

IRM Licence Informatique Réseaux Multimédias

Apprendre à coder avec Python
Version 1.0

Abdellah Adib

Dept. Informatique, FST, Mohammedia

October 2, 2024

Sommaire

- ① Avant-propos
- ② Environnement Python
- ③ Bases d'utilisation
- ④ Outils de contrôle de flux
- ⑤ Les collections de données
- ⑥ Les fonctions en Python
- ⑦ Les modules/librairies en Python
- ⑧ Les fichiers entrées/sorties en Python

Table des matières

1 Avant-propos

Origine de Python?

Pourquoi Python ?

2 Environnement Python

3 Bases d'utilisation

4 Outils de contrôle de flux

5 Les collections de données

6 Les fonctions en Python

7 Les modules/librairies en Python

8 Les fichiers entrées/sorties en Python

Origine de Python?

- Le langage de programmation Python a été créé en 1989 par **Guido van Rossum**, aux Pays-Bas et de nombreux contributeurs bénévoles.



Python



Langage Python



Monty Python's

Origine de Python?

- Le langage de programmation Python a été créé en 1989 par **Guido van Rossum**, aux Pays-Bas et de nombreux contributeurs bénévoles.



Python



Langage Python



Monty Python's

- Date de première version 20 février 1991.
- Dernière version stable "3.12" 02 Octobre 2023.

Table des matières

1 Avant-propos

Origine de Python?

Pourquoi Python ?

2 Environnement Python

3 Bases d'utilisation

4 Outils de contrôle de flux

5 Les collections de données

6 Les fonctions en Python

7 Les modules/librairies en Python

8 Les fichiers entrées/sorties en Python

Pourquoi Python?

- ▶ **Python** est un langage portable, dynamique, extensible, gratuit, qui permet (sans l'imposer) une approche modulaire et orientée objet de la programmation.

Pourquoi Python?

- ▶ **Python** est un langage portable, dynamique, extensible, gratuit, qui permet (sans l'imposer) une approche modulaire et orientée objet de la programmation.
- ▶ **Python** présente de nombreuses caractéristiques intéressantes :

Pourquoi Python?

- ▶ **Python** est un langage portable, dynamique, extensible, gratuit, qui permet (sans l'imposer) une approche modulaire et orientée objet de la programmation.
- ▶ **Python** présente de nombreuses caractéristiques intéressantes :
 - * Il est multiplateforme/portable ; il fonctionne sur de nombreux systèmes d'exploitation : Windows, Mac OS X, Linux, Android, iOS, depuis les mini-ordinateurs Raspberry Pi jusqu'aux supercalculateurs.

Pourquoi Python?

- ▶ **Python** est un langage portable, dynamique, extensible, gratuit, qui permet (sans l'imposer) une approche modulaire et orientée objet de la programmation.
- ▶ **Python** présente de nombreuses caractéristiques intéressantes :
 - ✳ Il est multiplateforme/portable ; il fonctionne sur de nombreux systèmes d'exploitation : Windows, Mac OS X, Linux, Android, iOS, depuis les mini-ordinateurs Raspberry Pi jusqu'aux supercalculateurs.
 - ✳ C'est un langage interprété. Un script Python n'a pas besoin d'être compilé pour être exécuté, contrairement à des langages comme le C ou le C++.

Pourquoi Python?

- ▶ **Python** est un langage portable, dynamique, extensible, gratuit, qui permet (sans l'imposer) une approche modulaire et orientée objet de la programmation.
- ▶ **Python** présente de nombreuses caractéristiques intéressantes :
 - ✳ Il est multiplateforme/portable ; il fonctionne sur de nombreux systèmes d'exploitation : Windows, Mac OS X, Linux, Android, iOS, depuis les mini-ordinateurs Raspberry Pi jusqu'aux supercalculateurs.
 - ✳ C'est un langage interprété. Un script Python n'a pas besoin d'être compilé pour être exécuté, contrairement à des langages comme le C ou le C++.
 - ✳ Il est gratuit. Il peut être installé sur autant d'ordinateurs que vous voulez (même sur un smartphone !).

Pourquoi Python?

- ▶ **Python** est un langage portable, dynamique, extensible, gratuit, qui permet (sans l'imposer) une approche modulaire et orientée objet de la programmation.
- ▶ **Python** présente de nombreuses caractéristiques intéressantes :
 - ✳ Il est multiplateforme/portable ; il fonctionne sur de nombreux systèmes d'exploitation : Windows, Mac OS X, Linux, Android, iOS, depuis les mini-ordinateurs Raspberry Pi jusqu'aux supercalculateurs.
 - ✳ C'est un langage interprété. Un script Python n'a pas besoin d'être compilé pour être exécuté, contrairement à des langages comme le C ou le C++.
 - ✳ Il est gratuit. Il peut être installé sur autant d'ordinateurs que vous voulez (même sur un smartphone !).
 - ✳ Il possède une bibliothèque standard ainsi qu'un ensemble de paquetages contribués, donnant accès à une grande variété de services.

Applications de Python

Python est un langage de programmation très polyvalent, puissant et adapté à de nombreux environnements de développement.

- ▶ **Développement Web** en utilisant des frameworks comme (Django, Flask, et FastAPI).

Applications de Python

Python est un langage de programmation très polyvalent, puissant et adapté à de nombreux environnements de développement.

- ▶ **Développement Web** en utilisant des frameworks comme (Django, Flask, et FastAPI).
- ▶ **Data Science et analyse de données** en utilisant des bibliothèques (Pandas, NumPy, Matplotlib, et Seaborn).

Applications de Python

Python est un langage de programmation très polyvalent, puissant et adapté à de nombreux environnements de développement.

- ▶ **Développement Web** en utilisant des frameworks comme (Django, Flask, et FastAPI).
- ▶ **Data Science et analyse de données** en utilisant des bibliothèques (Pandas, NumPy, Matplotlib, et Seaborn).
- ▶ **Apprentissage automatique et Intelligence Artificielle** en utilisant des bibliothèques (TensorFlow, Keras, PyTorch et Scikit-learn).

Applications de Python

Python est un langage de programmation très polyvalent, puissant et adapté à de nombreux environnements de développement.

- ▶ **Développement Web** en utilisant des frameworks comme (Django, Flask, et FastAPI).
- ▶ **Data Science et analyse de données** en utilisant des bibliothèques (Pandas, NumPy, Matplotlib, et Seaborn).
- ▶ **Apprentissage automatique et Intelligence Artificielle** en utilisant des bibliothèques (TensorFlow, Keras, PyTorch et Scikit-learn).
- ▶ **Automatisation et scripts** en utilisant (BeautifulSoup, Scrapy) pour scraping web et (Selenium ou PyTest) pour l'automatisation des tests logiciels.

Applications de Python

Python est un langage de programmation très polyvalent, puissant et adapté à de nombreux environnements de développement.

- ▶ **Développement Web** en utilisant des frameworks comme (Django, Flask, et FastAPI).
- ▶ **Data Science et analyse de données** en utilisant des bibliothèques (Pandas, NumPy, Matplotlib, et Seaborn).
- ▶ **Apprentissage automatique et Intelligence Artificielle** en utilisant des bibliothèques (TensorFlow, Keras, PyTorch et Scikit-learn).
- ▶ **Automatisation et scripts** en utilisant (BeautifulSoup, Scrapy) pour scraping web et (Selenium ou PyTest) pour l'automatisation des tests logiciels.
- ▶ **Développement de jeux vidéo** en utilisant une bibliothèque (Pygame).

Applications de Python

- ▶ **Applications de bureau (GUI)** en utilisant des bibliothèques (Tkinter, PyQt ou Kivy).

Applications de Python

- ▶ **Applications de bureau (GUI)** en utilisant des bibliothèques (Tkinter, PyQt ou Kivy).
- ▶ **Cybersécurité** en utilisant des bibliothèques (Scapy).

Applications de Python

- ▶ **Applications de bureau (GUI)** en utilisant des bibliothèques (Tkinter, PyQt ou Kivy).
- ▶ **Cybersécurité** en utilisant des bibliothèques (Scapy).
- ▶ **Calcul scientifique et mathématique** en utilisant des bibliothèques (SciPy, SymPy).

Applications de Python

- ▶ **Applications de bureau (GUI)** en utilisant des bibliothèques (Tkinter, PyQt ou Kivy).
- ▶ **Cybersécurité** en utilisant des bibliothèques (Scapy).
- ▶ **Calcul scientifique et mathématique** en utilisant des bibliothèques (SciPy, SymPy).
- ▶ **Traitement d'images et de vidéos** en utilisant des bibliothèques (OpenCV, Pillow/Pil, Scikit-Image).

Applications de Python

- ▶ **Applications de bureau (GUI)** en utilisant des bibliothèques (Tkinter, PyQt ou Kivy).
- ▶ **Cybersécurité** en utilisant des bibliothèques (Scapy).
- ▶ **Calcul scientifique et mathématique** en utilisant des bibliothèques (SciPy, SymPy).
- ▶ **Traitement d'images et de vidéos** en utilisant des bibliothèques (OpenCV, Pillow/Pil, Scikit-Image).
- ▶ **Traitement des signaux audio** en utilisant des bibliothèques (Librosa, Soundfile, Pydub).

Applications de Python

- ▶ **Applications de bureau (GUI)** en utilisant des bibliothèques (Tkinter, PyQt ou Kivy).
- ▶ **Cybersécurité** en utilisant des bibliothèques (Scapy).
- ▶ **Calcul scientifique et mathématique** en utilisant des bibliothèques (SciPy, SymPy).
- ▶ **Traitement d'images et de vidéos** en utilisant des bibliothèques (OpenCV, Pillow/Pil, Scikit-Image).
- ▶ **Traitement des signaux audio** en utilisant des bibliothèques (Librosa, Soundfile, Pydub).
- ▶ **Internet des objets (IoT)** en utilisant des bibliothèques (MicroPython).

Table des matières

- ① Avant-propos
- ② Environnement Python
 - Installer Python
 - Calcul avec Python
 - Éditeur Python
- ③ Bases d'utilisation
- ④ Outils de contrôle de flux
- ⑤ Les collections de données
- ⑥ Les fonctions en Python
- ⑦ Les modules/librairies en Python
- ⑧ Les fichiers entrées/sorties en Python

Installer et configurer Python

- ▶ Une maîtrise de la programmation **Python** passe obligatoirement par une bonne pratique, d'où la nécessité d'installer **Python** sur votre ordinateur.

Installer et configurer Python

- ▶ Une maîtrise de la programmation **Python** passe obligatoirement par une bonne pratique, d'où la nécessité d'installer **Python** sur votre ordinateur.
- ▶ Vous pouvez télécharger et installer **Python** à partir de son site web officiel :
<https://www.python.org/downloads/>

Installer et configurer Python

- ▶ Une maîtrise de la programmation **Python** passe obligatoirement par une bonne pratique, d'où la nécessité d'installer **Python** sur votre ordinateur.
- ▶ Vous pouvez télécharger et installer **Python** à partir de son site web officiel :
<https://www.python.org/downloads/>

Installer et configurer Python

- ▶ Une maîtrise de la programmation **Python** passe obligatoirement par une bonne pratique, d'où la nécessité d'installer **Python** sur votre ordinateur.
- ▶ Vous pouvez télécharger et installer **Python** à partir de son site web officiel :
<https://www.python.org/downloads/>



The screenshot shows the Python download page for Windows. At the top, it says "Download the latest version for Windows". Below that is a button labeled "Download Python 3.12.6". Further down, there's text about other OS versions and links for "Prereleases" and "Docker images". To the right of the text, there's a graphic of two brown cardboard boxes with yellow straps, each attached to a large, multi-colored (yellow, white, grey) parachute, falling against a blue background with white clouds.

Download the latest version for Windows

Download Python 3.12.6

Looking for Python with a different OS? Python for [Windows](#), [Linux/UNIX](#), [macOS](#), [Other](#)

Want to help test development versions of Python 3.13? [Prereleases](#), [Docker images](#)

Installer et configurer Python

- ▶ Le processus d'installation est facile et la vidéo ci-dessous présente l'essentiel des différentes phases de cette opération :

Installer et configurer Python

- ▶ Le processus d'installation est facile et la vidéo ci-dessous présente l'essentiel des différentes phases de cette opération :

Installer et configurer Python

- ▶ Le processus d'installation est facile et la vidéo ci-dessous présente l'essentiel des différentes phases de cette opération :

https://www.youtube.com/watch?v=IS117_uXZKE

- ▶ **Python** présente la particularité de pouvoir être utilisé de plusieurs manières différentes.

