_graphiques_rupture_de_stock.append(carte_en_rupture)
19 print(f"La {carte_en_rupture} n'est pas disponible en magasin.")
20
21
```

A context menu is open over the line starting with '6'. The menu items are:

- Python: Exécuter la sélection/la ligne dans Python
- Python: Run Selection/Line in Python Terminal
- PowerShell: Run Selection
- Jupyter: Run Selection/Line in Interactive View
- Python: Exécuter la sélection/la ligne dans Python
- Python: Run Selection/Line in Django Shell
- Terminal: Exécuter le texte sélectionné dans Terminal
- Terminal: Run Selected Text In Active Terminal

The 'Run Selection/Line in Python Terminal' option is highlighted with a red box and the number '3'.

# Exécution de code spécifique

- > « Run Selection/Line in python » lance l'interpréteur Python dans le terminal et lui passe les lignes sélectionner
- > « shift » + « return » exécute la dernière commande de Visual Studio Code
- > On peut continuer d'exécuter les portions de code désirées avec l'interpréteur.

The screenshot shows the Visual Studio Code interface with a Python file named `R2_Demo > /* R2_demo4 copy.py`. The code contains a loop that prints three graphics card models. A selection of the last two lines is highlighted with a yellow background. Below the editor, the terminal tab is active, displaying the output of the executed code:

```
PROBLÈMES 1 SORTIE CONSOLE DE DÉBOGAGE TERMINAL .NET INTERACTI

Copyright (C) Microsoft Corporation. Tous droits réservés.

Testez le nouveau système multiplateforme PowerShell https://aka.ms/pscore-dotnet-get-started

PS C:\Users\pierre-paul.gallant\Cégep Édouard-Montpetit\CMT-420_I> & "C:/Program Files/Python310/python.exe"
Python 3.10.5 (tags/v3.10.5:f377153, Jun  6 2022, 16:14:13) [MSC v.1932 Py310]
Type "help", "copyright", "credits" or "license" for more information
>>> cartes_graphiques_en_stock = ['GeForce RTX 3070Ti',
...                                'Radeon RX 6950 XT',
...                                'Radeon RX 6900 XT']
...
...
...
>>> for carte in cartes_graphiques_en_stock:
...     print(f"La {carte} est disponible en magasin.")
...
...
...
La GeForce RTX 3070Ti est disponible en magasin.
La Radeon RX 6950 XT est disponible en magasin.
La Radeon RX 6900 XT est disponible en magasin.
>>>
```

# Exemple

- > Deux sections de code exécuté sans exécuter les boucles infinies

```
/* R2_demo2.py */ R2_demo1.py /* Ex4_List.py 1 */ /* R2_demo3.py */ /* R2_demo4.py */

R02_Demo > /* R2_demo4 copy.py
1 #execution de lignes spécifiques
2
3 while True:
4     print("Boucle infinie")
5
6 cartes_graphiques_en_stock = ['GeForce RTX 3070Ti', ...
7     ... 'Radeon RX 6950 XT',
8     ... 'Radeon RX 6900 XT']
9 for carte in cartes_graphiques_en_stock:
10    print(f"La {carte} est disponible en magasin.")

11
12
13 while True:
14     print("Boucle infinie")
15
16 cartes_graphiques_rupture_de_stock = ['GeForce GTX 750Ti']
17 carte_en_rupture = cartes_graphiques_en_stock.pop()
18 cartes_graphiques_rupture_de_stock.append(carte_en_rupture)
19 print(f"La {carte_en_rupture} n'est pas disponible en magasin pour l'instant.

PROBLÈMES 1 SORTIE CONSOLE DE DÉBOGAGE TERMINAL .NET INTERACTIVE JUPYTER AZURE
>>> cartes_graphiques_en_stock = ['GeForce RTX 3070Ti',
...                                'Radeon RX 6950 XT',
...                                'Radeon RX 6900 XT']
>>>
>>> for carte in cartes_graphiques_en_stock:
...     print(f"La {carte} est disponible en magasin.")
...
La GeForce RTX 3070Ti est disponible en magasin.
La Radeon RX 6950 XT est disponible en magasin.
La Radeon RX 6900 XT est disponible en magasin.
>>> cartes_graphiques_rupture_de_stock = ['GeForce GTX 750Ti']
>>> carte_en_rupture = cartes_graphiques_en_stock.pop()
>>> cartes_graphiques_rupture_de_stock.append(carte_en_rupture)
>>> print(f"La {carte_en_rupture} n'est pas disponible en magasin pour l'instant.
La Radeon RX 6900 XT n'est pas disponible en magasin pour l'instant.
>>>
```



- > **Beautiful** is better than **ugly**.
- > **Explicit** is better than **implicit**.
- > **Simple** is better than **complex**.
- > **Complex** is better than **complicated**.
- > **Flat** is better than **nested**.
- > **Sparse** is better than **dense**.
- > **Readability** counts.
- > **Special cases** aren't special enough to **break the rules**.
- > Although practicality beats **purity**.
- > **Errors** should never **pass silently**.
- > Unless **explicitly silenced**.
- > In the face of **ambiguity**, refuse the temptation to **guess**.
- > There should be **one** – and preferably only one – **obvious way** to do it.
- > Although that way may not be **obvious at first** unless you're Dutch.
- > **Now** is better than **never**.
- > Although **never** is often better than **right now**.
- > If the implementation is **hard to explain**, it's a **bad idea**.
- > If the implementation is **easy to explain**, it may be a **good idea**.
- > **Namespaces** are one honking **great idea** – let's do more of those!

# Zen of Python



# 2N6 Programmation 2



python™



## Rencontre 2:

- GIT
- Strings
  - Méthodes
  - F-strings
  - Slicing



# GIT

Système de gestion de code source  
<https://info.cegepmontpetit.ca/git>



# Système de gestion de code source



- > Un système de gestion de versions permet de garder une trace de toutes les modifications faites dans un fichier, avec une date et une explication de ce qui a été fait.
- > Même lorsqu'on travaille seul, c'est très utile car en sauvegardant votre code à un certain intervalle de temps, vous ne risquez pas de perdre votre travail.
- > Cela évitera aussi d'avoir à enregistrer et déposer votre travail dans Teams à la fin du cours car en ajoutant votre professeur comme collaborateur, il pourra voir votre travail au besoin.
- > Facilite le travail d'équipe. Le système gestion de versions permet d'éviter la suppression accidentelle de code et de résoudre les conflits facilement.



# Système de gestion de code source

- > <https://education.github.com/pack/join>
- > Utilisez le compte email de l'école
  - > [matricule@cegepmontpetit.ca](mailto:matricule@cegepmontpetit.ca)
- > Vous devez fournir une preuve de votre statut étudiant
  - > ( une capture d'écran de votre grille de cheminement dans LÉA )
- > Créez un nouveau repo pour cette rencontre

The screenshot shows a GitHub user profile for 'Pierre-Paul-Gallant'. At the top, there's a navigation bar with tabs: Overview, **Repositories**, Projects, Packages, and Stars. Below the tabs is a search bar with placeholder text 'Find a repository...'. To the right of the search bar are buttons for Type, Language, Sort, and a green 'New' button. The main content area features a large circular profile picture with a green and white checkered pattern. Below the picture, the user's name 'Pierre-Paul-Gallant' is displayed. A message states 'Pierre-Paul-Gallant doesn't have any public repositories yet.'



# Système de gestion de code source

On ajoute les collaborateurs ( le professeur dans ce cas-ci )

[pierre-paul.gallant@cegepmontpetit.ca](mailto:pierre-paul.gallant@cegepmontpetit.ca)

Ou

Pierre-Paul-Gallant

The screenshot shows a GitHub repository settings page for a private repository named "Pierre-Paul-Gallant / H23-2N6R-R3".

- Step 1:** The "Settings" tab is highlighted with a red circle.
- Step 2:** The "Collaborators" section under the "Access" tab is highlighted with a red circle.
- Step 3:** The "Add people" button in the "Manage access" section is highlighted with a red circle.

The "Who has access" section indicates it's a PRIVATE REPOSITORY where only those with access can view it. The "Manage" link leads to the "Manage access" section, which currently says "You haven't invited any collaborators yet".



# Système de gestion de code source

- > Ouvrez le répertoire documents dans le terminal cmd ou PS
- > Utilisez la commande :  
`git clone ADDRESSE_DU_GIT nom_du_répertoire_local`

