

## 2N6 Programmation 2



python<sup>TM</sup>



> 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 ?

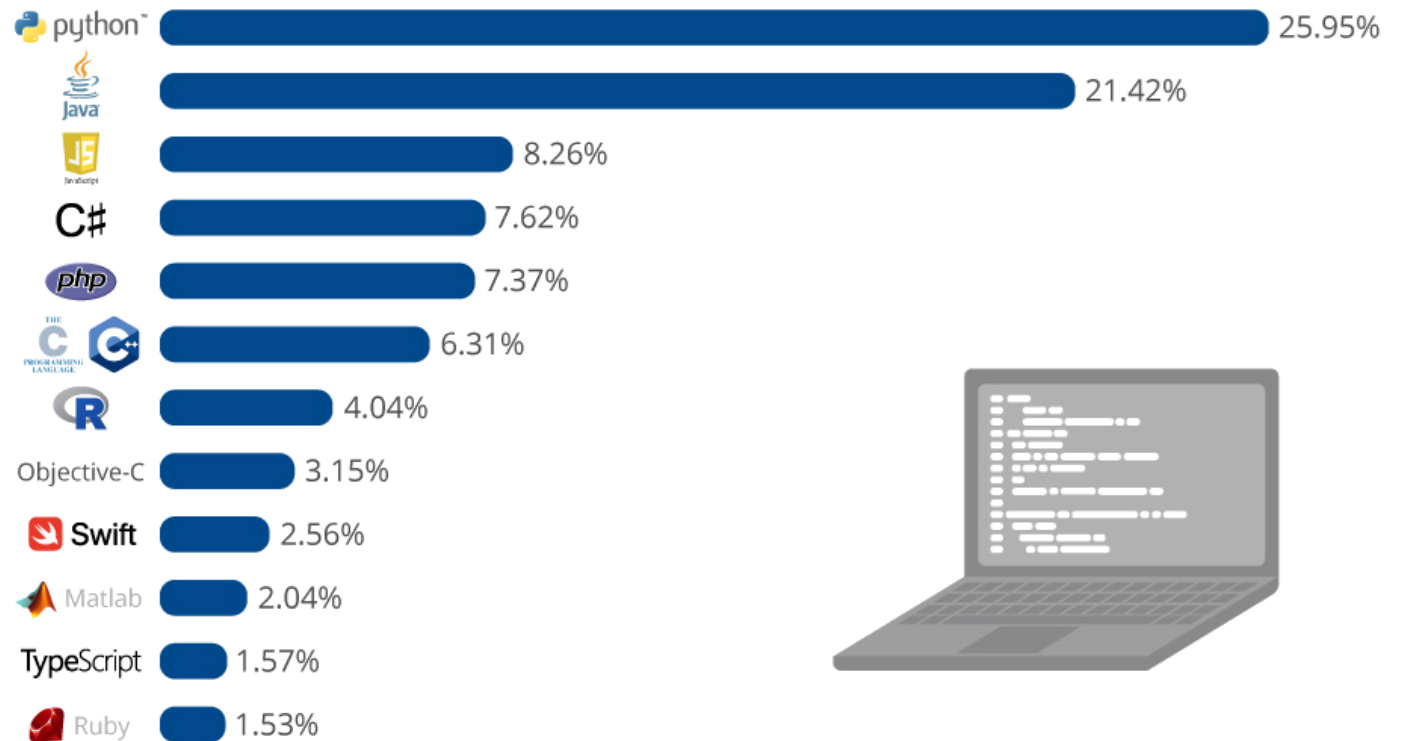
# Python



- > 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\*



@StatistaCharts

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

Source: PYPL

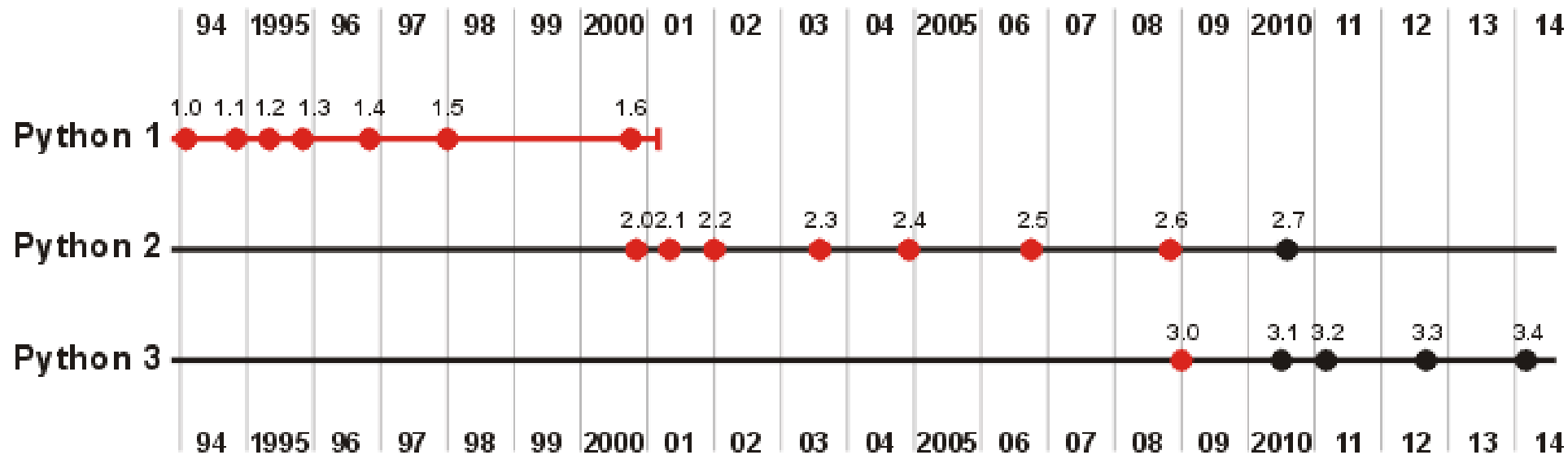
statista

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

# Historique



- > 1991 : Version 1.0, crée 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



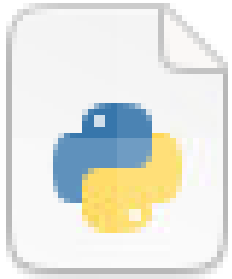
Nous utiliserons Python pour créer des scripts avancés :

- > 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 Fibonacci 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>
```