Installer et configurer Python

- ▶ Le processus d'installation est facile et la vidéo ci-dessous présente l'essentiel des différentes phases de cette opération :

https://www.youtube.com/watch?v=IS117_uXZKE

- ▶ **Python** présente la particularité de pouvoir être utilisé de plusieurs manières différentes.
- ▶ **Python** est un langage interprété qui va être lu ligne par ligne par un interpréteur.

Installer et configurer Python

- ▶ Le processus d'installation est facile et la vidéo ci-dessous présente l'essentiel des différentes phases de cette opération :

https://www.youtube.com/watch?v=IS117_uXZKE

- ▶ **Python** présente la particularité de pouvoir être utilisé de plusieurs manières différentes.
- ▶ **Python** est un langage interprété qui va être lu ligne par ligne par un interpréteur.
- ▶ L'interpréteur va passer à travers chaque ligne de votre code **Python** pour le traduire en langage machine pour qu'il soit exécuter par votre ordinateur.

Environnement de développement

- ▶ L'interpréteur **Python** présente la particularité de pouvoir être utilisé de deux manières différentes :

- ▶ L'interpréteur **Python** présente la particularité de pouvoir être utilisé de deux manières différentes :
 - * Il peut être utilisé en **mode interactif**, c'est-à-dire de manière à dialoguer avec lui directement depuis le clavier. L'interprète exécute une ligne de code et attend que vous en tapiez une autre.

Environnement de développement

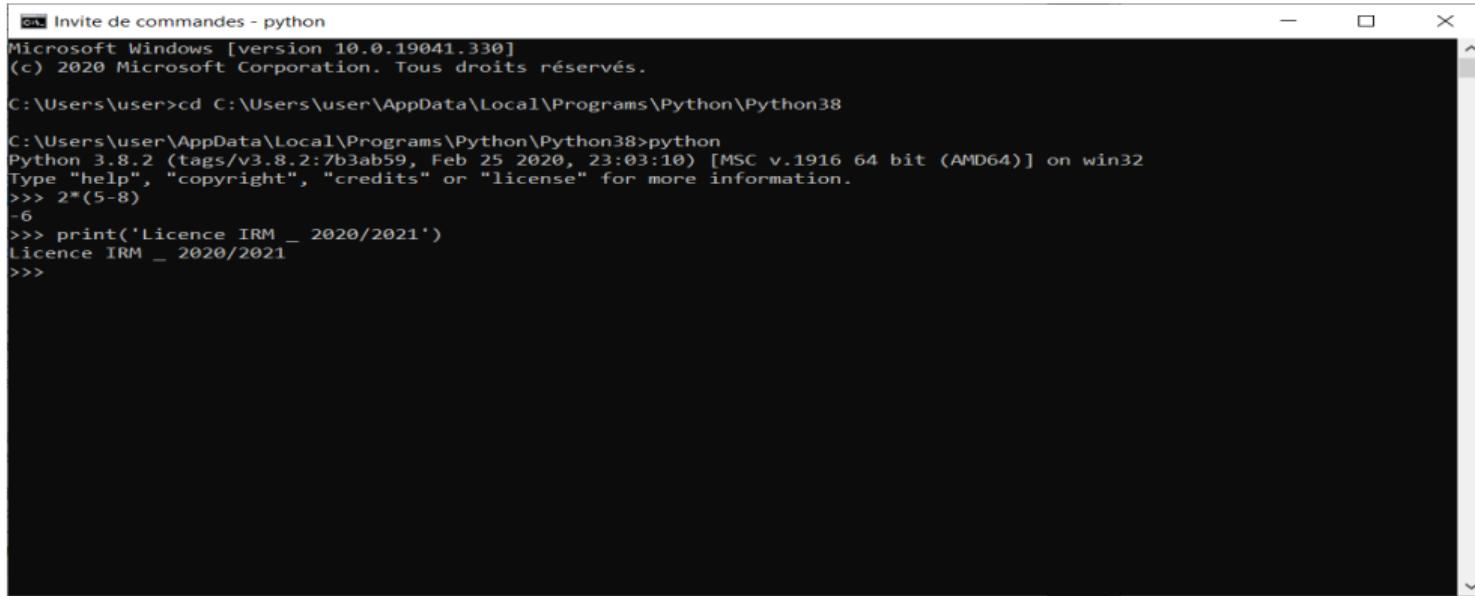
- ▶ L'interpréteur **Python** présente la particularité de pouvoir être utilisé de deux manières différentes :
 - * Il peut être utilisé en **mode interactif**, c'est-à-dire de manière à dialoguer avec lui directement depuis le clavier. L'interprète exécute une ligne de code et attend que vous en tapiez une autre.
 - * En **mode scripts**, l'interpréteur lit le contenu d'un fichier qui contient les instructions **Python**. Un tel fichier est connu sous le nom de programme ou script **Python**. Il exécute chaque instruction dans le programme **Python** au fur et à mesure qu'il la lit.

Table des matières

- 1 Avant-propos
- 2 Environnement Python
 - Installer Python
 - Calcul avec Python
 - Éditeur Python
- 3 Bases d'utilisation
- 4 Outils de contrôle de flux
- 5 Les collections de données
- 6 Les fonctions en Python
- 7 Les modules/librairies en Python
- 8 Les fichiers entrées/sorties en Python

Calculer avec Python

- ▶ L'interpréteur peut être lancé directement depuis la ligne de commande (dans une fenêtre DOS sous Windows) : il suffit d'y taper la commande **Python3**.



The screenshot shows a Microsoft Windows Command Prompt window titled "Invite de commandes - python". The window displays the following text:

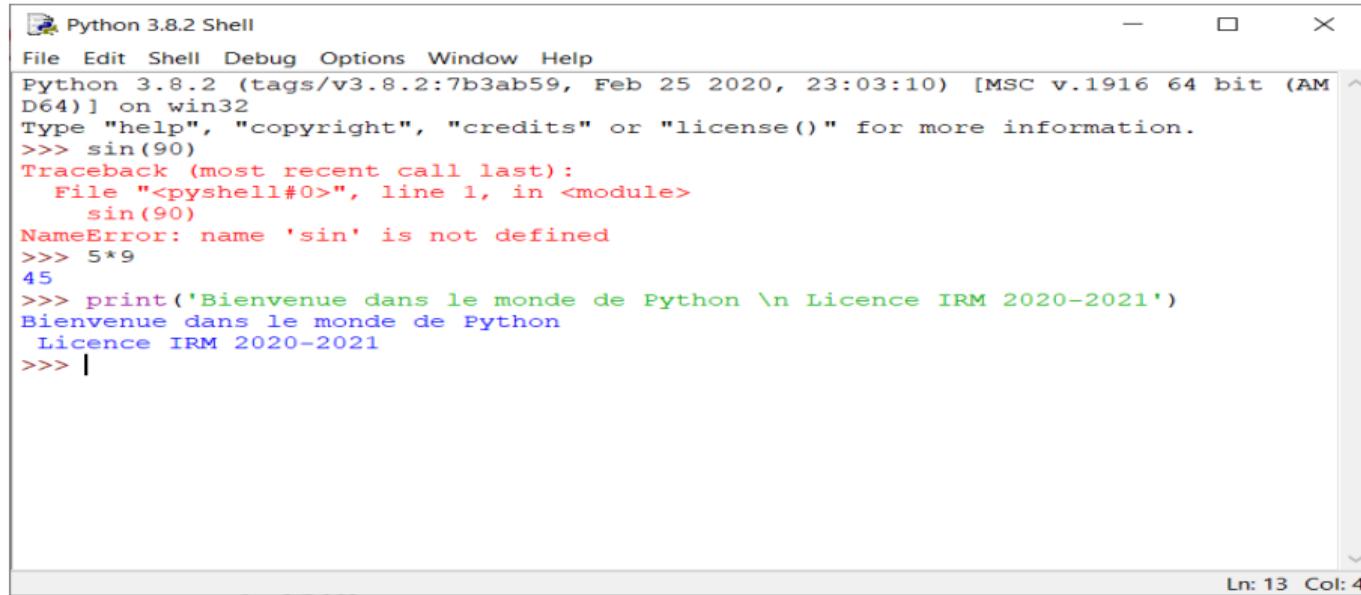
```
Microsoft Windows [version 10.0.19041.3301]
(c) 2020 Microsoft Corporation. Tous droits réservés.

C:\Users\user>cd C:\Users\user\AppData\Local\Programs\Python\Python38

C:\Users\user\AppData\Local\Programs\Python\Python38>python
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 23:03:10) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 2*(5-8)
-6
>>> print('Licence IRM _ 2020/2021')
Licence IRM _ 2020/2021
>>>
```

Calculer avec Python

- ▶ Dans le menu Démarrer, choisissez "IDLE" (**Python** GUI) dans le groupe de programmes **Python** pour lancer le shell :



```
Python 3.8.2 Shell
File Edit Shell Debug Options Window Help
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 23:03:10) [MSC v.1916 64 bit (AM
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> sin(90)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    sin(90)
NameError: name 'sin' is not defined
>>> 5*9
45
>>> print('Bienvenue dans le monde de Python \n Licence IRM 2020-2021')
Bienvenue dans le monde de Python
  Licence IRM 2020-2021
>>> |
```

Ln: 13 Col: 4

Calculer avec Python

- ▶ Le mode interactif permet d'apprendre très rapidement les bases du langage, par expérimentation directe.

Calculer avec Python

- ▶ Le mode interactif permet d'apprendre très rapidement les bases du langage, par expérimentation directe.
- ▶ Cette façon de faire présente toutefois un gros inconvénient :

Calculer avec Python

- ▶ Le mode interactif permet d'apprendre très rapidement les bases du langage, par expérimentation directe.
- ▶ Cette façon de faire présente toutefois un gros inconvénient :

- ▶ Le mode interactif permet d'apprendre très rapidement les bases du langage, par expérimentation directe.
- ▶ Cette façon de faire présente toutefois un gros inconvénient :

Remarques

Toutes les séquences d'instructions écrites disparaissent définitivement après fermeture de l'interpréteur.

- ▶ Le mode interactif permet d'apprendre très rapidement les bases du langage, par expérimentation directe.
- ▶ Cette façon de faire présente toutefois un gros inconvénient :

Remarques

Toutes les séquences d'instructions écrites disparaissent définitivement après fermeture de l'interpréteur.

- ▶ Il convient donc de sauvegarder les programmes dans des fichiers de manière à pouvoir les retravailler par étapes successives.

- ▶ Le mode interactif permet d'apprendre très rapidement les bases du langage, par expérimentation directe.
- ▶ Cette façon de faire présente toutefois un gros inconvénient :

Remarques

Toutes les séquences d'instructions écrites disparaissent définitivement après fermeture de l'interpréteur.

- ▶ Il convient donc de sauvegarder les programmes dans des fichiers de manière à pouvoir les retravailler par étapes successives.
- ▶ Par conséquent, il vaut mieux en utiliser des outils spécialisés en programmation, appelés Environnement de Développement Intégré (EDI).

Table des matières

- ① Avant-propos
- ② Environnement Python
 - Installer Python
 - Calcul avec Python
 - Éditeur Python
- ③ Bases d'utilisation
- ④ Outils de contrôle de flux
- ⑤ Les collections de données
- ⑥ Les fonctions en Python
- ⑦ Les modules/librairies en Python
- ⑧ Les fichiers entrées/sorties en Python

Choisir un bon éditeur de texte

- ▶ Le choix de l'éditeur est très fondamental, du fait qu'il doit aider à repérer rapidement certaines zones du programme, à le relire rapidement et à éviter les fautes de syntaxe.

Choisir un bon éditeur de texte

- ▶ Le choix de l'éditeur est très fondamental, du fait qu'il doit aider à repérer rapidement certaines zones du programme, à le relire rapidement et à éviter les fautes de syntaxe.
- ▶ En plus des fonctions de manipulation / remplacement / recherche de texte, un bon éditeur doit absolument posséder :

Choisir un bon éditeur de texte

- ▶ Le choix de l'éditeur est très fondamental, du fait qu'il doit aider à repérer rapidement certaines zones du programme, à le relire rapidement et à éviter les fautes de syntaxe.
- ▶ En plus des fonctions de manipulation / remplacement / recherche de texte, un bon éditeur doit absolument posséder :
 - ✳ La coloration syntaxique (syntax highlighting en anglais). Celle-ci change la couleur et / ou la police de certaines zones du code comme les mot-clés du langage, les zones entre guillemets, les commentaires, etc.