```
c:\Documents> git clone https://github.com/Pierre-Paul-Gallant/H23-2N6R-R3 2N6-R3-Exercices
```

- > Téléchargez les fichiers de départ depuis Teams et ajoutez-les au répertoire ainsi créé.

Partage Affichage			
Nom	Modifié le	Type	Taille
Ex3 Videos	2023-01-29 12:08	Dossier de fichiers	
.gitignore	2023-01-29 12:04	Document texte	2 Ko
Ex1 String.py	2023-01-29 09:02	Python File	3 Ko
Ex2_OS.py	2023-01-29 08:32	Python File	3 Ko
Ex3 Renommer plusieurs fichiers.py	2023-01-27 15:21	Python File	1 Ko
README.md	2023-01-29 12:04	Fichier MD	1 Ko



# Système de gestion de code source

- > On ouvre un terminal dans le répertoire.
- > En ligne de commande : \Documents\2N6-R3-Exercices> git add \*
  - > Afin d'ajouter les fichiers au contrôle de code source.
- > On ne veut pas modifier les fichiers videos, on va donc ouvrir le fichier .gitignore et y ajouter le répertoire contenant les videos.

The screenshot shows a Notepad++ window with the following content:

```
*C:\Users\pierre-paul.gallant\Documents\2N6-R3-Exercices\.gitignore - Notepad++
Fichier Édition Recherche Affichage Encodage Langage Paramètres Outils M
Ex1 String.py .gitignore
1 #Videos à ne pas modifier après initialisation
2 Ex3 Videos/
3
4
5 # Byte-compiled / optimized / DLL files
6 __pycache__/
7 *.npy *.npz
```

The file path is displayed at the top: \*C:\Users\pierre-paul.gallant\Documents\2N6-R3-Exercices\.gitignore. The menu bar includes Fichier, Édition, Recherche, Affichage, Encodage, Langage, Paramètres, Outils, and M. The toolbar below the menu has various icons for file operations. The main window shows two tabs: Ex1 String.py and .gitignore. The .gitignore tab contains the ignore patterns listed above.



# Système de gestion de code source

- > On veut maintenant ajouter les fichiers au dépôt en ligne.
- > On fait un " git commit " avec un message informatif.

```
s\2N6-R3-Exercices> git add *
s\2N6-R3-Exercices> git commit -m "Initialisation"
```

- > Suivi d'un " git push " pour envoyer le tout au dépôt en ligne

```
ts\2N6-R3-Exercices> git push
```



# Utilisation de git

- > Si on travaille avec le même dépôt en ligne à partir d'un autre ordinateur et ou d'une autre session de travail.

```
C:\Documents> git clone https://github.com/Pierre-Paul-Gallant/H23-2N6R-R3 2N6-R3-Exercices|
```

- > SI on travaille à plusieurs, on commence par faire un "git pull" même si le répertoire existe déjà. (afin d'avoir les dernières modifications de nos collègues)

```
PS C:\Users\pierre-paul.gallant\Documents\2N6-R3-Exaette> git pull
Already up to date.
PS C:\Users\pierre-paul.gallant\Documents\2N6-R3-Exaette> |
```



# Utilisation de git

> Lorsqu'on fait des modifications, on doit ajouter les fichiers que l'on veut "commit"

> Pour un seul fichier :

```
s\2N6-R3-Exercices> git add '.\Ex1 String.py'
```

> Pour tous les fichiers modifiés:

```
\2N6-R3-Exercices> git add -A
```

> Puis on fait un git commit avec un message court et informatif :

```
\2N6-R3-Exercices> git commit -m "Q1 terminé"
```

> Enfin, on fait un git push pour envoyer le commit sur le dépôt en ligne :

```
s\2N6-R3-Exercices> git push
```



- > Dans ce cours :
- > On fait un commit / push pour **CHAQUE** exercice terminé avec le message commençant par **FCT**
- > S'il y plus de 30 minutes de travail depuis votre dernier commit... faites un commit / push avec le message commençant par **PROGRES**

#### Types de commits

Inscris le type de commit au début du titre.

**FCT** (Fonctionnalité) : un nouvel aspect dans le code fonctionne et augmente la valeur de ton travail / tu as gagné un point du barème.

**PROGRES** (Progrès) : tu n'as pas fini la fonctionnalité mais tu dois t'arrêter.

**BUGFIX** (Débogage) : tu as résolu un bogue. Décris la solution pour l'avoir plus tard.

**NETTOYAGE** (Nettoyage) : tu supprimes du code qui n'est plus nécessaire.

**REFACTOR** (Réorganisation) : tu as changé l'organisation de ton code pour le rendre plus lisible mais le comportement reste le même.

<https://info.cegepmontpetit.ca/git>



# Approfondissement des strings

Ex1 – string.py



# Les strings

- › Les objets « str » peuvent être considérés comme des listes de caractères.
- › Mais ils sont immuables (ne peuvent pas être changés). Les sous strings générés sont de nouveaux objets.
- › Toutes les opérations de « slicing » peuvent être effectuées sur des « strings ».

```
>>> liste1 = "Bonjour le monde"
>>> print(liste1)
Bonjour le monde
```

```
>>> print(liste1[0:5])
Bonjo
>>> print(liste1[:5])
Bonjo
```

```
Bonjo
>>> print(liste1[2:])
njour le monde
```

```
monde
>>> print(liste1[-1])
e
>>> print(liste1[-5:])
monde
```



# Les méthodes des strings

- > `encode()` change l'encodage des caractères du string
- > `find()` et `index()` retournent l'index du premier caractère de la séquence trouvée
- > Les méthodes `is...` retournent une valeur booléenne indiquant si le string correspond à ce que la méthode vérifie.

```
>>> cours = "réseau 1"
>>> cours.encode("utf-8")
```

```
>>> print(cours.find("eau"))
3
```

```
>>> "3".isdigit()
True
```



# Les méthodes des strings

> `split()` / `rsplit()` retournent les portions de strings en séparant le string initial selon un caractère passé en paramètre.

> Peut-être capturé dans une seule variable sous forme de liste.

```
>>> cours = "réseau1/prog1/prog2/prog3"
>>> liste_cours = cours.split("/")
>>> print(liste_cours)
['réseau1', 'prog1', 'prog2', 'prog3']
>>>
```

> Peut aussi être capturé dans plusieurs variables distinctes

```
>>> cours = "réseau1/prog1"
>>> cours1, cours2 = cours.split("/")
>>> print(cours1)
réseau1
>>> print(cours2)
prog1
>>> ■
```



# Les méthodes des strings

- > lstrip() retire les espaces au début du string (left-strip)
- > rstrip() retire les espaces à la fin du string (right-strip)
- > strip() retire les espaces au début et à la fin

```
>>> x = ("  foo  ")  
>>> x.rstrip()  
'  foo'  
>>> x.lstrip()  
'foo  '  
>>> x.strip()  
'foo'
```



# Les méthodes des strings

> `zfill()` ajoute des zéros à gauche d'un string pour obtenir un string d'une taille prédéfinie

```
>>> y = ["1-foo", "2-bar", "10-byr"]
>>> print(y.sort())
None
>>> y = ["1-foo", "2-bar", "10-byr"]
>>> y.sort()
>>> print(y)
['1-foo', '10-byr', '2-bar']
```

```
>>> for i in y : x.append(i.zfill(6))
...
>>> print(x)
['01-foo', '10-byr', '02-bar']
>>> x.sort()
>>> print(x)
['01-foo', '02-bar', '10-byr']
>>>
```



\*ce n'est qu'un exemple de boucle. `zfill()` peut être utilisé sur n'importe quel string



# Sous-strings ( « slicing » )

- > Facile d'accéder à différentes portions des « strings » ou de créer des « sous-strings »
- > En spécifiant la sous-chaine recherchée
- > Ou même en commençant de la fin avec des index négatifs
- > On appelle ces types de manipulation : « slicing »

```
>>> liste1 = "Bonjour le monde"
>>> print(liste1)
Bonjour le monde
```

```
Bonjour le monde
>>> print(liste1[0:5])
Bonjo
>>> print(liste1[:5])
Bonjo
```