# 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__', '__getattr__', '__getitem__', '__getnewargs__', '
hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__',
__l od__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr
__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',
'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'f
ormat_map', 'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit',
er', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'j
st', 'lower', 'lstrip', 'maketrans', 'partition', 'removeprefix', 'removesuffix
e', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'spl
'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
>>>
>>> help(str.upper)
Help on method_descriptor:

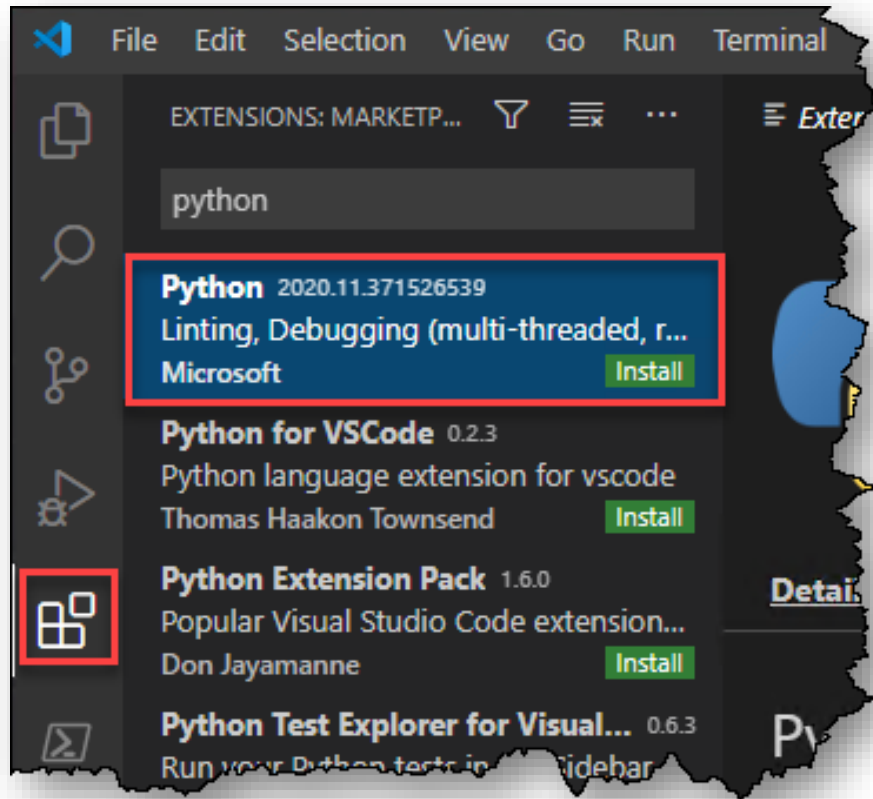
upper(self, /)
    Return a copy of the string converted to uppercase.

>>>
```