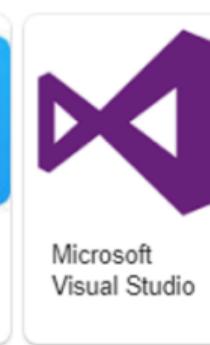
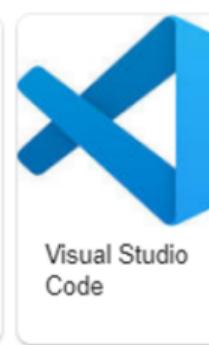
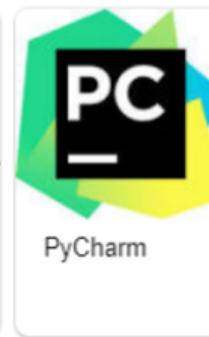
Choisir un bon éditeur de texte

- ▶ Le choix de l'éditeur est très fondamental, du fait qu'il doit aider à repérer rapidement certaines zones du programme, à le relire rapidement et à éviter les fautes de syntaxe.
- ▶ En plus des fonctions de manipulation / remplacement / recherche de texte, un bon éditeur doit absolument posséder :
 - ✳ La coloration syntaxique (syntax highlighting en anglais). Celle-ci change la couleur et / ou la police de certaines zones du code comme les mot-clés du langage, les zones entre guillemets, les commentaires, etc.
 - ✳ La complétion (*Complètement de la saisie au clavier qui se fait automatiquement*) automatique aide à taper le code source plus rapidement et en évitant les fautes de frappe.

Choisir un bon éditeur de texte

- ▶ Le choix de l'éditeur est très fondamental, du fait qu'il doit aider à repérer rapidement certaines zones du programme, à le relire rapidement et à éviter les fautes de syntaxe.
- ▶ En plus des fonctions de manipulation / remplacement / recherche de texte, un bon éditeur doit absolument posséder :
 - ✳ La coloration syntaxique (syntax highlighting en anglais). Celle-ci change la couleur et / ou la police de certaines zones du code comme les mot-clés du langage, les zones entre guillemets, les commentaires, etc.
 - ✳ La complétion (*Complètement de la saisie au clavier qui se fait automatiquement*) automatique aide à taper le code source plus rapidement et en évitant les fautes de frappe.
 - ✳ La mise en forme automatique qui permettra d'organiser le code en passant à la ligne à la fin d'une déclaration ou d'un " ; " par exemple, indentation automatique, etc.

Autres IDE



Autres IDE



Sublime Text



- ▶ **Sublime Text** est un éditeur de texte et de code source, disponible sur de multiples plates-formes ; Windows, Mac et Linux et dispose d'une API utilisant **Python**.



- ▶ **Sublime Text** est un éditeur de texte et de code source, disponible sur de multiples plates-formes ; Windows, Mac et Linux et dispose d'une API utilisant **Python**.
- ▶ **Sublime Text** propose un éditeur de texte se démarquant par son interface et ses fonctionnalités. Il supporte la coloration personnalisable selon les langages de programmation utilisés ainsi que, l'auto complétion, ...



- ▶ **Sublime Text** est un éditeur de texte et de code source, disponible sur de multiples plates-formes ; Windows, Mac et Linux et dispose d'une API utilisant **Python**.
- ▶ **Sublime Text** propose un éditeur de texte se démarquant par son interface et ses fonctionnalités. Il supporte la coloration personnalisable selon les langages de programmation utilisés ainsi que, l'auto complétion, ...
- ▶ **Sublime Text** dispose d'une interface pratique qui comprend un panel avec l'arborescence des dossiers des différentes sources éditées. Ensuite, on retrouve la gestion d'onglets pour un accès rapide aux fichiers en cours d'édition.



- ▶ **Sublime Text** est un éditeur de texte et de code source, disponible sur de multiples plates-formes ; Windows, Mac et Linux et dispose d'une API utilisant **Python**.
- ▶ **Sublime Text** propose un éditeur de texte se démarquant par son interface et ses fonctionnalités. Il supporte la coloration personnalisable selon les langages de programmation utilisés ainsi que, l'auto complétion, ...
- ▶ **Sublime Text** dispose d'une interface pratique qui comprend un panel avec l'arborescence des dossiers des différentes sources éditées. Ensuite, on retrouve la gestion d'onglets pour un accès rapide aux fichiers en cours d'édition.
- ▶ **Sublime Text** offre la modification de variables instantanées ou encore l'affichage en miniature du code sur un volet à droite du texte édité.



-/Documents/sage/src/sage/probability/random_variable.py (sage) - Sublime Text

FOLDERS

- sage
- build
- config
- local
 - bin
 - etc
- gap
- gap-4.8.3
 - bin
 - cnf
 - doc
 - etc
 - extern
 - grp
 - lib
 - pkg
 - src
 - tst
- CITATION
- CONTRIBUTING.md
- INSTALL.md
- LICENSE
- Makefile
- Makefile-default64
- Makefile.in
- README.md
- config.log
- config.status
- configure

random_variable.py

```
43 # We could inherit from a functions class here but use ParentWithBase
44
45 class RandomVariable_generic(ParentWithBase):
46     """
47     A random variable.
48     """
49     def __init__(self, X, RR):
50         if not is_ProbabilitySpace(X):
51             raise TypeError("Argument X (= %s) must be a probability space" % X)
52         ParentWithBase.__init__(self, X)
53         self._codomain = RR
54
55     def probability_space(self):
56         return self.base()
57
58     def domain(self):
59         return self.base()
60
61     def codomain(self):
62         return self._codomain
63
64     def field(self):
65         return self._codomain
66
67 class DiscreteRandomVariable(RandomVariable_generic):
68     """
69     A random variable on a discrete probability space.
70     """
71     def __init__(self, X, f, codomain = None, check = False):
72         """
73             Create free binary string monoid on 'n' generators.
74         
```

mode-fixes, Line 1, Column 1

Spaces: 4 Python





▶ Pour télécharger **Sublime Text**, il faut aller sur le site officiel :

<https://www.sublimetext.com/download>



- ▶ Pour télécharger **Sublime Text**, il faut aller sur le site officiel :

<https://www.sublimetext.com/download>

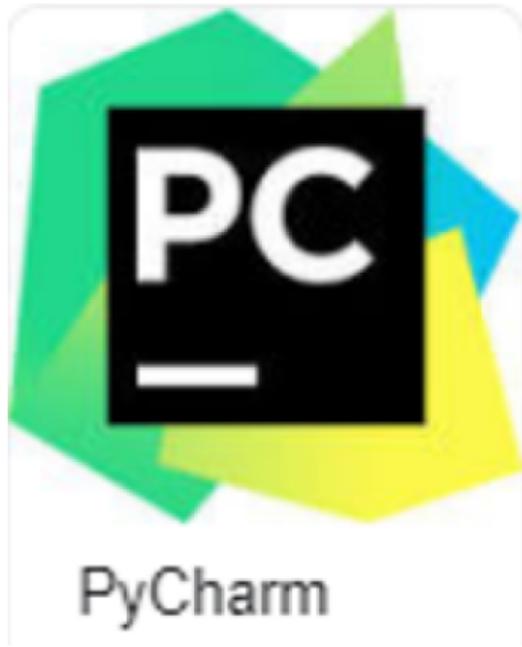
- ▶ Tout le détail sur la procédure d'installation de **Sublime Text** se trouve sur la page :

<https://docs.sublimetext.io/guide/getting-started/installation.html>



- ▶ Pour télécharger **Sublime Text**, il faut aller sur le site officiel :
<https://www.sublimetext.com/download>
- ▶ Tout le détail sur la procédure d'installation de **Sublime Text** se trouve sur la page :
<https://docs.sublimetext.io/guide/getting-started/installation.html>
- ▶ Toute la documentation sur l'éditeur **Sublime Text** est disponible sur le site non officiel suivant :
<https://sublime-text-unofficial-documentation.readthedocs.io/en/latest/intro.html>

Autres IDE



PyCharm



- ▶ **PyCharm** est un environnement de programmation léger et open source, spécialisé pour les langages de programmation **Python** et **Django**.



- ▶ **PyCharm** est un environnement de programmation léger et open source, spécialisé pour les langages de programmation **Python** et **Django**.
- ▶ **PyCharm** existe en deux versions disponibles en téléchargement, une version libre et gratuite appelée Community Edition et une version complète payante Professional Edition.



- ▶ **PyCharm** est un environnement de programmation léger et open source, spécialisé pour les langages de programmation **Python** et **Django**.
- ▶ **PyCharm** existe en deux versions disponibles en téléchargement, une version libre et gratuite appelée Community Edition et une version complète payante Professional Edition.
- ▶ L'interface de **PyCharm** permet la visualisation de l'ensemble des projets dans le volet de gauche, et l'éditeur de code dans le volet de droite.



- ▶ **PyCharm** est un environnement de programmation léger et open source, spécialisé pour les langages de programmation **Python** et **Django**.
- ▶ **PyCharm** existe en deux versions disponibles en téléchargement, une version libre et gratuite appelée Community Edition et une version complète payante Professional Edition.
- ▶ L'interface de **PyCharm** permet la visualisation de l'ensemble des projets dans le volet de gauche, et l'éditeur de code dans le volet de droite.
- ▶ L'éditeur de **PyCharm** fait état d'une foule de fonctionnalités d'édition telles que la vérification de bugs à la volée, la correction rapide des erreurs, la coloration syntaxique et la complétion automatique du code.



PyCharm

django_tutorial - models.py

```
import datetime
from django.db import models
from django.utils import timezone

class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')

    def was_published_recently(self):
        now = timezone.now()
        return now - timedelta(days=1) <= self.pub_date <= now

    class Choice(models.Model):
        question = models.ForeignKey(Question, on_delete=models.CASCADE)
        choice_text = models.CharField(max_length=200)
        votes = models.IntegerField(default=0)

        def __str__(self):
            return self.choice_text
```

Usages of Question in Project Files Found 15 usages

- admin.py: from .models import Question, Choice
- admin.py: admin.site.register(Question, QuestionAdmin)
- models.py: question = models.ForeignKey(Question, on_delete=models.CASCADE)
- tests.py: from .models import Question
- tests.py: return Question.objects.create(question_text=question_text, pub_date=time)
- tests.py: future_question = Question(pub_date=time)
- tests.py: old_question = Question(pub_date=time)
- tests.py: recent_question = Question(pub_date=time)
- views.py: from .models import Question, Choice
- views.py: model = Question
- views.py: return Question.objects.filter(model=Question)
- views.py: model = Question
- views.py: return Question.objects.filter(pub_date__lte=timezone.now())
- views.py: question = get_object_or_404(Question, pk=question_id)

Django Console

- Python Console
- Django 3.0.6
- >>>

Special Variables

7:12 LF UTF-8 4 spaces Python 3.8 (django_tutorial) master Event Log

Tests passed: 10 (33 minutes ago)



▶ Pour télécharger **PyCharm**, il faut aller sur le site officiel :

<https://www.jetbrains.com/pycharm/download>



- ▶ Pour télécharger **PyCharm**, il faut aller sur le site officiel :

<https://www.jetbrains.com/pycharm/download>

- ▶ Pour plus de détails sur les différentes étapes de l'installation de l'environnement **PyCharm**, le site suivant offre les différentes étapes :

<https://www.jetbrains.com/help/pycharm/installation-guide.html>



- ▶ Pour télécharger **PyCharm**, il faut aller sur le site officiel :

<https://www.jetbrains.com/pycharm/download>

- ▶ Pour plus de détails sur les différentes étapes de l'installation de l'environnement **PyCharm**, le site suivant offre les différentes étapes :

<https://www.jetbrains.com/help/pycharm/installation-guide.html>

- ▶ Pour une meilleure utilisation de **PyCharm**, une présentation détaillée de ses différentes fonctionnalités est disponible sur :

https://doplab.unil.ch/wp-content/uploads/2018/09/Pycharm_Tutorial-v1.pdf

Autres IDE



- ▶ L'installation d'un environnement **Python** complet peut-être une vraie galère. Déjà, il faut télécharger **Python** et l'installer. Par la suite, télécharger un à un les packages dont on a besoin. Parfois, le nombre de ces librairies peut-être grand.