```
Bonjour le monde
>>> print(liste1[2:])
njour le monde
```

```
e
>>> print(liste1[-1])
e
>>> print(liste1[-5:])
monde
```

# 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 utilisée 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-Montpetit

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

PS C:\> []





# Introduction aux modules

Ex-2 os.py



# Le module os

- > `os.chdir()` : change le répertoire dans lequel l'interpréteur « agit »
- > `os.getcwd()` : affiche le répertoire courant
- > `os.listdir()` : liste les répertoires et fichiers
- > `os.mkdir()` : crée un répertoire
- > `os.makedirs()` : crée répertoire(s) et sous-répertoire(s)
- > `os.rmdir()` : supprime un répertoire VIDE



# Le module os

- > `os.environ` : dictionnaire des variables de système
  - > `getenv()` et `putenv()` pour obtenir ou modifier ces variables
- > `os.remove()` : supprime un fichier
- > `os.rename()` : renomme un fichier ou répertoire
- > `os.path()` : sous-module pour travailler avec les paths
- > `os.system()` : exécute des commandes dans le shell
- > `os.stat()` : donne des informations sur le fichier / répertoire passé en argument

# 2N6 Programmation 2



python™



Travailler avec des fichiers et CSVs



# Rappel

# Boucles for et listes



> 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



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

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

Liste sur laquelle on itère.

- > Lorsque qu'on arrive à ligne 3 dans cette 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.



▶ DEMO

# 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



96



# Fichiers et Répertoires

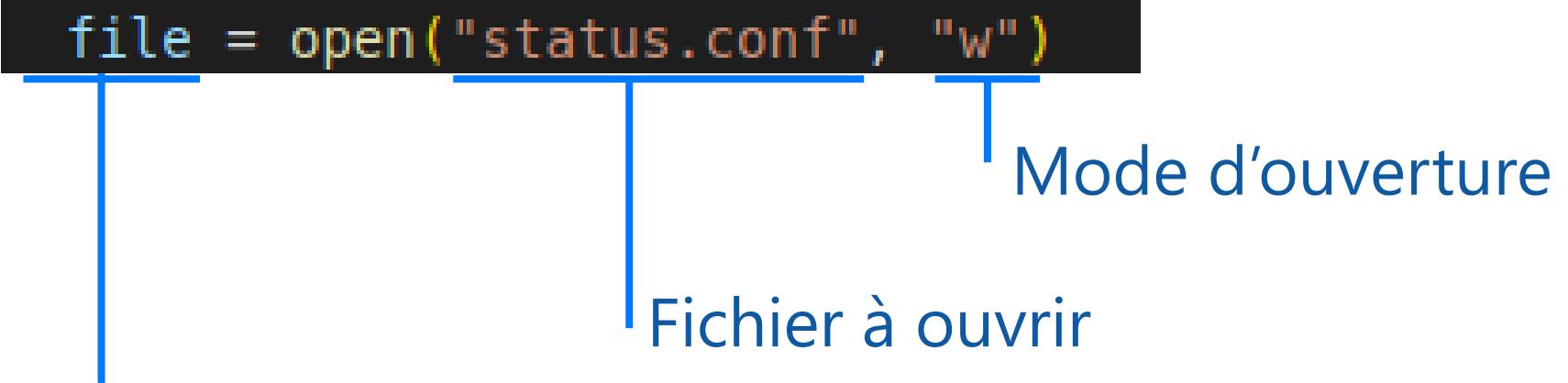
Comment interagir avec les fichiers avec python



# 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")
```



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()

- > Retourne un objet correspondant au fichier ouvert avec ses propres méthodes.

```
1  file = open("status.conf","r")
2  file.read()      #lis le reste du contenu du fichier
3  file.read(10)    #lis les 10 caractères suivants dans le fichier
4  file.readline() #lis la prochaine ligne
5  file.readable() #retourne booléen si lisible
6
7  file = open("status.conf","w")
8  file.write("texte") #écrit "texte" dans le fichier
9  file.writable()    #retourne bouléean si écrivable
10
11 # Peu importe comment on crée l'objet file
12 file.seek(7)      #place le pointeur au 7ième caractère du fichier
13 file.seek(0)       #retourne le pointeur au début du fichier
14 file.tell()        #indique la position du pointeur
15
16 file.close()      # ferme le fichier
```



# L'instruction "with"

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

## Mauvais et Dangereux

```
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
- > Assignation 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

```
➊ 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



# 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 ;



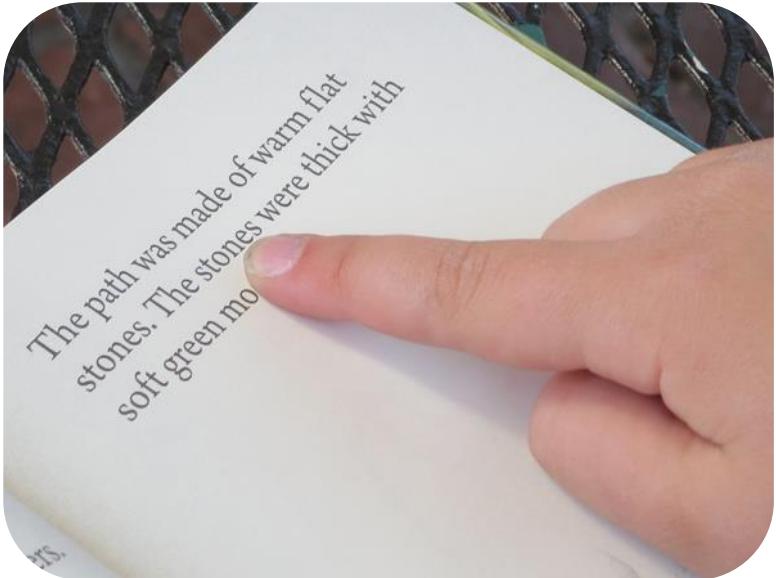
# Travailler avec les CSVs

- > 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



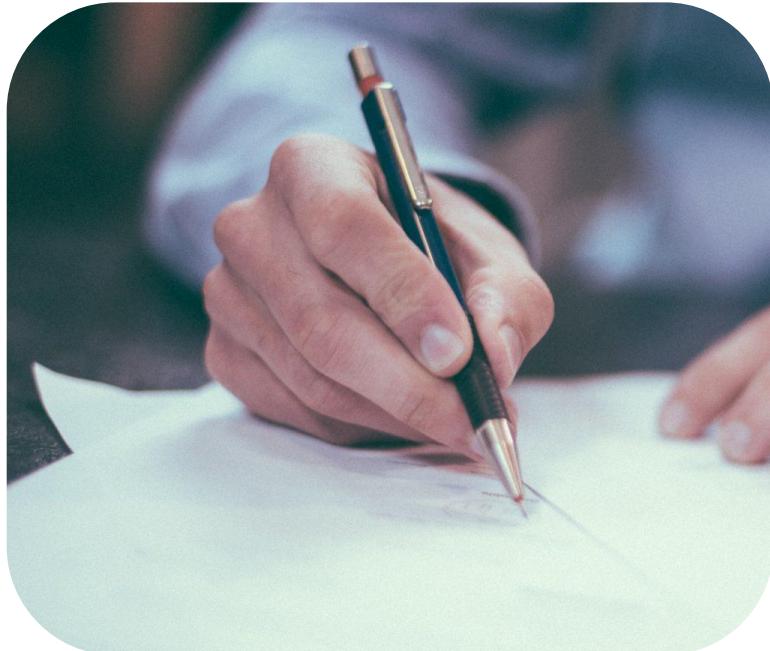
- > Pour lire un csv. On commence par instancier un objet à partir de la classe « reader » avec le fichier qu'on désire lire en paramètre.
- > Cet objet est itérable et peut donc être utilisé avec une boucle for.

```
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)
```

# B73 Scripting sous Linux



python™



Fonctions, modules, et librairie standard



# Les fonctions



# Les fonctions

- > Blocs de code réutilisable.
- > Permet de structurer et d'organiser le code en séparant les tâches en sous-tâches plus simples.
- > Rend le code plus lisible en donnant des noms significatifs aux différentes tâches.
- > Peuvent prendre des paramètres d'entrée (ou pas) et renvoyer des résultats (ou pas). Permet de les utiliser de manière flexible dans différentes parties du programme.
- > Rend le code plus maintenable.

```
1 def message_invitation(nom):  
2     print(f"""\nBonjour {nom} et bienvenue au cours de programmation 2.  
3         \nProgrammation objet pour profil résautique\n""")  
4     ....
```





# Fonctions sans paramètres

- > Toutes les fonctions commencent avec le mot-clef « def » suivie du nom de la fonction et des parenthèses qui prennent ou non des arguments.
- > Cette fonction est sans paramètre.

```
>>> def bonjour():  
...     print ("Bonjour")  
...  
>>> bonjour()  
Bonjour
```

Définition de la fonction

Appel de la fonction

Résultat obtenu



# Fonctions avec paramètres

- Les fonctions avec paramètres reçoivent une valeur pour chaque paramètre lors de l'appel.