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

# Visual Studio Code (VSC)



IDE simple et léger

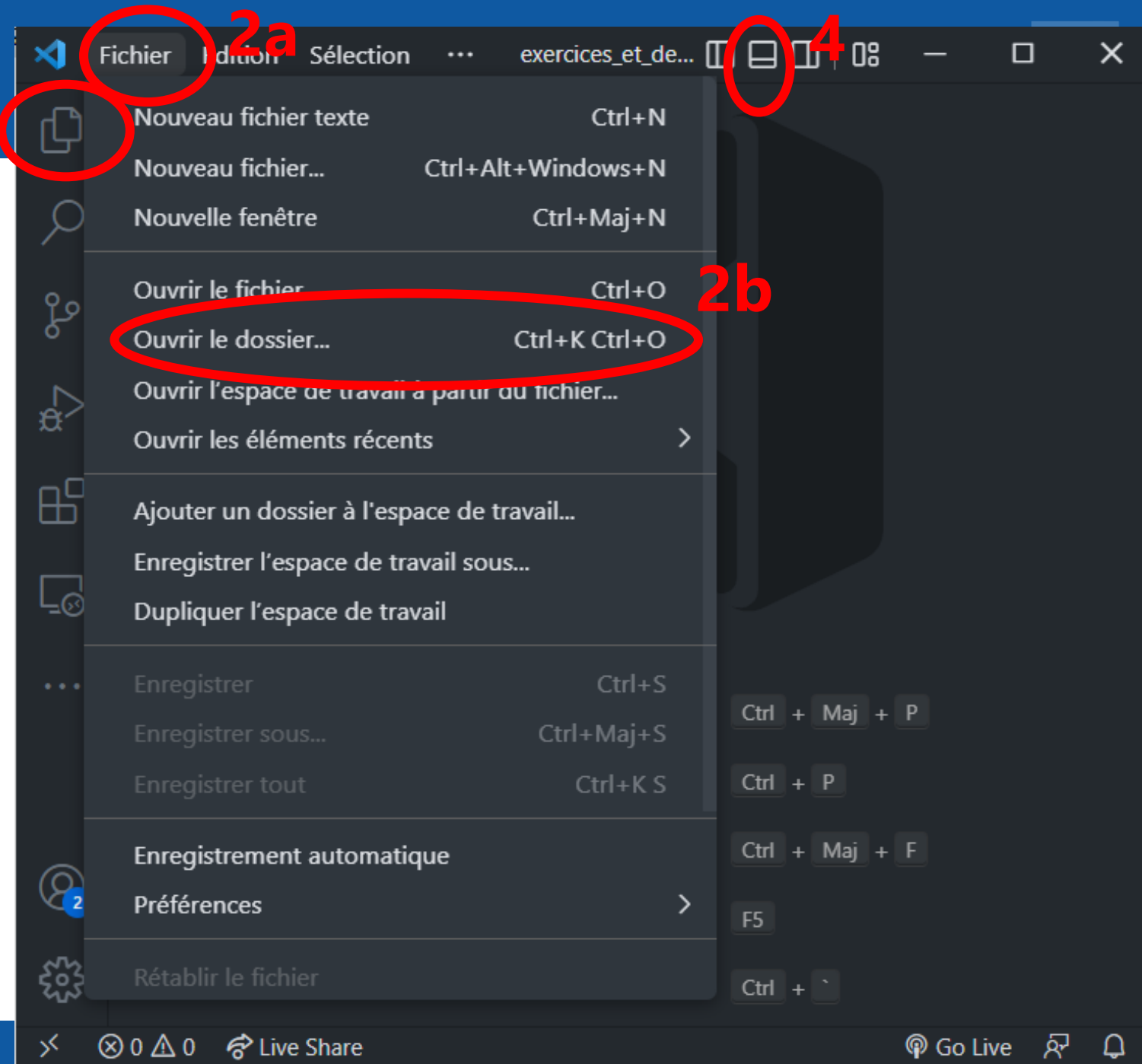
Utilisable sur tous les OS

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

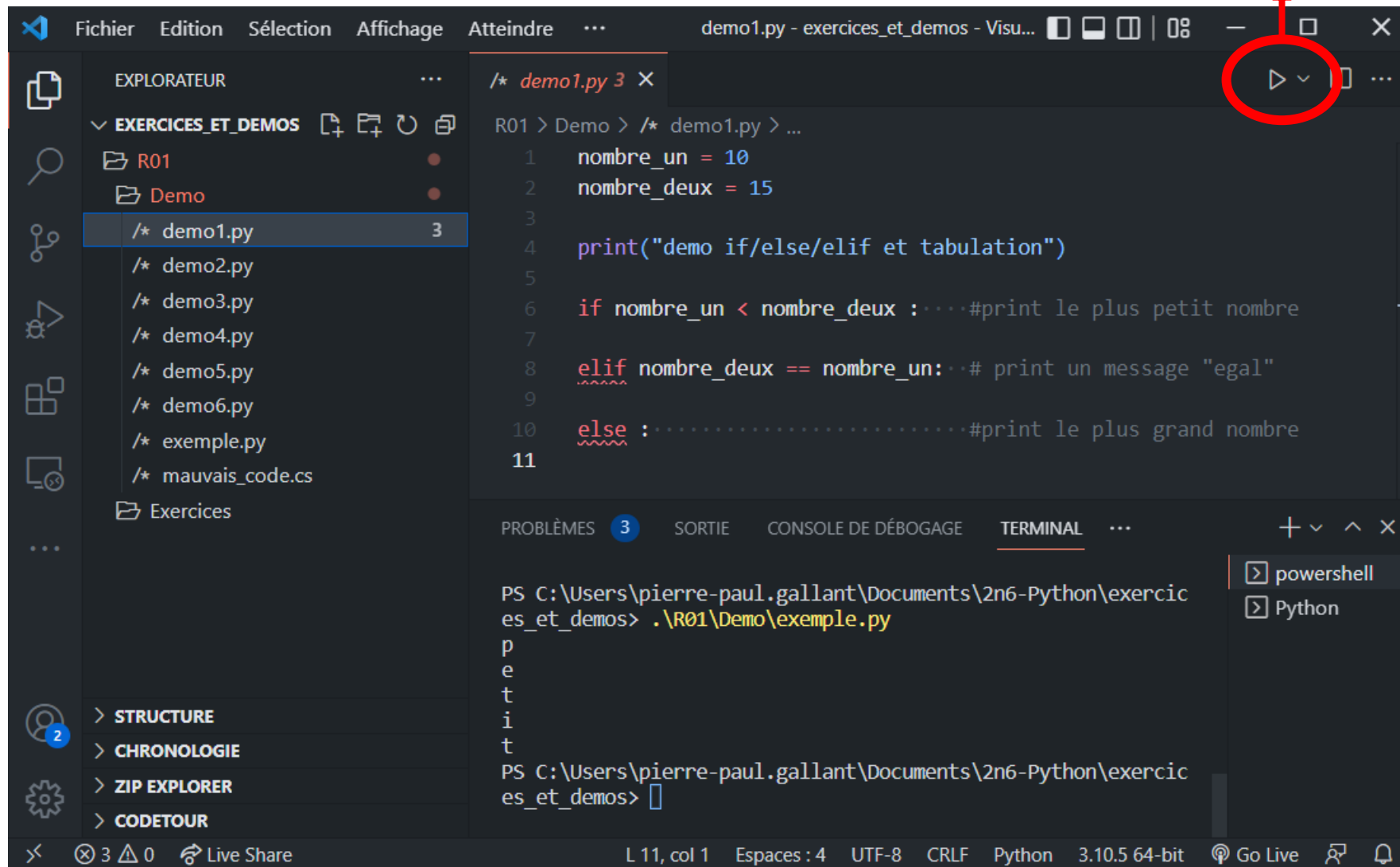


# Utiliser VSC

Exécution du code  
/ du débogueur



Explorateur  
de fichiers



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'>
>>>
```

# Typeage 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
```

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

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

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



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

➡ formatted\_strings.py > ...

```
1   prenom = "Pierre-Paul"
2   nom_famille = "Gallant"
3
4   print("Bonjour,", prenom, nom_famille, "!")
5   print("Bonjour, {0} {1} !".format(prenom, nom_famille))
6   print(f"Bonjour, {prenom} {nom_famille} !")
```

```
8   # Mais si on décide de changer le "!" pour un "."
9   print("Bonjour, ", prenom, " ", nom_famille, ".", sep=" ")
10  print("Bonjour, {0} {1}.".format(prenom, nom_famille))
11  print(f"Bonjour, {prenom} {nom_famille}.")
```

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

```
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 utilisé directement en mettant le nom de la variable entre { }

On peut y mettre du texte comme dans un string normal

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



# Opérateurs

# Opérateurs arithmétiques



Opérateurs	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

Opérateur	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**



# Opérateurs 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
```








# Structures de contrôle (exemple)

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

```
with open('contributeursNomPrenom.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 INTER/

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.ga  
p  
e  
t  
i  
t
```

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

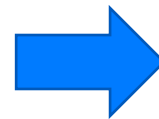
```
pierre-paul@pp-vm:~/scripts$ ./loop.py  
M  
o  
n  
t  
p  
e  
t  
i  
t
```



# Structures de contrôle (boucles)

- > Deux mots-clefs 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 !
```

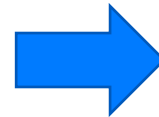




# Structures de contrôle (boucles)

- > Deux mots-clefs 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
e
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 retournée.

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  
!  
>>> █
```



# Zen of Python



- › **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!



# 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](#)