- ▶ L'installation d'un environnement **Python** complet peut-être une vraie galère. Déjà, il faut télécharger **Python** et l'installer. Par la suite, télécharger un à un les packages dont on a besoin. Parfois, le nombre de ces librairies peut-être grand.

Remarques

Pour débuter, et avoir un environnement Python complet et prêt à l'emploi, l'idéal est d'installer la distribution Anaconda.

- ▶ L'installation d'un environnement **Python** complet peut-être une vraie galère. Déjà, il faut télécharger **Python** et l'installer. Par la suite, télécharger un à un les packages dont on a besoin. Parfois, le nombre de ces librairies peut-être grand.

Remarques

Pour débuter, et avoir un environnement Python complet et prêt à l'emploi, l'idéal est d'installer la distribution Anaconda.

- ▶ **Anaconda** est une distribution **Python** libre qui englobe directement une multitude de packages.

- ▶ L'installation d'un environnement **Python** complet peut-être une vraie galère. Déjà, il faut télécharger **Python** et l'installer. Par la suite, télécharger un à un les packages dont on a besoin. Parfois, le nombre de ces librairies peut-être grand.

Remarques

Pour débuter, et avoir un environnement Python complet et prêt à l'emploi, l'idéal est d'installer la distribution Anaconda.

- ▶ **Anaconda** est une distribution **Python** libre qui englobe directement une multitude de packages.
- ▶ **Anaconda** dispose d'un outil de gestion de packages appelé **Conda** qui permet d'installer facilement les librairies pour vos développements, ainsi que de les mettre à jour.

- ▶ La distribution **Anaconda** est présentée sur son site officiel, et ce pour les plateformes Linux, Windows, et Mac OS X. Pour son téléchargement, il faut aller sur le site officiel :

<https://www.anaconda.com/products/individual>

- ▶ La distribution **Anaconda** est présentée sur son site officiel, et ce pour les plateformes Linux, Windows, et Mac OS X. Pour son téléchargement, il faut aller sur le site officiel :

<https://www.anaconda.com/products/individual>

- ▶ Afin d'installer l'environnement de développement **Python** avec **Conda**, ainsi que d'importer la plupart des modules qui peuvent servir lors de vos opérations de codage, on peut se référer au site suivant pour plus d'informations :

*[https://www.editions-
eni.fr/open/mediabook.aspx?idR=797d189f805391b60342dca6f13e9cf](https://www.editions-eni.fr/open/mediabook.aspx?idR=797d189f805391b60342dca6f13e9cf)*

- ▶ La distribution **Anaconda** est présentée sur son site officiel, et ce pour les plateformes Linux, Windows, et Mac OS X. Pour son téléchargement, il faut aller sur le site officiel :

<https://www.anaconda.com/products/individual>

- ▶ Afin d'installer l'environnement de développement **Python** avec **Conda**, ainsi que d'importer la plupart des modules qui peuvent servir lors de vos opérations de codage, on peut se référer au site suivant pour plus d'informations :

[https://www.editions-
eni.fr/open/mediabook.aspx?idR=797d189f805391b60342dca6f13e9cf](https://www.editions-eni.fr/open/mediabook.aspx?idR=797d189f805391b60342dca6f13e9cf)

- ▶ Les différentes phases de l'installation de **Anaconda** sont résumées dans la vidéo suivante :

<https://www.youtube.com/watch?v=6YBsHr10qjc>



Notebook

6.4.12

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.



PyCharm Professional

2022.3.2

A full-fledged IDE by JetBrains for both Scientific and Web Python development. Supports HTML, JS, and SQL.



Spyder

5.2.2

Scientific Python Development Environment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features



VS Code

1.78.2

Streamlined code editor with support for development operations like debugging, task running and version control.



Anaconda Navigator

File Help

ANACONDA.NAVIGATOR

Home Environments Learning Community

Search Environments Q

Installed Channel Update Index Search Packages Q

Name	Description	Version
_ipyw_jlab_nb_ext...		0.1.0
alabaster		0.7.12
anaconda		2020.11
anaconda-client		1.7.2
anaconda-project		0.8.4
argh		0.26.2
argon2-cffi		20.1.0
asn1crypto		1.4.0
astroid		2.4.2
astropy		4.0.2
async-generator		1.10
async_generator		1.10
atomicwrites		1.4.0
attrr		20.3.0
autouseB		1.5.4
babel		2.8.1
beckcell		0.2.0
beckports		1.0
beckports.functools-truth-cache		1.6.1
beckports.functool...		1.6.1
beckports.shutil-glob-terminal-size		1.0.0
bekmunks-shell-in		1.0.0

127 packages available

Create Clone Import Remove

ANACONDA NUCLEUS Join Now

Discover premium data science content!

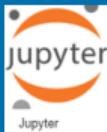
Documentation

Anaconda Blog

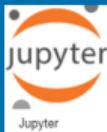
Twitter YouTube GitHub

Autres IDE





- ▶ Les **Jupyter Notebooks** sont des cahiers électroniques permettant de créer et de modifier des documents qui affichent les entrées et les sorties d'un script en langage **Python**. Une fois enregistrés, vous pouvez partager ces fichiers avec d'autres personnes.



- ▶ Les **Jupyter Notebooks** sont des cahiers électroniques permettant de créer et de modifier des documents qui affichent les entrées et les sorties d'un script en langage **Python**. Une fois enregistrés, vous pouvez partager ces fichiers avec d'autres personnes.
- ▶ Le même document **Jupyter Notebook** peut rassembler du texte, des images, des formules mathématiques et du code informatique exécutable. Il est manipulable interactivement dans un navigateur web.



- ▶ Les **Jupyter Notebooks** sont des cahiers électroniques permettant de créer et de modifier des documents qui affichent les entrées et les sorties d'un script en langage **Python**. Une fois enregistrés, vous pouvez partager ces fichiers avec d'autres personnes.
- ▶ Le même document **Jupyter Notebook** peut rassembler du texte, des images, des formules mathématiques et du code informatique exécutable. Il est manipulable interactivement dans un navigateur web.
- ▶ **Jupyter Notebook** était créée initialement pour les langages de programmation **Julia**, **Python** et **R** (Jupyter), actuellement il supporte près de 40 langages différents. L'extension d'un fichier **Jupyter Notebook** est **.ipynb** pour **I PYthon NoteBook**.



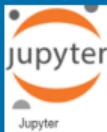
- ▶ Un document **Jupyter Notebook** permet d'intégrer en plus du code, des équations écrites en LATEX, du texte formaté en Markdown (convertit du format txt en html), différents médias (audio, vidéo) et qui s'exporte dans plusieurs formats (HTML, PDF, LATEX...).



- ▶ Un document **Jupyter Notebook** permet d'intégrer en plus du code, des équations écrites en LATEX, du texte formaté en Markdown (convertit du format txt en html), différents médias (audio, vidéo) et qui s'exporte dans plusieurs formats (HTML, PDF, LATEX...).
- ▶ La cellule est l'élément de base d'un **Jupyter Notebook**. Elle peut contenir du texte formaté au format Markdown ou du code informatique qui pourra être exécuté.



- ▶ Un document **Jupyter Notebook** permet d'intégrer en plus du code, des équations écrites en LATEX, du texte formaté en Markdown (convertit du format txt en html), différents médias (audio, vidéo) et qui s'exporte dans plusieurs formats (HTML, PDF, LATEX...).
- ▶ La cellule est l'élément de base d'un **Jupyter Notebook**. Elle peut contenir du texte formaté au format Markdown ou du code informatique qui pourra être exécuté.
- ▶ Pour lancer **Jupyter Notebook**, on fait recours au navigateur d'anaconda et on clic sur l'icône relative à **Jupyter Notebook**, ou on tape sur le Shell d'anaconda jupyter notebook.



▶ Jupyter Notebook Dashboard :

A screenshot of a web browser displaying the Jupyter Notebook dashboard. The address bar shows "try.jupyter.org". The page title is "jupyter". The top navigation bar includes icons for back, forward, and search, along with a "try.jupyter.org" button and a "Hosted by Rackspace" logo. Below the title, there are three tabs: "Files" (selected), "Running", and "Clusters". A sub-header says "Select items to perform actions on them." On the left, there's a sidebar with a file browser showing "communities", "datasets", and "featured" sections, each with a list of notebooks. On the right, there's a large, empty workspace area with "Upload", "New", and "New from template" buttons.



▶ Jupyter Notebook Editor :

The screenshot shows a web browser window for the Jupyter Notebook. The title bar reads "try.jupyter.org" and "jupyter Welcome to Python (autosaved)". The main content area displays a temporary notebook service message from Rackspace:

jupyter

Welcome to the Temporary Notebook (tmpnb) service!

This Notebook Server was launched just for you. It's a temporary way for you to try out a recent development version of the IPython/Jupyter notebook.

WARNING
Don't rely on this server for anything you want to last - your server will be deleted after 10 minutes of inactivity.

Your server is hosted thanks to [Rackspace](#), on their on-demand bare metal servers, [OnMetal](#).

Run some Python code!

To run the code below:

1. Click on the cell to select it.
2. Press SHIFT+ENTER on your keyboard or press the play button (▶) in the toolbar above.

A full tutorial for using the notebook interface is available [here](#).

```
In [1]: %matplotlib notebook
import pandas as pd
import numpy as np
import matplotlib

from matplotlib import pyplot as plt
import seaborn as sns

ts = pd.Series(np.random.randn(1000), index=pd.date_range('1/1/2000', periods=1000))
ts = ts.cumsum()
df = pd.DataFrame(np.random.rand(1000, 4), index=ts.index,
                  columns=['A', 'B', 'C', 'D'])
df = df.cumsum()
df.plot(); plt.legend(loc='best')
```

Feel free to open new cells using the plus button (+), or hitting shift-enter while this cell is selected.

Behind the scenes, the software that powers this is [tmpnb](#), a Tornado application that spawns pre-built Docker containers and then uses the [jupyter/configurable-http-proxy](#) to put your notebook server on a unique path.

Autres IDE





- ▶ **Spyder** (Scientific PYthon Development EnviRonment) est un environnement scientifique puissant écrit en **Python**, pour **Python**, et conçu par et pour les scientifiques, les ingénieurs et les analystes de données.



- ▶ **Spyder** (Scientific PYthon Development EnviRonment) est un environnement scientifique puissant écrit en **Python**, pour **Python**, et conçu par et pour les scientifiques, les ingénieurs et les analystes de données.
- ▶ **Spyder** propose à la fois un éditeur et une console, avec en outre de nombreuses fonctionnalités supplémentaires, comme la gestion de projet, un explorateur de fichier, un historique des commandes, un debugger, etc.



- ▶ **Spyder** (Scientific PYthon Development EnviRonment) est un environnement scientifique puissant écrit en **Python**, pour **Python**, et conçu par et pour les scientifiques, les ingénieurs et les analystes de données.
- ▶ **Spyder** propose à la fois un éditeur et une console, avec en outre de nombreuses fonctionnalités supplémentaires, comme la gestion de projet, un explorateur de fichier, un historique des commandes, un debugger, etc.
- ▶ Pour lancer **Spyder**, on peut passer par un terminal, en évaluant tout simplement **Spyder** (ou en lançant le logiciel depuis le menu démarrer sous Windows). Il est également possible de lancer **Spyder** depuis **Anaconda**.



- ▶ **Spyder** (Scientific PYthon Development EnviRonment) est un environnement scientifique puissant écrit en **Python**, pour **Python**, et conçu par et pour les scientifiques, les ingénieurs et les analystes de données.
- ▶ **Spyder** propose à la fois un éditeur et une console, avec en outre de nombreuses fonctionnalités supplémentaires, comme la gestion de projet, un explorateur de fichier, un historique des commandes, un debugger, etc.
- ▶ Pour lancer **Spyder**, on peut passer par un terminal, en évaluant tout simplement **Spyder** (ou en lançant le logiciel depuis le menu démarrer sous Windows). Il est également possible de lancer **Spyder** depuis **Anaconda**.
- ▶ Une présentation de l'environnement **Spyder** est détaillée dans la vidéo suivante :

<https://www.youtube.com/watch?v=Txibtd8zWBs>



Spyder (Python 3.8)

Fichier Édition Recherche Source Exécution Déboguer Consoles Projets Outils Affichage Aide

Éditeur - /home/dominique/.config/spyder-py3/temp.py

temp.py X

```
1 # -*- coding: utf-8 -*-
2 """
3 Éditeur de Spyder
4 Ceci est un script temporaire.
5 """
6
7
8 for i in range(10):
9     print("Hello World %d" % i)
```

Aide Source Console Objet

Utilisation
Pour obtenir de l'aide ici, sélectionner (ou placer le curseur sur)

Explorateur de vari... Explorateur de fic... A...