Définition de la fonction

```
def bonjour_toi(ami):  
    print(f"Bonjour {ami}")
```

```
bonjour_toi("Steve Buscemi")
```

Appel de la fonction

Valeur passée à la fonction pour le paramètre ami



# Fonctions avec paramètres

- Les fonctions avec paramètres peuvent comporter des valeurs par défaut. La variable prendra cette valeur si aucune n'est fournie lors de l'appel de la fonction.

Appel de la fonction avec et sans valeur donnée pour ami.

```
def bonjour_toi(ami="mon ami"):  
    print(f"Bonjour {ami}.")  
  
bonjour_toi("Steve Buscemi")  
bonjour_toi()  
✓ 0.5s  
  
Bonjour Steve Buscemi.  
Bonjour mon ami.
```

Valeur par défaut pour le paramètre ami.



# Fonctions avec valeur de retour

- > Quand une fonction retourne une valeur (avec return), il faut capturer cette valeur dans une variable lors de l'appel.
- > Le retour du premier appel n'a pas été capturé. Le message produit n'est pas utilisable ailleurs dans le code.
- > C'est comme faire un appel à quelqu'un et ne pas écouter sa réponse.  
(ou demander une question au prof et ne pas écouter la réponse)

```
19
20     def bonjour_toi(ami="mon ami"):
21         return(f"Bonjour {ami}.")
22
23 bonjour_toi("Steve Buscemi")
24
25 a = bonjour_toi()
26 print(a)
27
```

PROBLÈMES SORTIE CONSOLE DE DÉBOGAGE TERMINAL

R07\_Exercices\_Solution/R07\_ex1\_Fonctions.py

Bonjour mon ami.

# Les fonctions

- › Peuvent agir différemment selon les valeurs passées.
- › Les fonctions peuvent appeler d'autres fonctions, encore une fois pour éviter les répétitions, et faciliter la lisibilité et maintenance du code.

```
* exemples.py > ...
1  def impression_liste(texte="Liste : ", liste=None):
2      if liste == None:
3          print("Oops ! utilisé cette fonction avec une liste")
4      else:
5          print(texte)
6          for valeur in liste:
7              print(f"ID: {id(valeur)} - Cours: {valeur}")
8          print("")
```

```
* exemples.py > ⌂ impression_liste_et_id > [❷] liste
1  def impression_liste_et_id(liste):
2      for valeur in liste:
3          print(f"ID: {id(valeur)} | {valeur}")
4
5
6  def impression_liste_avec_titre(texte="Liste : ", liste=None):
7      if liste == None:
8          print("Oops ! utilisé cette fonction avec une liste")
9      else:
10         print(texte)
11         impression_liste_et_id(liste)
12         print("")
```





# Modules

- > Scripts de code Python réutilisables dans différents programmes.
- > Regroupent fonctions, variables et autres éléments ayant une fonctionnalité similaire.
- > Code mieux organisé et plus facile à maintenir.
- > Différentes parties du code séparées en modules distincts.
- > Python possède la librairie standard qui regroupe de nombreux modules prédéfinis et prêts à l'emploi. Donc le code est déjà écrit et testé.



# Modules

- > Il y a plusieurs façons de faire l'import, les deux premières sont équivalentes et très bonnes.
- > On peut renommer les modules pour simplifier notre code, surtout si les modules ont des noms long ou complexe.
- > Les syntaxes aux lignes 3 et 4 sont valides mais peuvent causer des conflits.

```
/* test.py
1  import subprocess
2  import subprocess as sb
3  from subprocess import run
4  from subprocess import *
5
6  sb.run(["echo", "test sb"], shell=True)
7
```



# Modules

> Utiliser des modules est très facile. On les utilise déjà dans le cours.

```
import os
utilisateur = os.getenv("USER")
```

> Créer de nouveaux modules est aussi simple que d'écrire un script

The screenshot shows a code editor with two files open:

- utilitaire\_admin\_CEM.py**: A module containing functions for adding and removing users via the subprocess module.
- petit\_script\_rapide.py**: A script that imports the module and uses its functions to add users from a CSV file.

```
EXPLORATEUR ... /* utilitaire_admin_CEM.py */
...
MODULES
/* petit_script_rapide.py */
/* utilitaire_admin_CEM.py */

/* utilitaire_admin_CEM.py > supprimer_utilisateurs
1 import subprocess
2 def ajout_utilisateur(nom="",password=""):
3     if nom == "":
4         nom = input("Entrez nom d'utilisateur : ")
5     if password == "":
6         password = input("Entrez mot de passe : ")
7     subprocess.run(["useradd","-p", nom, password])
8
9 def supprimer_utilisateurs(nom=""):
10    if nom == "":
11        nom = input("Entrez nom d'utilisateur : ")

/* petit_script_rapide.py > ...
1 #importe notre module fait maison
2 import utilitaire_admin_CEM
3 import csv
4 #ouvre un csv et exécute une fonction provenant du module
5 with open("liste_nouveaux_utilisateurs.csv","r",) as fichier_users:
6     csv_read = csv.reader(fichier_users)
7     for ligne in fichier_users:
8         nom = ligne[0]
9         mot_de_passe = ligne[1]
10        utilitaire_admin_CEM.ajout_utilisateur(nom, mot_de_passe)
```



# Range / scope des variables

- > Quand une variable est définie globalement elle est disponible dans le script et dans les fonctions de ce script.
- > Mais quand une variable est redéfinie localement, la variable globale ne change pas. Cela fait juste une autre variable qui est locale à la fonction.
- > Lorsqu'on définit des variables dans une fonction, elles sont locales à cette fonction. Ce qui signifie qu'elles n'existent que dans la fonction.
- > Lorsqu'on termine l'exécution d'une fonction, les variables locales à cette fonction cessent d'exister.





# Le module os

- > environ : dictionnaire des variables de système
  - > Getenv() et putenv() pour obtenir ou modifier ces variables
- > os.Chdir() : change le répertoire dans lequel l'interpréteur « agit »
- > os.Getcwd() : affiche le répertoire courant
- > os.mkdir() : crée un répertoire
- > os.rmdir() : supprime un répertoire VIDE
- > os.remove() : supprime un fichier
- > os.rename() : renomme un fichier ou répertoire
- > os.path() : sous-module pour travailler avec les paths
- > os.system() : exécute des commandes dans le shell



# Le module sys

- > `sys.argv` : contient les valeurs passées en arguments
  - > (souvent utilisé avec le module argparse)
- > `sys.exit()` : permet de sortir d'un script de façon « propre » et en envoyant un message de retour
- > `sys.platform` : indique la plateforme dans laquelle le script est exécuté (si on veut des parties différentes exécutées dans linux et window)
- > `sys.path` : liste des répertoires où l'interpréteur recherche les modules



# Le module subprocess

- > `subprocess.run(*arg, shell=True)`
  - > Permet de passer des commandes au shell à partir du code python.
  - > Très utile pour la gestion de systèmes
- > `Subprocess.Popen` : Permet de lancer de nouveaux processus et d'interagir avec.

# 2N6 Programmation 2



python™





# Dictionnaire



# Dictionnaires

- > Similaire aux listes, les dictionnaires sont des structures de données qui peuvent stocker plusieurs valeurs.
- > Les valeurs sont associées à une "clef" qui permet de récupérer les valeurs.