Console IPython

Console 1/A X

IPython 7.12.0 -- An enhanced Interactive Python.

In [1]: runfile('/home/dominique/.config/spyder-py3/temp.py', wdir='/home/dominique/.config/spyder-py3')

Hello World 0
Hello World 1
Hello World 2
Hello World 3
Hello World 4
Hello World 5
Hello World 6
Hello World 7
Hello World 8
Hello World 9

Console IPython Historique

Droits d'accès : RW Fins de ligne : LF Encodage : UTF-8 Ligne : 9 Colon 32 Mémoire : 55 %

Table des matières

- ① Avant-propos
- ② Environnement Python
- ③ Bases d'utilisation
 - Notion de variable
 - Types standards
 - Chaînes de caractères
 - Affichage
- ④ Outils de contrôle de flux
- ⑤ Les collections de données
- ⑥ Les fonctions en Python
- ⑦ Les modules/librairies en Python
- ⑧ Les fichiers entrées/sorties en Python

Identificateurs et mots clés

- ▶ Il est impossible de faire du calcul dans un ordinateur sans utiliser de **variable**.

Identificateurs et mots clés

- ▶ Il est impossible de faire du calcul dans un ordinateur sans utiliser de **variable**.
- ▶ En programmation, une variable décrit un emplacement mémoire où sont rangées des informations : Nombre, Texte, Listes de nombres et de texte, etc.

Identificateurs et mots clés

- ▶ Il est impossible de faire du calcul dans un ordinateur sans utiliser de **variable**.
- ▶ En programmation, une variable décrit un emplacement mémoire où sont rangées des informations : Nombre, Texte, Listes de nombres et de texte, etc.
- ▶ Créer une variable nommée **Formation**, consiste à utiliser le signe égal (=), suivi de l'information que la variable désigne.
» Formation = "Licence IRM"

Identificateurs et mots clés

- ▶ Il est impossible de faire du calcul dans un ordinateur sans utiliser de **variable**.
- ▶ En programmation, une variable décrit un emplacement mémoire où sont rangées des informations : Nombre, Texte, Listes de nombres et de texte, etc.
- ▶ Créer une variable nommée **Formation**, consiste à utiliser le signe égal (=), suivi de l'information que la variable désigne.
 - » Formation = "Licence IRM"
- ▶ Cette affectation crée une variable nommée **Formation** et lui affecte la chaîne de caractères **Licence IRM**. On peut dès lors utiliser cette variable.

Identificateurs et mots clés

- ▶ Il est impossible de faire du calcul dans un ordinateur sans utiliser de **variable**.
- ▶ En programmation, une variable décrit un emplacement mémoire où sont rangées des informations : Nombre, Texte, Listes de nombres et de texte, etc.
- ▶ Créer une variable nommée **Formation**, consiste à utiliser le signe égal (=), suivi de l'information que la variable désigne.
 - » Formation = "Licence IRM"
- ▶ Cette affectation crée une variable nommée **Formation** et lui affecte la chaîne de caractères **Licence IRM**. On peut dès lors utiliser cette variable.
- ▶ L'instruction **print** permet d'afficher le contenu d'une variable :
 - » **print(Formation)**

Identificateurs et mots clés

- ▶ Lorsqu'on écrit un programme, on doit fixer des noms significatifs pour les variables de ce programme.

Identificateurs et mots clés

- ▶ Lorsqu'on écrit un programme, on doit fixer des noms significatifs pour les variables de ce programme.
- ▶ **Python** a des règles sur la façon dont on peut former des noms :



Identificateurs et mots clés

- ▶ Lorsqu'on écrit un programme, on doit fixer des noms significatifs pour les variables de ce programme.
- ▶ **Python** a des règles sur la façon dont on peut former des noms :
 - * Un nom doit commencer obligatoirement par une lettre ou le caractère de soulignement (_) et peut contenir des lettres, des chiffres ou le caractère de soulignement.

Identificateurs et mots clés

- ▶ Lorsqu'on écrit un programme, on doit fixer des noms significatifs pour les variables de ce programme.
- ▶ **Python** a des règles sur la façon dont on peut former des noms :
 - * Un nom doit commencer obligatoirement par une lettre ou le caractère de soulignement (_) et peut contenir des lettres, des chiffres ou le caractère de soulignement.
 - * **Python** considère les lettres majuscules et minuscules différemment ; par exemple, LIC_IRM est considérée par **Python** comme un nom différent de lic_irm ou Lic_irm.

Identificateurs et mots clés

- ▶ Lorsqu'on écrit un programme, on doit fixer des noms significatifs pour les variables de ce programme.
- ▶ **Python** a des règles sur la façon dont on peut former des noms :
 - * Un nom doit commencer obligatoirement par une lettre ou le caractère de soulignement (_) et peut contenir des lettres, des chiffres ou le caractère de soulignement.
 - * **Python** considère les lettres majuscules et minuscules différemment ; par exemple, LIC_IRM est considérée par **Python** comme un nom différent de lic_irm ou Lic_irm.
 - * Plus de détails peuvent être trouvés ici :

<https://www.python.org/dev/peps/pep-0008/#naming-conventions>

Identificateurs et mots clés

- ▶ Les identificateurs suivants sont utilisés comme des mots réservés, et ne peuvent pas être utilisés comme des identificateurs ordinaires :

and	del	false	lambda	return
as	elif	from	none	true
assert	else	global	not	try
break	except	if	or	while
class	exec	import	pass	with
continue	finally	in	print	yield
def	for	is	raise	

Table des matières

- ① Avant-propos
- ② Environnement Python
- ③ Bases d'utilisation
 - Notion de variable
 - Types standards
 - Chaînes de caractères
 - Affichage
- ④ Outils de contrôle de flux
- ⑤ Les collections de données
- ⑥ Les fonctions en Python
- ⑦ Les modules/librairies en Python
- ⑧ Les fichiers entrées/sorties en Python

Le type des variables

Important

Comme le typage est dynamique en **Python**, le type n'est pas précisé explicitement, il est implicitement lié à l'information manipulée.

- ▶ Les variables **Python** ne sont donc pas associées à des types, mais les valeurs qu'elles contiennent, elles, le sont !

Important

Comme le typage est dynamique en **Python**, le type n'est pas précisé explicitement, il est implicitement lié à l'information manipulée.

- ▶ Les variables **Python** ne sont donc pas associées à des types, mais les valeurs qu'elles contiennent, elles, le sont !
- ▶ C'est au programmeur de savoir le type de la valeur contenue dans une variable à un moment donné pour ne pas faire d'opération incompatible.

Le type des variables

Important

Comme le typage est dynamique en **Python**, le type n'est pas précisé explicitement, il est implicitement lié à l'information manipulée.

- ▶ Les variables **Python** ne sont donc pas associées à des types, mais les valeurs qu'elles contiennent, elles, le sont !
- ▶ C'est au programmeur de savoir le type de la valeur contenue dans une variable à un moment donné pour ne pas faire d'opération incompatible.
- ▶ **Python** présente 5 types de base : Entier, Décimal, Complexe, Booléen et rien.

Le type des variables

Important

Comme le typage est dynamique en **Python**, le type n'est pas précisé explicitement, il est implicitement lié à l'information manipulée.

- ▶ Les variables **Python** ne sont donc pas associées à des types, mais les valeurs qu'elles contiennent, elles, le sont !
- ▶ C'est au programmeur de savoir le type de la valeur contenue dans une variable à un moment donné pour ne pas faire d'opération incompatible.
- ▶ **Python** présente 5 types de base : Entier, Décimal, Complexe, Booléen et rien.
- ▶ L'instruction **type** permet d'afficher le type d'une variable :
 - »» `type(Formation)`
 - `<class 'str'>`

Le type des variables

- Le tableau suivant donne un aperçu des types de données les plus fréquemment utilisés en **Python**.

Type de Variable	Exemple	Commentaire sur l'utilisation
bool	Module_vlide = True Doubler = False	Valeurs true/false
int, long	taille_chaussures = 42 rayon_Ter = 6 371 000	Différents chiffres entiers
float	pi = 3.14159265359	Les nombres qui ont un point décimal
str	Licence = "IRM"	Tout texte
None	my_Stock = None	Variable vide sans aucune valeur significative
complex	onde = 3 + 6j	Différents chiffres complexes

Opérations de base

- ▶ **Python** a un paquet d'opérateurs arithmétiques intégrés. Ci-dessous on fournit une brève comparaison de la notation courte et de leurs équivalents plus longs.

Opérations de base

- ▶ **Python** a un paquet d'opérateurs arithmétiques intégrés. Ci-dessous on fournit une brève comparaison de la notation courte et de leurs équivalents plus longs.
- ▶ Soit la variable $a = 3$.

Opérations de base

- ▶ **Python** a un paquet d'opérateurs arithmétiques intégrés. Ci-dessous on fournit une brève comparaison de la notation courte et de leurs équivalents plus longs.
- ▶ Soit la variable $a = 3$.

Opérations de base

- ▶ Python a un paquet d'opérateurs arithmétiques intégrés. Ci-dessous on fournit une brève comparaison de la notation courte et de leurs équivalents plus longs.
- ▶ Soit la variable $a = 3$.

Opération	Notation courte	Notation longue	Valeur de a	Commentaire
Addition	$a += 1$	$a = a + 1$	4	
Soustraction	$a -= 1$	$a = a - 1$	2	
Multiplication	$a *= 2$	$a = a * 2$	6	
Division	$a /= 2$	$a = a / 2$	1	Retourne des valeurs décimales ou flottants

Opérations de base

- ▶ Python a un paquet d'opérateurs arithmétiques intégrés. Ci-dessous fournit une brève comparaison des notations courtes et de leurs équivalentes plus longues.
- ▶ Soit la variable $a = 3$.

Opération	Notation courte	Notation longue	Valeur de a	Commentaire
Modulo	$a \%= 2$	$a = a \% 2$	1	Retourne le reste entier de la division
Puissance	$a **= 2$	$a = a ** 2$	9	Semblable à a^2 en mathématique
Division entière	$a //= 2$	$a = a // 2$	1	Renvoie uniquement des valeurs décimales (int)
Négation	$a = -a$	$a = 0 - a$	-3	Retourne la même valeur avec un signe opposé

Opérations de base

- ▶ L'ordre d'exécution des opérations mathématiques en **Python** est similaire à l'ordre utilisé dans les mathématiques classiques.



Table des matières

- ① Avant-propos
- ② Environnement Python
- ③ Bases d'utilisation
 - Notion de variable
 - Types standards
 - Chaînes de caractères**
 - Affichage
- ④ Outils de contrôle de flux
- ⑤ Les collections de données
- ⑥ Les fonctions en Python
- ⑦ Les modules/librairies en Python
- ⑧ Les fichiers entrées/sorties en Python

Opérations sur les chaînes de caractères

- En **Python**, les mots et les phrases sont des collections de caractères.

Opérations sur les chaînes de caractères

- ▶ En **Python**, les mots et les phrases sont des collections de caractères.
- ▶ On peut accéder à n'importe quel caractère d'une chaîne de caractères, en **Python**, en utilisant son index, en comptant à partir du début ou de la fin de la chaîne.

Opérations sur les chaînes de caractères

- ▶ En **Python**, les mots et les phrases sont des collections de caractères.
- ▶ On peut accéder à n'importe quel caractère d'une chaîne de caractères, en **Python**, en utilisant son index, en comptant à partir du début ou de la fin de la chaîne.
- ▶ Lorsqu'on commence par le début, on compte à partir de 0.

Opérations sur les chaînes de caractères

- ▶ En **Python**, les mots et les phrases sont des collections de caractères.
- ▶ On peut accéder à n'importe quel caractère d'une chaîne de caractères, en **Python**, en utilisant son index, en comptant à partir du début ou de la fin de la chaîne.
- ▶ Lorsqu'on commence par le début, on compte à partir de 0.
- ▶ Lorsqu'on commence par la fin, on compte à partir de -1 .

Opérations sur les chaînes de caractères

- ▶ En **Python**, les mots et les phrases sont des collections de caractères.
- ▶ On peut accéder à n'importe quel caractère d'une chaîne de caractères, en **Python**, en utilisant son index, en comptant à partir du début ou de la fin de la chaîne.
- ▶ Lorsqu'on commence par le début, on compte à partir de 0.
- ▶ Lorsqu'on commence par la fin, on compte à partir de -1 .

Opérations sur les chaînes de caractères

- ▶ En **Python**, les mots et les phrases sont des collections de caractères.
- ▶ On peut accéder à n'importe quel caractère d'une chaîne de caractères, en **Python**, en utilisant son index, en comptant à partir du début ou de la fin de la chaîne.
- ▶ Lorsqu'on commence par le début, on compte à partir de 0.
- ▶ Lorsqu'on commence par la fin, on compte à partir de -1 .

senti =	"	I	f	e	e	I	g	o	o	d	"
Indice		0	1	2	3	4	5	6	7	8	9
		-11	-10	-9	-8	-7	-6	-5	-4	-3	-2

Éléments d'une chaîne de caractère

- ▶ Pour adresser (accéder à) un élément particulier d'une chaîne de caractère, on dispose de l'opérateur [].

Éléments d'une chaîne de caractère

- ▶ Pour adresser (accéder à) un élément particulier d'une chaîne de caractère, on dispose de l'opérateur [].
- ▶ Par exemple :

Éléments d'une chaîne de caractère

- ▶ Pour adresser (accéder à) un élément particulier d'une chaîne de caractère, on dispose de l'opérateur [].
- ▶ Par exemple :

Éléments d'une chaîne de caractère

- ▶ Pour adresser (accéder à) un élément particulier d'une chaîne de caractère, on dispose de l'opérateur [].
- ▶ Par exemple :

» print(senti)

I fell good

Éléments d'une chaîne de caractère

- ▶ Pour adresser (accéder à) un élément particulier d'une chaîne de caractère, on dispose de l'opérateur [].
- ▶ Par exemple :

»» print(senti)

I fell good

»» print(senti[2])

f

Éléments d'une chaîne de caractère

- ▶ Pour adresser (accéder à) un élément particulier d'une chaîne de caractère, on dispose de l'opérateur [].
- ▶ Par exemple :

»» print(senti)

I fell good

»» print(senti[2])

f

»» print(senti[-4])

g

Éléments d'une chaîne de caractère

- ▶ Pour adresser (accéder à) un élément particulier d'une chaîne de caractère, on dispose de l'opérateur [].
- ▶ Par exemple :

»» print(senti)

I fell good

»» print(senti[2])

f

»» print(senti[-4])

g

»» print(senti[-11])

I

Découpage en tranches

- ▶ On peut prendre des sous-chaînes plus courtes à l'intérieur d'une chaîne plus longue. Cette opération est appelée "tranchage" ou slicing en anglais.

» formation = 'Licence IRM'

» formation[2:7] # du 2ème au 6ème, le 7ème est exclu
cenc

Découpage en tranches

- ▶ On peut prendre des sous-chaînes plus courtes à l'intérieur d'une chaîne plus longue. Cette opération est appelée "tranchage" ou slicing en anglais.
 - » formation = 'Licence IRM'
 - » formation[2:7] # du 2ème au 6ème, le 7ème est exclu cenc
- ▶ Pour vérifier si une chaînes de caractères contient une sous-chaîne particulière, on peut utiliser l'opérateur "**in**".
 - » 'L' **in** formation[1:3]
False
 - » 'cence' **in** formation[0:8]
True

Concaténation

- En **Python**, il est possible de combiner plusieurs chaînes de caractères en une seule, cette opération est appelée concaténation.

Concaténation

- ▶ En **Python**, il est possible de combiner plusieurs chaînes de caractères en une seule, cette opération est appelée concaténation.
- ▶ Pour concaténer des chaînes de caractères, c'est-à-dire les mettre bout à bout, **Python** propose d'utiliser l'opérateur + :
 - » formation = 'Licence IRM'
 - » Promo = '8ème promotion'
 - » Ann_Scol = '2024 - 2025'
 - » Actuel = formation + ',' + ' ' + Promo + ' ' + Ann_Scol
 - » **print(Actuel)**

Licence IRM, 8ème promotion 2024 - 2025

Opérations sur les chaînes de caractères

- ▶ De nombreuses méthodes sont disponibles pour les chaînes de caractères.

En ajoutant un point (.) après le nom d'un objet désignant une chaîne de caractères puis en appuyant sur la touche de tabulation, les méthodes disponibles s'affichent dans un menu déroulant.

Opérations sur les chaînes de caractères

- ▶ De nombreuses méthodes sont disponibles pour les chaînes de caractères.
En ajoutant un point (.) après le nom d'un objet désignant une chaîne de caractères puis en appuyant sur la touche de tabulation, les méthodes disponibles s'affichent dans un menu déroulant.
- ▶ Par exemple, la méthode **count()** permet de compter le nombre d'occurrences d'un motif dans la chaîne.

Opérations sur les chaînes de caractères

- ▶ De nombreuses méthodes sont disponibles pour les chaînes de caractères.

En ajoutant un point (.) après le nom d'un objet désignant une chaîne de caractères puis en appuyant sur la touche de tabulation, les méthodes disponibles s'affichent dans un menu déroulant.

- ▶ Par exemple, la méthode **count()** permet de compter le nombre d'occurrences d'un motif dans la chaîne.
- ▶ Pour compter le nombre d'occurrence de "ma" dans la chaîne suivante :
 - »» ch = "maman aime manger"
 - »» **print(ch.count(" ma"))**

Opérations sur les chaînes de caractères

- ▶ De nombreuses méthodes sont disponibles pour les chaînes de caractères.

En ajoutant un point (.) après le nom d'un objet désignant une chaîne de caractères puis en appuyant sur la touche de tabulation, les méthodes disponibles s'affichent dans un menu déroulant.

- ▶ Par exemple, la méthode **count()** permet de compter le nombre d'occurrences d'un motif dans la chaîne.
- ▶ Pour compter le nombre d'occurrence de "ma" dans la chaîne suivante :
 - »» ch = "maman aime manger"
 - »» **print(ch.count(" ma"))**

Opérations sur les chaînes de caractères

- ▶ De nombreuses méthodes sont disponibles pour les chaînes de caractères.

En ajoutant un point (.) après le nom d'un objet désignant une chaîne de caractères puis en appuyant sur la touche de tabulation, les méthodes disponibles s'affichent dans un menu déroulant.

- ▶ Par exemple, la méthode **count()** permet de compter le nombre d'occurrences d'un motif dans la chaîne.
- ▶ Pour compter le nombre d'occurrence de "ma" dans la chaîne suivante :

»» ch = "maman aime manger"

»» **print(ch.count(" ma"))**

Opérations sur les chaînes de caractères

- ▶ La fonction "**len()**" permet de calculer la taille d'une chaîne de caractères.

Opérations sur les chaînes de caractères

- ▶ La fonction "**len()**" permet de calculer la taille d'une chaîne de caractères.

Opérations sur les chaînes de caractères

- ▶ La fonction "**len()**" permet de calculer la taille d'une chaîne de caractères.

```
» str = "anticonstitutionnellement"
```

```
» print(len(str))
```

25

- ▶ Les méthodes "**lower()**", "**upper()**" et "**capitalize()**" renvoient respectivement une chaîne de caractères en minuscules, majuscules, et avec la première lettre en majuscule.

Opérations sur les chaînes de caractères

- ▶ La fonction "**len()**" permet de calculer la taille d'une chaîne de caractères.

```
» str = "anticonstitutionnellement"
```

```
» print(len(str))
```

25

- ▶ Les méthodes "**lower()**", "**upper()**" et "**capitalize()**" renvoient respectivement une chaîne de caractères en minuscules, majuscules, et avec la première lettre en majuscule.

Opérations sur les chaînes de caractères

- ▶ La fonction "**len()**" permet de calculer la taille d'une chaîne de caractères.

```
»» str = "anticonstitutionnellement"
```

```
»» print(len(str))
```

25

- ▶ Les méthodes "**lower()**", "**upper()**" et "**capitalize()**" renvoient respectivement une chaîne de caractères en minuscules, majuscules, et avec la première lettre en majuscule.

```
»» str = "Recto-Verso"
```

```
»» print(str.lower())
```

recto-verso

```
»» str = "licence"
```

```
»» print(str.upper())
```

LICENCE

```
»» str = "python"
```

```
»» print(str.capitalize())
```

Python

Opérations sur les chaînes de caractères

- ▶ La méthode "**strip()**" permet de supprimer les espaces superflus en début et en fin de chaîne.

Opérations sur les chaînes de caractères

- ▶ La méthode "**strip()**" permet de supprimer les espaces superflus en début et en fin de chaîne.

Opérations sur les chaînes de caractères

- ▶ La méthode "**strip()**" permet de supprimer les espaces superflus en début et en fin de chaîne.

```
»» str = " Bien venue! "
»» print(str.strip())
```

Bien venue!

- ▶ La méthode "**find()**", permet de chercher la première occurrence d'un caractère ou d'une séquence de caractères et renvoie leur position.

Opérations sur les chaînes de caractères

- ▶ La méthode "**strip()**" permet de supprimer les espaces superflus en début et en fin de chaîne.

```
»» str = " Bien venue! "
»» print(str.strip())
```

Bien venue!

- ▶ La méthode "**find()**", permet de chercher la première occurrence d'un caractère ou d'une séquence de caractères et renvoie leur position.

Opérations sur les chaînes de caractères

- ▶ La méthode "**strip()**" permet de supprimer les espaces superflus en début et en fin de chaîne.

```
»» str = " Bien venue! "
»» print(str.strip())
```

Bien venue!

- ▶ La méthode "**find()**", permet de chercher la première occurrence d'un caractère ou d'une séquence de caractères et renvoie leur position.

```
»» str = "Python est le langage le plus employé par les data scientistes."
»» print(str.find("langage"))
```

14

Opérations sur les chaînes de caractères

- ▶ Les méthodes "**startswith()**" et "**endswith()**" permettent de vérifier si une chaîne commence ou se termine bien par un caractère ou par une séquence de caractères et renvoient un booléen.

Opérations sur les chaînes de caractères

- ▶ Les méthodes "**startswith()**" et "**endswith()**" permettent de vérifier si une chaîne commence ou se termine bien par un caractère ou par une séquence de caractères et renvoient un booléen.

Opérations sur les chaînes de caractères

- ▶ Les méthodes "**startswith()**" et "**endswith()**" permettent de vérifier si une chaîne commence ou se termine bien par un caractère ou par une séquence de caractères et renvoient un booléen.

» str = "Commence par Com."

» `print(str.startswith("Com"))`

True

Opérations sur les chaînes de caractères

- ▶ Les méthodes "**startswith()**" et "**endswith()**" permettent de vérifier si une chaîne commence ou se termine bien par un caractère ou par une séquence de caractères et renvoient un booléen.

» str = "Commence par Com."

» **print(str.startswith("Com"))**

True

» str = "Termine par mines."