Clef : valeur

```
auto = { "marque": "Ford",
          "modele": "Mustang",
          "annee
```



# Dictionnaires

Clef : valeur

```
auto = { "marque": "Ford",
          "modele": "Mustang",
          "annee": 1964 }
```

```
print(auto)
# {'marque': 'Ford', 'modele': 'Mustang', 'annee': 1964}
```

```
print(auto['marque'])
# Ford
```

```
print(f"{auto['marque']} {auto['modele']} {auto['annee']}")
# Ford Mustang 1964
```



# Dictionnaires - Méthodes

- > `dict.get("clef")` retourne la valeur de la clef dans le dictionnaire
  - > `auto.get("modele") → "Mustang"`      `auto = { "marque": "Ford", "modele": "Mustang", "annee": 1964 }`
  
- > `dict["clef"] = valeur` Change la valeur à correspondant à la clef. Si elle n'existe pas, ajoute la paire clef:valeur
  - > `auto["annee"] = 1968`      `auto → { "marque": "Ford", "modele": "Mustang", "annee": 1968 , "couleur": "rouge"}`
  - > `auto["couleur"] = "rouge"`



# Dictionnaires - Méthodes

> dict.update( {dictionnaire} ) update peut ajouter des paires clef:valeur ou changer la valeur de clefs existantes ou être utilisé pour concaténer des dictionnaires

> auto.update(ajout)

```
auto = { "marque": "Ford",  
         "modele": "Mustang",  
         "annee": 1964 }
```

```
ajout = { "puissance": 7800,  
          "couleur": "rouge"}
```





# Dictionnaires - Méthodes

> del dict["clef"] retire la paire clef:valeur du dictionnaire.

> del auto["annee"]

```
auto = { "marque": "Ford",  
        "modele": "Mustang",  
        "annee": 1964 }
```



```
auto → { "marque": "Ford",  
        "modele": "Mustang" }
```

> dict.pop["clef"] retire la paire clef:valeur du dictionnaire et retourne la valeur uniquement.

> annee\_fabrication = auto.pop("annee")

```
auto = { "marque": "Ford",  
        "modele": "Mustang",  
        "annee": 1964 }
```



```
auto → { "marque": "Ford",  
        "modele": "Mustang" }
```

annee\_fabrication → 1964



# Dictionnaires - Méthodes

- > `len(dict)` la fonction `len` nous retourne le nombre total de paires clef:valeur dans le dictionnaire.
  - > `nb_auto = len(auto)`
  - > `nb_auto → 3`
- > `dict.keys()` retourne toutes les clefs dans le dictionnaire.
  - > `clef = auto.keys()`
  - > `clef → dict_keys(['marque', 'modele', 'annee'])`
- > `dict.values()` retourne toutes les valeurs dans le dictionnaire.
  - > `valeurs = auto.values()`
  - > `valeurs → dict_values(['Ford', 'Mustang', '1964'])`

```
auto = { "marque": "Ford",
          "modele": "Mustang",
          "annee": 1964 }
```



# Dictionnaires - Méthodes

- > Les méthodes `keys()` et `values()` nous redonne des objets itérables.  
On peut donc passer au travers avec une boucle `for`.

```
for clef in auto.keys():
    print(clef)
```

*# marque  
# modèle  
# année*

```
for valeur in auto.values():
    print(valeur)
```

*# Ford  
# Mustang  
# 1964*



# Dictionnaires - Méthodes

- > `dict.items()` permet d'obtenir toutes les paires clef:valeur dans un objet itérable

```
# On peut obtenir toutes les clés:valeurs dans notre dictionnaire avec la méthode .items()
print(etudiant.items())
#dict_items([('nom', 'Lucie'), ('cours', ['Reseau 1', 'Prog 2 en Python']), ('Tel', '514-321-1234')])

# Pour passer à travers toutes les paires clés:valeurs de notre dictionnaire
for key, value in etudiant.items():
    ... print(key,value)
#nom Lucie
#cours ['Reseau 1', 'Prog 2 en Python']
#Tel 514-321-1234
```



# Liste dans un dictionnaire

- > Les dictionnaires et les listes sont souvent utilisés ensemble pour permettre de stocker de nombreuses données de façon flexible.
- > Ici, un dictionnaire représentant une auto et contenant une liste d'accessoires.

```
auto = {  
    "marque": "Reliant",  
    "modele": "Robin",  
    "annee": 1988,  
    "accessoires": [  
        "Marchepied chromé",  
        "Moteur V8",  
        "Dés en minou sur le rétroviseur"  
    ]  
}  
  
print(f"Il y a {len(auto['accessoires'])} accessoires:")  
  
for item in auto['accessoires']:  
    print("- " + item)  
  
# Il y a 3 accessoires:  
# - Marche-pied chromé  
# - Moteur V8  
# - Dés en minou sur le rétroviseur
```

A red rectangular box highlights the list value of the 'accessoires' key in the dictionary. A red arrow points from the word 'Liste' at the bottom right to the opening bracket of the list [.



# Dictionnaires dans une liste

- > Ici, une listes de autos. Chaque voiture est représentée par un dictionnaire.
- > Il n'y a pas de limites aux « niveau de profondeur » des dictionnaires et listes.

```
autos = [
    {"marque": "Ford", "modele": "Mustang", "annee": 1964},
    {"marque": "Reliant", "modele": "Robin", "annee": 1988},
    {"marque": "Toyota", "modele": "Tercel", "annee": 1991}
]

print(f"Il y a {len(autos)} autos:")
for auto in autos:
    print(f"- {auto['marque']} {auto['modele']} {auto['annee']}")

# Il y a 3 autos:
# - Ford Mustang 1964
# - Reliant Robin 1988
# - Toyota Tercel 1991
```



# JSON



# Formats de sérialisation

## JSON

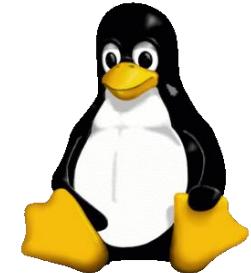
```
{  
    "first_name": "John",  
    "last_name": "Smith",  
    "age": 25,  
    "address": {  
        "street_address": "21 2nd Street",  
        "city": "New York",  
        "state": "NY",  
        "postal_code": "10021"  
    },  
    "phone_numbers": [  
        {  
            "type": "home",  
            "number": "212 555-1234"  
        },  
        {  
            "type": "fax",  
            "number": "646 555-4567"  
        }  
    ],  
    "sex": {  
        "type": "male"  
    }  
}
```

## XML

```
<person>  
    <firstName>John</firstName>  
    <lastName>Smith</lastName>  
    <age>25</age>  
    <address>  
        <streetAddress>21 2nd Street</streetAddress>  
        <city>New York</city>  
        <state>NY</state>  
        <postalCode>10021</postalCode>  
    </address>  
    <phoneNumbers>  
        <phoneNumber>  
            <type>home</type>  
            <number>212 555-1234</number>  
        </phoneNumber>  
        <phoneNumber>  
            <type>fax</type>  
            <number>646 555-4567</number>  
        </phoneNumber>  
    </phoneNumbers>  
    <sex>  
        <type>male</type>  
    </sex>  
</person>
```

## YAML

```
first_name: John  
last_name: Smith  
age: 25  
address:  
    street_address: 21 2nd Street  
    city: New York  
    state: NY  
    postal_code: "10021"  
phone_numbers:  
    - type: home  
        number: 212 555-1234  
    - type: fax  
        number: 646 555-4567  
sex:  
    type: male
```



Souvent utilisé  
dans les fichiers de  
configuration Linux



- > JSON (JavaScript Object Notation) est un format standard de sérialisation d'objets sous forme de données textuelles.
- > Permet de stocker et transmettre des objets dans un format standard, indépendant du langage.
- > Format indépendant et ouvert, créé au début des années 2000 et standardisé en 2017.
- > Présentement le standard le plus répandu pour les échanges de données entre les applications.



```
[  
  {  
    "marque": "Ford",  
    "modele": "Mustang",  
    "annee": 1964,  
    "accessoires": []  
  },  
  {  
    "marque": "Reliant",  
    "modele": "Robin",  
    "annee": 1988,  
    "accessoires": [  
      "Moteur V8",  
      "Dés en minou"  
    ]  
  },  
  {  
    "marque": "Toyota",  
    "modele": "Tercel",  
    "annee": 1991,  
    "accessoires": []  
  }]
```

## Les **listes** sont entre **crochets** [ ]

- > Une liste peut contenir des valeurs brutes, des dictionnaires, ou d'autres tableaux.

## Les **dictionnaires** sont entre **accolades** { }

- > Un dictionnaire est composé d'un ou plusieurs champs composés d'une clé et une valeur.
- > La valeur d'un champ peut être une chaîne de caractères, un nombre, un tableau ou un dictionnaire.

Chaque élément est séparé des autres par des virgules

# 2N6 Programmation 2



python™



Les fichiers JSON et les requêtes http



# JSON

# JSON



```
[  
  {  
    "marque": "Ford",  
    "modele": "Mustang",  
    "annee": 1964,  
    "accessoires": []  
  },  
  {  
    "marque": "Reliant",  
    "modele": "Robin",  
    "annee": 1988,  
    "accessoires": [  
      "Moteur V8",  
      "Dés en minou"  
    ]  
  },  
  {  
    "marque": "Toyota",  
    "modele": "Tercel",  
    "annee": 1991,  
    "accessoires": []  
  }]
```

- > Ici, on a 3 objets JSON correspondant à des voitures.
- > Ils sont énumérés de façon séquentielle, séparés par des virgules (",") et contenus dans des crochets. [ ]
- > Une fois convertis en objet dans Python, il s'agira d'une liste de dictionnaires ayant tous les mêmes clefs.



```
[  
  {  
    "marque": "Ford",  
    "modele": "Mustang",  
    "annee": 1964,  
    "accessoires": []  
  },  
  {  
    "marque": "Reliant",  
    "modele": "Robin",  
    "annee": 1988,  
    "accessoires": [  
      "Moteur V8",  
      "Dés en minou"  
    ]  
  },  
  {  
    "marque": "Toyota",  
    "modele": "Tercel",  
    "annee": 1991,  
    "accessoires": []  
  }]
```

## Les **listes** sont entre **crochets** [ ]

- > Une liste peut contenir des valeurs brutes, des dictionnaires, ou d'autres listes.

## Les **dictionnaires** sont entre **accolades** { }

- > Un dictionnaire est composé d'un ou plusieurs champs composés d'une clé et une valeur.
- > La valeur d'un champ peut être une chaîne de caractères, un nombre, une liste ou un dictionnaire.

Chaque élément est séparé des autres par des virgules



# Convertir du contenu JSON en objet

- › La méthode **loads()** du module **json** convertit du texte formaté JSON en objet natif Python
- › Cet objet peut représenter une hiérarchie d'objets listes et dictionnaires

Reponse\_req.json

```
1  [{"marque": "Ford", "modele": "Must  
2   ang", "annee": 1964, "accessoires":  
3     []}, {"marque": "Reliant", "modele":  
4      "Robin", "annee": 1988, "accessoir  
5      es": ["MoteurV8", "Désenminou"]}, {  
6        "marque": "Toyota", "modele": "Terc  
7        el", "annee": 1991, "accessoir  
es": []}, {"marque": "Relia...
```

json.loads()

```
[  
  {  
    "marque": "Ford",  
    "modele": "Mustang",  
    "annee": 1964,  
    "accessoires": []  
  },  
  {  
    "marque": "Reliant",  
    "modele": "Robin",  
    "annee": 1988,  
    "accessoires": [  
      "Moteur V8",  
      "Dés en minou" ]  
  }]  
]
```



# Convertir du contenu JSON en objet

```
import json

with open('Reponse_req.json', 'r') as file:
    text = file.read()

type(text)      # <class 'str'>
# C'est du texte brut

autos = json.loads(text)
# Convertit le contenu JSON en objet

type(autos)      # <class 'list'>
type(autos[0])   # <class 'dict'>
# C'est une liste de dictionnaires

print(autos[0]['marque'])
# Ford
```

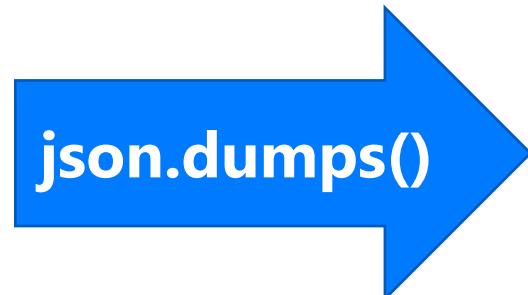
- > Dans cet exemple on lit un fichier texte appelé « Reponse\_req.json » et on met le contenu (un string) dans la variable « text ».
- > On utilise ensuite la fonction json.loads() pour convertir le string en un objet utilisable dans Python. Ici, une liste contenant des dictionnaires.



# Convertir du contenu JSON en objet

- > La méthode **dumps()** du module **json** prend un objet dans Python et le transforme en une chaîne de caractères suivant le format JSON
- > On peut ensuite enregistrer cette chaîne de caractères ou l'envoyer dans une requête http.

```
[  
  {  
    "marque": "Ford",  
    "modele": "Mustang",  
    "annee": 1964,  
    "accessoires": []  
  },  
  {  
    "marque": "Reliant",  
    "modele": "Robin",  
    "annee": 1988,  
    "accessoires": [  
      "Moteur V8",  
      "Dés en minou"  
    ]  
  }]
```



```
[{"marque": "Ford", "modele": "Mustang", "annee": 1964, "accessoires": []}, {"marque": "Reliant", "modele": "Robin", "annee": 1988, "accessoires": ["MoteurV8", "Désenminou"]}, {"marque": "Toyota", "modele": "Tercel", "annee": 1991, "accessoires": []}, {"marque": "Relia..."]
```



# Convertir un objet en contenu JSON

- La méthode **dumps()** peut aussi prendre une valeur pour son paramètre « indent » afin de rendre la chaîne de caractères facilement lisible par l'être humain.

```
autos = [
    {
        "marque": "Ford",
        "modele": "Mustang",
        "annee": 1964,
        "accessoires": []
    },
    {
        "marque": "Reliant",
        "modele": "Robin",
        "annee": 1988,
        "accessoires": [
            "Moteur V8",
            "Dés en minou"
        ]
    }
]
```

```
print(json.dumps(autos,
                  indent=4))
```

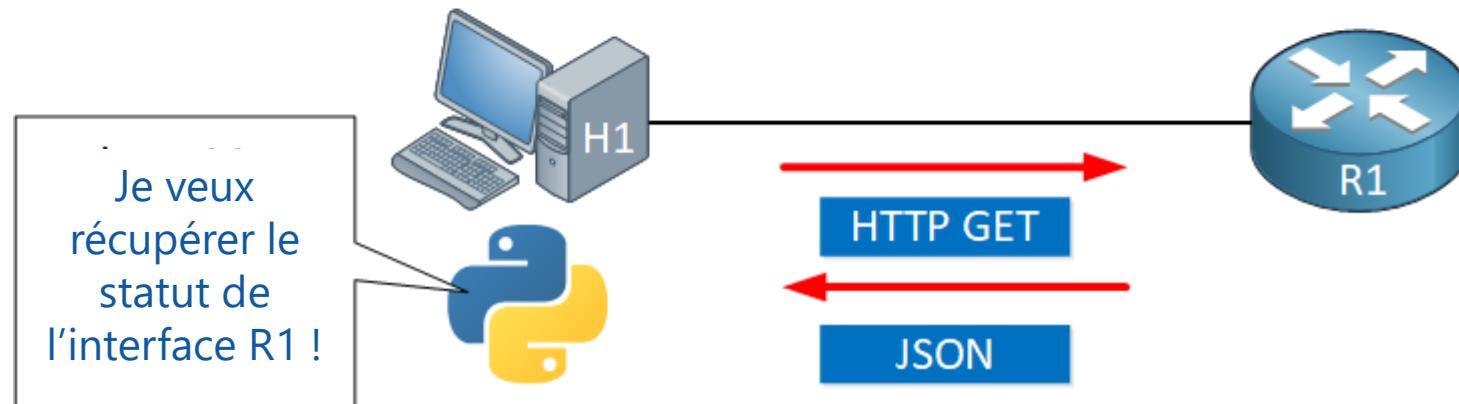
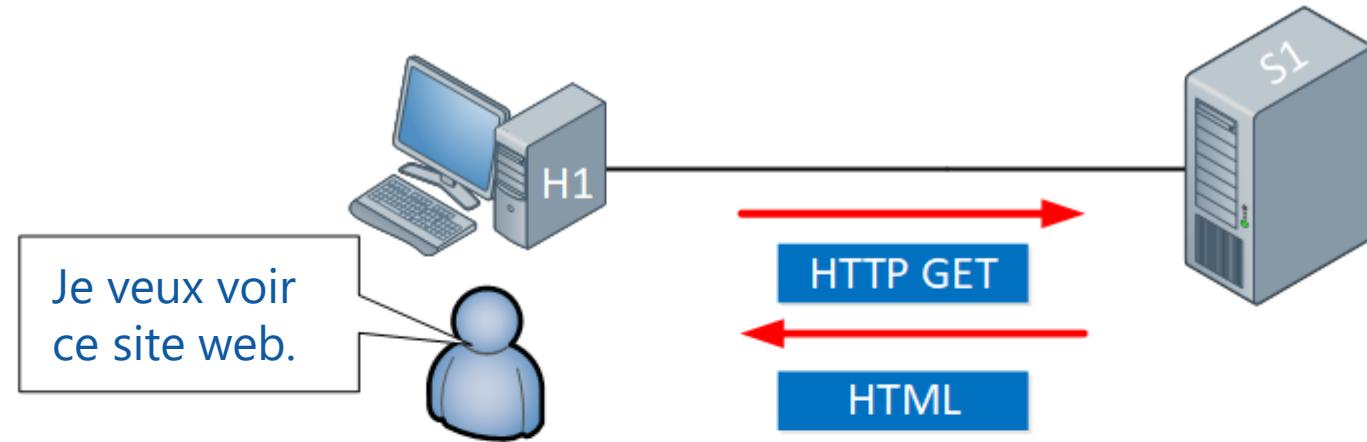
```
>>> print(json.dumps(autos, indent=4))
[{
    {
        "marque": "Ford",
        "modele": "Mustang",
        "annee": 1964,
        "accessoires": []
    },
    {
        "marque": "Reliant",
        "modele": "Robin",
        "annee": 1988,
        "accessoires": [
            "MoteurV8",
            "D\u00e9s en minou"
        ]
    }
]
>>> -
```