» **print(str.endswith("mines"))**

False

Opérations sur les chaînes de caractères

D'autres Méthodes

Méthode	Description
capitalize()	Convertit le premier caractère en majuscule
casefold()	Convertit une chaîne de caractères en minuscules
center()	Retourne une chaîne centrée
count()	Renvoie le nombre de fois qu'une valeur donnée apparaît dans une chaîne
encode()	Renvoie une version codée de la chaîne
endswith()	Retourne vrai si la chaîne se termine par la valeur spécifiée
expandtabs()	Définit la taille de l'onglet de la chaîne
find()	Recherche la chaîne de caractères pour une valeur donnée et renvoie la position de l'endroit où elle a été trouvée

Opérations sur les chaînes de caractères

D'autres Méthodes

Méthode	Description
format()	Formats des valeurs spécifiées dans une chaîne
format_map()	Formats des valeurs spécifiées dans une chaîne
index()	Recherche la chaîne de caractères pour une valeur donnée et renvoie la position de l'endroit où elle a été trouvée
isalnum()	Retourne Vrai si tous les caractères de la chaîne sont alphanumériques
isalpha()	Retourne Vrai si tous les caractères de la chaîne sont dans l'alphabet
isdecimal()	Retourne Vrai si tous les caractères de la chaîne sont des décimales
isdigit()	Retourne Vrai si tous les caractères de la chaîne sont des chiffres
isidentifier()	Retourne True si la chaîne est un identifiant

Opérations sur les chaînes de caractères

D'autres Méthodes

Méthode	Description
islower()	Retourne Vrai si tous les caractères de la chaîne sont en minuscules
isnumeric()	Retourne Vrai si tous les caractères de la chaîne sont numériques
isprintable()	Retourne Vrai si tous les caractères de la chaîne sont imprimables
isspace()	Retourne Vrai si tous les caractères de la chaîne sont des espaces
istitle()	Retourne Vrai si la chaîne suit les règles d'un titre
isupper()	Retourne Vrai si tous les caractères de la chaîne sont en majuscules
join(seq)	Joint les éléments d'un itérable à l'extrémité de la chaîne
ljust()	Retourne une version justifiée à gauche de la chaîne
lower()	Convertit une chaîne de caractères en minuscules

Opérations sur les chaînes de caractères

D'autres Méthodes

Méthode	Description
lstrip()	Retourne une version de la chaîne coupée à gauche
maketrans()	Renvoie un tableau de traduction à utiliser dans les traductions
partition()	Renvoie un tuple où la corde est divisée en trois parties
replace()	Renvoie une chaîne de caractères où une valeur spécifiée est remplacée par une valeur spécifiée
rfind()	Recherche la chaîne de caractères pour une valeur donnée et renvoie la dernière position de l'endroit où elle a été trouvée
rindex()	Recherche la chaîne de caractères pour une valeur donnée et renvoie la dernière position de l'endroit où elle a été trouvée

Opérations sur les chaînes de caractères

D'autres Méthodes

Méthode	Description
rjust()	Retourne une version justifiée de la chaîne
rpartition()	Renvoie un tuple où la corde est divisée en trois parties
rsplit()	Sépare la chaîne de caractères au niveau du séparateur spécifié et renvoie une liste
rstrip()	Retourne une version de la chaîne coupée à droite
split()	Sépare la chaîne de caractères au niveau du séparateur spécifié et renvoie une liste
splitlines()	Sépare la chaîne de caractères à la fin de la ligne et renvoie une liste
startswith()	Retourne vrai si la chaîne commence par la valeur spécifiée
strip()	Retourne une version coupée de la chaîne

Opérations sur les chaînes de caractères

D'autres Méthodes

Méthode	Description
swapcase()	Échange de cas, les minuscules deviennent des majuscules et vice versa
title()	Convertit le premier caractère de chaque mot en majuscule
translate()	Renvoie une chaîne traduite
upper()	Convertit une chaîne de caractères en majuscules
zfill()	Remplit la chaîne avec un nombre spécifié de valeurs 0 au début

Les commentaires

- ▶ Les commentaires sont destinés à aider l'utilisateur à comprendre le code, et sont ignorés par l'interprète.

Les commentaires

- ▶ Les commentaires sont destinés à aider l'utilisateur à comprendre le code, et sont ignorés par l'interprète.
- ▶ Sous **Python**, l'intégration d'un commentaire dans le code peut être faite de deux manières différentes ; les commentaires sur une seule ligne et les commentaires portant sur plusieurs lignes.

Les commentaires

- ▶ Les commentaires sont destinés à aider l'utilisateur à comprendre le code, et sont ignorés par l'interprète.
- ▶ Sous **Python**, l'intégration d'un commentaire dans le code peut être faite de deux manières différentes ; les commentaires sur une seule ligne et les commentaires portant sur plusieurs lignes.
 - ✳️ Les commentaires portant sur une seule ligne sont obtenus en mettant le caractère `#` au début.

Les commentaires

- ▶ Les commentaires sont destinés à aider l'utilisateur à comprendre le code, et sont ignorés par l'interprète.
- ▶ Sous **Python**, l'intégration d'un commentaire dans le code peut être faite de deux manières différentes ; les commentaires sur une seule ligne et les commentaires portant sur plusieurs lignes.
 - ✳ Les commentaires portant sur une seule ligne sont obtenus en mettant le caractère `#` au début.
 - ✳ Au lieu d'enchaîner plusieurs commentaires d'une seule ligne, pour les commentaires plus importants, on peut ajouter un triple guillemets "consécutifs" `"""` au début de la première ligne et à la fin de la dernière ligne.

Table des matières

- ① Avant-propos
- ② Environnement Python
- ③ Bases d'utilisation
 - Notion de variable
 - Types standards
 - Chaînes de caractères
 - Affichage
- ④ Outils de contrôle de flux
- ⑤ Les collections de données
- ⑥ Les fonctions en Python
- ⑦ Les modules/librairies en Python
- ⑧ Les fichiers entrées/sorties en Python

Écriture formatée

- ▶ La fonction `print()` qui permet d'afficher une chaîne de caractères. Elle permet en plus d'afficher le contenu d'une ou plusieurs variables :

Écriture formatée

- ▶ La fonction `print()` qui permet d'afficher une chaîne de caractères. Elle permet en plus d'afficher le contenu d'une ou plusieurs variables :

Écriture formatée

- ▶ La fonction `print()` qui permet d'afficher une chaîne de caractères. Elle permet en plus d'afficher le contenu d'une ou plusieurs variables :

```
»> age = 32, pren = 'Ahmed'  
»> print(pren, ' a ', age, ' ans')
```

Ahmed a 32 ans

- ▶ La méthode `.format()` offre une bonne organisation de l'affichage des variables.

Écriture formatée

- ▶ La fonction `print()` qui permet d'afficher une chaîne de caractères. Elle permet en plus d'afficher le contenu d'une ou plusieurs variables :

```
»> age = 32, pren = 'Ahmed'  
»> print(pren, ' a ', age, ' ans')
```

Ahmed a 32 ans

- ▶ La méthode `.format()` offre une bonne organisation de l'affichage des variables.

Écriture formatée

- ▶ La fonction `print()` qui permet d'afficher une chaîne de caractères. Elle permet en plus d'afficher le contenu d'une ou plusieurs variables :

```
»» age = 32, pren = 'Ahmed'  
»» print(pren, ' a ', age, ' ans')
```

Ahmed a 32 ans

- ▶ La méthode `.format()` offre une bonne organisation de l'affichage des variables.

```
»» tier = 1 / 3  
»» print("Le tier est", tier)  
»» print("Le tier est {:.2f}".format(tier))  
»» print("Le tier est {:.3f}".format(tier))
```

Le tier est 0.3333333333333333

Le tier est 0.33

Le tier est 0.333

Écriture formatée

- ▶ Il est possible de préciser sur combien de caractères vous voulez qu'un résultat soit écrit et comment se fait l'alignement (à gauche, à droite ou centré) :

Écriture formatée

- ▶ Il est possible de préciser sur combien de caractères vous voulez qu'un résultat soit écrit et comment se fait l'alignement (à gauche, à droite ou centré) :

Écriture formatée

- ▶ Il est possible de préciser sur combien de caractères vous voulez qu'un résultat soit écrit et comment se fait l'alignement (à gauche, à droite ou centré) :

```
»> print(10) ; print(1000)
```

10

1000

```
»> print(" {:>6d}".format(10)) ; print(" {:>6d}".format(1000)) # à droite
```

10

1000

```
»> print(" {:<6d}".format(10)) ; print(" {:<6d}".format(1000)) # à gauche
```

10

1000

```
»> print(" {:^6d}".format(10)) ; print(" {:^6d}".format(1000)) # centré
```

10

1000

Écriture formatée

- ▶ Pour les nombres réels, on peut contrôler le nombre de chiffres après la virgule :

Écriture formatée

- ▶ Pour les nombres réels, on peut contrôler le nombre de chiffres après la virgule :

Écriture formatée

- ▶ Pour les nombres réels, on peut contrôler le nombre de chiffres après la virgule :

```
»» a = (1024 + 2597)/2359
```

```
»» print("la valeur de a est : ", a)
```

```
la valeur de a est : 1.5349724459516745
```

- ▶ Pour plus de lisibilité, on peut spécifier dans les accolades {} le format souhaité :

Écriture formatée

- ▶ Pour les nombres réels, on peut contrôler le nombre de chiffres après la virgule :

```
»> a = (1024 + 2597)/2359
```

```
»> print("la valeur de a est : ", a)
```

```
la valeur de a est : 1.5349724459516745
```

- ▶ Pour plus de lisibilité, on peut spécifier dans les accolades {} le format souhaité :

Écriture formatée

- ▶ Pour les nombres réels, on peut contrôler le nombre de chiffres après la virgule :

```
»> a = (1024 + 2597)/2359
```

```
»> print("la valeur de a est : ", a)
```

```
la valeur de a est : 1.5349724459516745
```

- ▶ Pour plus de lisibilité, on peut spécifier dans les accolades {} le format souhaité :

```
»> print("la valeur de a est {:.3f }".format(a))
```

```
la valeur de a est : 1.535
```

- ✓ Les deux points : indiquent que l'on veut préciser le format.
- ✓ La lettre f indique que l'affichage est sous forme d'un réel (float).
- ✓ Les caractères .3 indiquent la précision voulue, soit ici 3 chiffres après la virgule.
- ✓ Notez enfin que le formatage avec .xf ($x > 0$) renvoie un résultat arrondi.

Écriture formatée

- ▶ Il est également possible d'indiquer le caractère qui servira de remplissage lors des alignements (l'espace par défaut) :

Écriture formatée

- ▶ Il est également possible d'indiquer le caractère qui servira de remplissage lors des alignements (l'espace par défaut) :

Écriture formatée

- ▶ Il est également possible d'indiquer le caractère qui servira de remplissage lors des alignements (l'espace par défaut) :

```
»> print(10) ; print(1000)
```

10

1000

```
»> print(" {:+ <6d}".format(10)) ; print(" {:+ <6d}".format(1000))
```

10++++

1000++

```
»> print(" {:*^6d}".format(10)) ; print(" {:*^6d}".format(1000))
```

10

1000

```
»> print(" :0 <6d".format(10)) ; print(" :0 <6d".format(1000))
```

000010

001000

Longues instructions

- ▶ Si une instruction de programmation est trop longue, cependant, vous ne pourrez pas la voir dans votre fenêtre d'édition sans faire défiler horizontalement.