# API Web



# Transmission de données

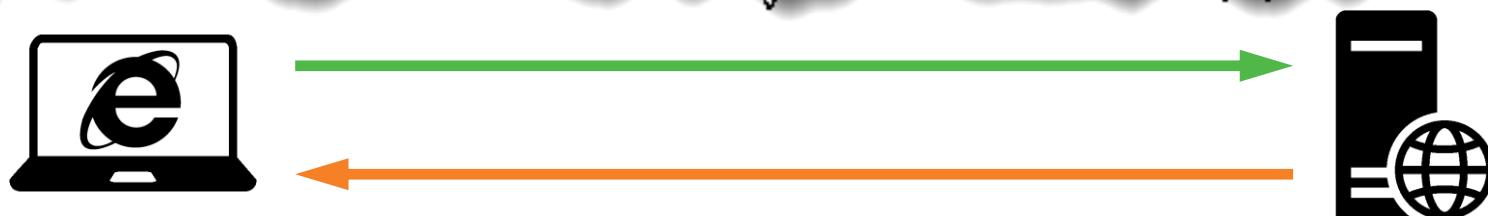


Source: NetworkLessons.com



# Échange HTTP

```
GET https://www.cegepmontpetit.ca/ HTTP/1.1
Accept: text/html, application/xhtml+xml, image/jxr, */*
Accept-Language: en-US,en;q=0.5
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; Touch; rv:11.0) like Gecko
Accept-Encoding: gzip, deflate
Host: www.cegepmontpetit.ca
Connection: Keep-Alive
```



```
HTTP/1.1 200 OK
Date: Thu, 10 Oct 2019 04:25:29 GMT
Server: Apache-Coyote/1.1
Content-Type: text/html; charset=UTF-8
Set-Cookie: JSESSIONID=66BB8CA8BDAF7102EB1CF89B569BDF57; Path=/; HttpOnly
Vary: Accept-Encoding
Connection: close
Content-Length: 59803

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"><html
xmlns="http://www.w3.org/1999/xhtml" xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr"
lang="fr"><head><title>Bienvenue ! | Cégep Édouard-Montpetit</title><meta name="description"
content="Cégep Édouard Montpetit"></meta><meta http-equiv="Content-Type" content="text/html;
charset=UTF-8"><meta name="viewport" content="width=device-width, initial-scale=1.0" />
```



# Toutes les Méthodes HTTP

Méthode	Description
GET	Demande une ressource (un fichier, un objet, etc.)
POST	Envoie des données à une ressource (la requête a un <i>payload</i> )
PUT	Crée ou remplace une ressource sur le serveur
HEAD	Ne demande que les informations sur la ressource, pas son <i>payload</i>
DELETE	Supprime une ressource du serveur
OPTIONS	Obtient les options d'une ressource du serveur
TRACE	Demande au serveur de retourner ce qu'il a reçu, à des fins de diagnostic
CONNECT	Permet d'ouvrir un tunnel de communication (par exemple, SSL)
PATCH	Modifie une ressource, comme PUT, mais partiellement

Plus de détails: <https://www.iana.org/assignments/http-methods/http-methods.xhtml>



# La méthode GET

Méthode	Description
GET	Demande une ressource (un fichier, un objet, etc.)

> Une requête http est un message qu'on envoie à une adresse url avec un format standard indiquant les données avec lesquelles on veut interagir et la façon dont on veut traiter ces données.

```
response = requests.get('https://fakestoreapi.com/products?limit=5&sort=desc')
```

Adresse url où on  
envoie notre requête

Paramètres de notre  
requête



# Utilisation de la méthode GET

## > Exemples de requêtes effectuées à [fakestoreapi.com](https://fakestoreapi.com)

```
_req.py > ...
import requests as rq
import json

res = rq.get('https://fakestoreapi.com/products') → Tous les produits

res = rq.get('https://fakestoreapi.com/products/9') → Le produit avec l'ID 9

res = rq.get('https://fakestoreapi.com/products?limit=5') → Les 5 premiers produits

res = rq.get('https://fakestoreapi.com/products?limit=5&sort=desc') → Les 5 premiers produits en ordre décroissant d'ID
```

## > Toujours consulter la documentation pour formuler une requête



<https://fakestoreapi.com/docs>



# Exemple détaillé en Python

## Python

```
import json, requests  
  
url = "http://date.jsontest.com/"  
response = requests.get(url)  
data = json.loads(response.text)  
  
print(f"Il est { data['time'] } sur le serveur !")
```

# IL est 02-20-2023 sur le serveur

Il faut installer le module **requests** avec:

pip install requests

Il faut s'assurer d'avoir pip déjà installé :

python get-pip.py



# La réponse...

- > La commande `requests.get()` nous retourne un objet de la classe `<Response>`.
- > Cet objet possède ses propres attributs et méthodes.
- > On s'intéresse particulièrement aux attributs « `status_code` », « `ok` », « `text` », « `headers` » et la méthode « `json()` ».



# Réponse

```
>>> response = requests.get("http://date.jsontest.com/")

>>> print(response.status_code)
200

>>> print(response.ok)
True

>>> print(response.text)
{
    "date": "12-07-2020",
    "milliseconds_since_epoch": 1607327636741,
    "time": "07:53:56 AM"
}

>>> print(response.json())
{"date": "12-07-2020", "milliseconds_since_epoch": 1607327636741, "time": "07:53:56 AM"}

>>> print(response.headers)
{'Access-Control-Allow-Origin': '*', 'Content-Type': 'application/json', 'X-Cloud-Trace-Context':
'd9a3a2f216dc37228d6ae1f376eece11', 'Date': 'Mon, 07 Dec 2020 07:53:56 GMT', 'Server': 'Google Frontend',
'Content-Length': '100'}
```



# Exemple: données météo

```
import datetime, json, requests

uri = "https://www.metaweather.com/api/location"

woeid = json.loads(
    requests.get(f"{uri}/search/?query=montr%C3%A9al").text)[0]['woeid']
    today = datetime.datetime.now().strftime('%Y/%m/%d')
data = json.loads(
    requests.get(f"{uri}/{woeid}/{today}").text
)

for item in data:
    print(f"{item['created']} : {item['the_temp']}°C")
```

# Fakestore ➔ Une ressource pour exemples et exercices



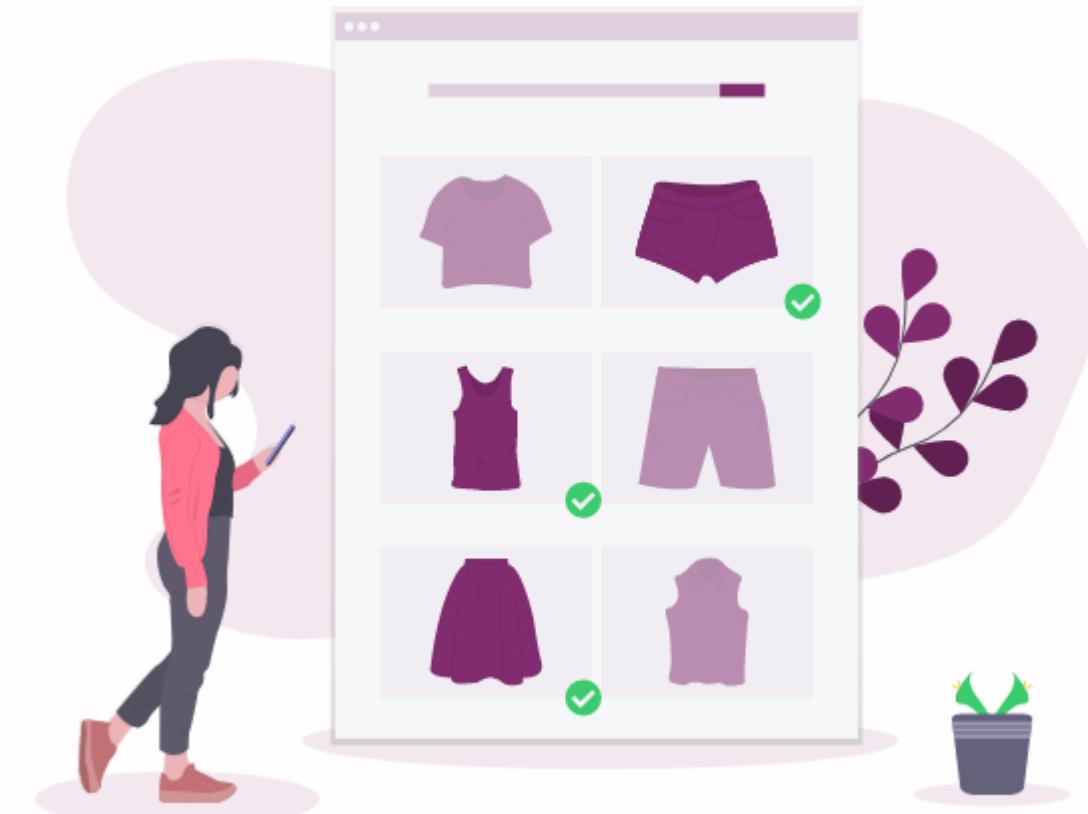
## Fake Store API

Fake store rest API for your e-commerce or shopping website prototype

[View on GitHub](#)



[Read Docs](#)



# Fakestore

- > <https://fakestoreapi.com/>
- > Fake store api est un site web contenant un api avec lequel on peut interagir comme si nous avions toutes les autorisations.
- > Pas limité au GET seulement
- > On peut faire des POST, PUT, PATCH et DELETE

## Routes

All HTTP methods are supported

GET	/products
GET	/products/1
GET	/products/categories
GET	/products/category/jewelery
GET	/cart?userId=1
GET	/products?limit=5
POST	/products
PUT	/products/1
PATCH	/products/1
DELETE	/products/1

[View Details on Docs](#)



# Pour en savoir plus

- > [API Integration in Python – Part 1 – Real Python](#)
- > <https://apislist.com/>
- > <http://www.jsonplaceholder.typicode.com/>
- > [Util: Fiddler](#)
- > [Liste d'API publics pour s'amuser](#)
- > [Authentication using Python requests - GeeksforGeeks](#)

# 2N6 Programmation 2



avec python™



Les fonctions et modules



# Modules

Nous avons vus comment utiliser des modules.  
Maintenant nous allons voir comment en créer.



# Utilisation de modules

```
> # Création de répertoires  
> import os
```

```
> cours='420-2N6R'  
> groupe='1070'  
> os.makedirs(cours+'/'+groupe)  
>
```

> Ici, on importe le module os.

> os.makedirs utilise la fonction  
makedirs() du module os



# Utilisation de modules

```
> # Fake Store REST API
> import requests
> import json
>
> BASE_URL = 'https://fakestoreapi.com'
>
> # faire une demande de tous les produits (GET)
> response = requests.get(f"{BASE_URL}/products")
> donne_req = response.json()
> print( json.dumps( donne_req, indent=4 ) )
```

- > Ici, on importe les modules `requests` et `json`
- > `requests.get` utilise la fonction `get()` du module `request`
- > Puis on utilise la méthode `.json()` de l'objet `response` du module `requests`
- > `Json.dumps` utilise la fonction `dumps()` du module `json`



# Modules

- > Fichiers de code Python réutilisable dans différents programmes.
- > Regroupe fonctions, variables et autres éléments ayant une fonctionnalité similaire.
- > Code mieux organisé et plus facile à maintenir.
- > Différentes parties du code séparées en modules distincts.
- > Python possède la librairie standard qui regroupe de nombreux modules prédéfinis et prêts à l'emploi. Donc le code est déjà écrit et testé.



# Modules qu'on a vu et utilisés

- > **os** qui regroupe des fonctions pour le système d'opération
- > **csv** qui regroupe des fonctions pour traiter les données d'un fichier csv
- > **requests** qui regroupe des fonctions pour faire des requêtes http
- > **json** qui regroupe des fonctions pour traiter ce type de données



# Création d'un module

- > Si on réutilise le même code dans plusieurs scripts, on va faire un nouveau module afin de s'assurer de toujours utiliser exactement le même code.
- > Vous pouvez donc créer votre propre module pour regrouper des fonctions sur un sujet.
- > Créer un nouveau module est aussi simple que d'écrire un script



# Modules

EXPLORATEUR ...

MODULES

- /\* petit\_script\_rapide.py
- /\* utilitaire\_admin\_CEM.py

/\* utilitaire\_admin\_CEM.py × ...

```
/* utilitaire_admin_CEM.py > ⚙ supprimer_utilisateurs
1 import subprocess
2 def ajout_utilisateur(nom="",password ""):
3     if nom == "":
4         nom = input("Entrez nom d'utilisateur : ")
5     if password == "":
6         password = input("Entrez mot de passe : ")
7     subprocess.run(["useradd","-p", nom, password])
8
9 def supprimer_utilisateurs(nom=""):
10    if nom == "":
11        nom = input("Entrez nom d'utilisateur : ")
```



# Importer le module et utiliser ses fonctions

```
/* petit_script_rapide.py */
/* petit_script_rapide.py > ...
1 #importe notre module fait maison
2 import utilitaire_admin_CEM
3 import csv
4 #ouvre un csv et exécute une fonction provenant du module
5 with open("liste_nouveaux_utilisateurs.csv", "r",) as fichier_users:
6     csv_read = csv.reader(fichier_users)
7     for ligne in fichier_users:
8         nom = ligne[0]
9         mot_de_passe = ligne[1]
10        utilitaire_admin_CEM.ajout_utilisateur(nom, mot_de_passe)
11    ...
```

Importer le module

Utiliser une de ses fonctions



# Tester les cas limites

- > Quand vous écrivez une fonction, vous devez tester les cas limites pour vérifier que tout est ok.
- > Les cas limites sont les valeurs qui pourraient causer des problèmes.
- > Par exemple, si votre fonction prend en paramètre un entier entre 1 et 10, les cas limites seraient:
  - > Passer 1 en paramètre
  - > Passer 10 en paramètre
  - > Passer -1 en paramètre
  - > Passer 11 en paramètre
  - > Ne rien passer en paramètre
  - > Passer 'patate' en paramètre

Nous allons voir plus de possibilités dans ces cas plus tard, en utilisant les try...except

# Tester les cas limites pour toutes les fonctions de vos modules



- > Quand vous écrivez un module, c'est très important de tester les cas limites des fonctions qui sont dans vos modules.
- > On s'attend à ce qu'un module ait été bien testé et qu'il soit vraiment fonctionnel.
- > Si une fonction pose problème, cette erreur existera dans tous les scripts utilisant ce module.



# Un fichier : module ET script

- > Lorsqu'on importe un module, l'interpréteur pas au travers de chacune des lignes du module.
- > Toutes les lignes seront exécutés.
- > Pour utiliser un même fichier en module et script, il faut inclure les lignes à exécuter en script doivent être incluses dans un bloc conditionnel :

```
if __name__ == "__main__":
    ...
    while True:
        var = input("Entrez le nombre : ")
        if var.isdigit():
            print(var)
        else:
            print("Veuillez entrer un nombre valide")
```