Longues instructions

- ▶ Si une instruction de programmation est trop longue, cependant, vous ne pourrez pas la voir dans votre fenêtre d'édition sans faire défiler horizontalement.
- ▶ **Python** vous permet de diviser une instruction en plusieurs lignes en utilisant le caractère, *barre oblique inverse* (\) :

Longues instructions

- ▶ Si une instruction de programmation est trop longue, cependant, vous ne pourrez pas la voir dans votre fenêtre d'édition sans faire défiler horizontalement.
- ▶ **Python** vous permet de diviser une instruction en plusieurs lignes en utilisant le caractère, *barre oblique inverse* (\) :

Longues instructions

- ▶ Si une instruction de programmation est trop longue, cependant, vous ne pourrez pas la voir dans votre fenêtre d'édition sans faire défiler horizontalement.
- ▶ **Python** vous permet de diviser une instruction en plusieurs lignes en utilisant le caractère, *barre oblique inverse* (\) :

```
»> result = var1 * 2 + var2 * 3 + \
           var3 * 4 + var4 * 5
```

- ▶ **Python** vous permet également de d'éclater une instruction qui est entre parenthèses () sur plusieurs lignes sans utiliser \ :

Longues instructions

- ▶ Si une instruction de programmation est trop longue, cependant, vous ne pourrez pas la voir dans votre fenêtre d'édition sans faire défiler horizontalement.
- ▶ **Python** vous permet de diviser une instruction en plusieurs lignes en utilisant le caractère, *barre oblique inverse* (\) :

```
»> result = var1 * 2 + var2 * 3 + \
           var3 * 4 + var4 * 5
```

- ▶ **Python** vous permet également de d'éclater une instruction qui est entre parenthèses () sur plusieurs lignes sans utiliser \ :

Longues instructions

- ▶ Si une instruction de programmation est trop longue, cependant, vous ne pourrez pas la voir dans votre fenêtre d'édition sans faire défiler horizontalement.
- ▶ **Python** vous permet de diviser une instruction en plusieurs lignes en utilisant le caractère, *barre oblique inverse* (\) :

```
>>> result = var1 * 2 + var2 * 3 + \
           var3 * 4 + var4 * 5
```

- ▶ **Python** vous permet également de d'éclater une instruction qui est entre parenthèses () sur plusieurs lignes sans utiliser \ :

```
>>> result_moy = (var1 * 2 + var2 * 3 + var3 * 4 +
                  var4 * 5 + var5 * 7)/5
```

Suppression de la fin de ligne de la fonction d'affichage

- ▶ Chaque fonction `print()` affiche une nouvelle ligne de sortie :



Suppression de la fin de ligne de la fonction d'affichage

- ▶ Chaque fonction `print()` affiche une nouvelle ligne de sortie :

Suppression de la fin de ligne de la fonction d'affichage

- ▶ Chaque fonction `print()` affiche une nouvelle ligne de sortie :

```
>>> print('Licence')
>>> print('IRM')
>>> print('2023-2024')
```

Licence

IRM

2023-2024

- ▶ Pour continuer l'affichage sur une même ligne, on peut passer l'argument spécial `end = ''` comme suit :

Suppression de la fin de ligne de la fonction d'affichage

- ▶ Chaque fonction `print()` affiche une nouvelle ligne de sortie :

```
>>> print('Licence')
>>> print('IRM')
>>> print('2023-2024')
```

Licence

IRM

2023-2024

- ▶ Pour continuer l'affichage sur une même ligne, on peut passer l'argument spécial `end = ''` comme suit :

Suppression de la fin de ligne de la fonction d'affichage

- ▶ Chaque fonction `print()` affiche une nouvelle ligne de sortie :

```
»> print('Licence')
»> print('IRM')
»> print('2023-2024')
```

Licence

IRM

2023-2024

- ▶ Pour continuer l'affichage sur une même ligne, on peut passer l'argument spécial `end = ''` comme suit :

```
»> print('Licence', end = '')
»> print('IRM', end = '')
»> print('2023-2024')
```

Licence IRM 2023-2024

Table des matières

- ① Avant-propos
- ② Environnement Python
- ③ Bases d'utilisation
- ④ Outils de contrôle de flux
 - Le contrôle de conditions
 - Les structures de contrôle ou boucles
- ⑤ Les collections de données
- ⑥ Les fonctions en Python
- ⑦ Les modules/librairies en Python
- ⑧ Les fichiers entrées/sorties en Python

Les branchements conditionnels

- ▶ **Python** permet à un programme de procéder dans différentes directions en fonction du résultat d'un test, qui évalue une expression donnée pour une valeur booléenne de **Vrai** ou **Faux**, connu sous le nom de "branchement conditionnel".

Les branchements conditionnels

- ▶ **Python** permet à un programme de procéder dans différentes directions en fonction du résultat d'un test, qui évalue une expression donnée pour une valeur booléenne de **Vrai** ou **Faux**, connu sous le nom de "branchement conditionnel".
- ▶ Les tests permettent d'exécuter telle ou telle instruction selon la valeur d'une condition. Le test est suivi de **:** et les instructions dépendant de ce test sont **indentées** (*décalées vers la droite*).

Les branchements conditionnels

- ▶ **Python** permet à un programme de procéder dans différentes directions en fonction du résultat d'un test, qui évalue une expression donnée pour une valeur booléenne de **Vrai** ou **Faux**, connu sous le nom de "branchement conditionnel".
- ▶ Les tests permettent d'exécuter telle ou telle instruction selon la valeur d'une condition. Le test est suivi de **:** et les instructions dépendant de ce test sont **indentées** (*décalées vers la droite*).
- ▶ Pour contrôler conditionnellement la suite des instructions du programme, les blocs **Si/SiNon** sont un élément syntaxique de base en **Python**.

Les branchements conditionnels

- ▶ **Python** permet à un programme de procéder dans différentes directions en fonction du résultat d'un test, qui évalue une expression donnée pour une valeur booléenne de **Vrai** ou **Faux**, connu sous le nom de "branchement conditionnel".
- ▶ Les tests permettent d'exécuter telle ou telle instruction selon la valeur d'une condition. Le test est suivi de **:** et les instructions dépendant de ce test sont **indentées** (*décalées vers la droite*).
- ▶ Pour contrôler conditionnellement la suite des instructions du programme, les blocs **Si/SiNon** sont un élément syntaxique de base en **Python**.
- ▶ Il existe deux variantes du bloc **Si/SiNon** selon que la condition est réalisée ou pas.

Les branchements conditionnels

- ▶ Python suppose que toute valeur **différente de zéro** ou **non nulle** est VRAIE, et si elle est soit **égale à zéro**, soit **nulle**, alors elle est supposée être FAUSSE.

Les branchements conditionnels

- ▶ **Python** suppose que toute valeur **différente de zéro** ou **non nulle** est VRAIE, et si elle est soit **égale à zéro**, soit **nulle**, alors elle est supposée être FAUSSE.
- ▶ **Python** fournit les différents types d'énoncés de prise de décision suivants :

Les branchements conditionnels

- ▶ **Python** suppose que toute valeur **différente de zéro** ou **non nulle** est VRAIE, et si elle est soit **égale à zéro**, soit **nulle**, alors elle est supposée être FAUSSE.
- ▶ **Python** fournit les différents types d'énoncés de prise de décision suivants :

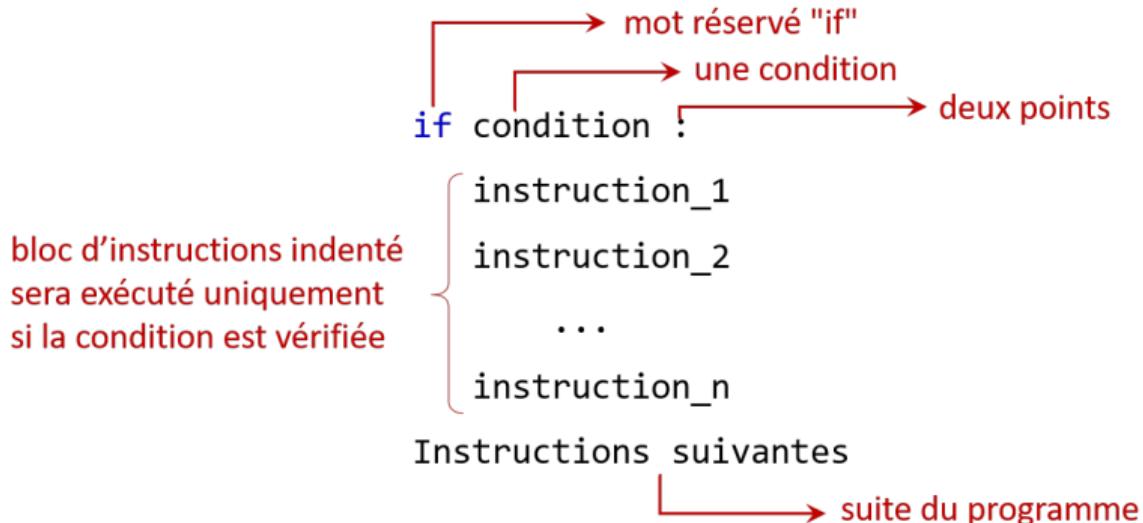
Les branchements conditionnels

- ▶ **Python** suppose que toute valeur **différente de zéro** ou **non nulle** est VRAIE, et si elle est soit **égale à zéro**, soit **nulle**, alors elle est supposée être FAUSSE.
- ▶ **Python** fournit les différents types d'énoncés de prise de décision suivants :

Type	Déclaration et description
1	si bloc d'instructions Une instruction si consiste en une expression booléenne suivie d'une instruction ou de plusieurs instructions.
2	si ... sinon bloc d'instructions. Une instruction si peut être suivie d'une déclaration optionnelle sinon , qui s'exécute lorsque l'expression booléenne est FAUSSE.
3	si emboîté bloc d'instructions On peut utiliser une instruction si ou si ... sinon à l'intérieur d'une autre instruction si ou si ... sinon

Procédures de sélection à sens unique

- La commande **if** de **Python** permet de tester le contenu d'une variable et exécute une série d'instructions uniquement lorsqu'une expression booléenne est vraie.



Procédures de sélection à deux sens

- On peut aussi exécuter des instructions si la condition n'est pas remplie à l'aide du mot **else**.

```
if condition :  
    instruction_V1  
    instruction_V2  
    ...  
    instruction_Vn  
  
else :  
    instruction_F1  
    instruction_F2  
    ...  
    instruction_Fm
```

Instructions suivantes

bloc exécuté si la condition est vérifiée

bloc exécuté si la condition n'est pas vérifiée

Procédures de sélection multiple

- ▶ Certains programmes peuvent être confrontés à plusieurs tests successifs, ce qu'on appelle aussi l'alternative multiple grâce au mot réservé **elif**.

```
if condition_1 : } bloc exécuté si la  
    bloc_instr_1 } condition 1 est vérifiée  
  
elif condition_2 : } bloc exécuté si la condition 1  
    bloc_instr_2 } est fausse et la condition 2  
                  est vérifiée  
...  
elif condition_n-1 : } bloc exécuté si la condition 1 ...  
    bloc_instr_n-1 } jusqu'à la condition n-2 sont fausses  
                  et la condition n-1 est vérifiée  
  
else condition_n : } bloc exécuté si la condition 1 ...  
    bloc_instr_n } jusqu'à la condition n-1 sont  
                  fausses et la condition n est vérifiée
```

Instructions suivantes

Table des matières

- ① Avant-propos
- ② Environnement Python
- ③ Bases d'utilisation
- ④ Outils de contrôle de flux
 - Le contrôle de conditions
 - Les structures de contrôle ou boucles
- ⑤ Les collections de données
- ⑥ Les fonctions en Python
- ⑦ Les modules/librairies en Python
- ⑧ Les fichiers entrées/sorties en Python

Traitement répétitif

- ▶ Outre l'exécution séquentielle qui caractérise le langage de programmation **Python**, certains blocs du code peuvent être exécutés plusieurs fois.

Traitement répétitif

- ▶ Outre l'exécution séquentielle qui caractérise le langage de programmation **Python**, certains blocs du code peuvent être exécutés plusieurs fois.
- ▶ Les commandes de répétition, également connues sous le nom de boucles, permettent de répéter une action plusieurs fois ; l'action est connue sous le nom de passe ou une itération.

Traitement répétitif

- ▶ Outre l'exécution séquentielle qui caractérise le langage de programmation **Python**, certains blocs du code peuvent être exécutés plusieurs fois.
- ▶ Les commandes de répétition, également connues sous le nom de boucles, permettent de répéter une action plusieurs fois ; l'action est connue sous le nom de passe ou une itération.
- ▶ **Python** offre deux types de boucles pour répondre aux besoins de bouclage :

Traitement répétitif

- ▶ Outre l'exécution séquentielle qui caractérise le langage de programmation **Python**, certains blocs du code peuvent être exécutés plusieurs fois.
- ▶ Les commandes de répétition, également connues sous le nom de boucles, permettent de répéter une action plusieurs fois ; l'action est connue sous le nom de passe ou une itération.
- ▶ **Python** offre deux types de boucles pour répondre aux besoins de bouclage :
 - * Celle qui répète une action un nombre connu de fois (itération définie) : **for**.

Traitement répétitif

- ▶ Outre l'exécution séquentielle qui caractérise le langage de programmation **Python**, certains blocs du code peuvent être exécutés plusieurs fois.
- ▶ Les commandes de répétition, également connues sous le nom de boucles, permettent de répéter une action plusieurs fois ; l'action est connue sous le nom de passe ou une itération.
- ▶ **Python** offre deux types de boucles pour répondre aux besoins de bouclage :
 - ✳ Celle qui répète une action un nombre connu de fois (itération définie) : **for**.
 - ✳ Celle qui exécute l'action jusqu'à ce que le programme détermine qu'il doit s'arrêter (itération indéfinie) : **while**.

La boucle while

- En langage **Python**, la boucle **while**, tant que en français, est une structure de contrôle permettant d'exécuter un ensemble d'instructions de façon répétée sur la base d'une condition booléenne.

La boucle while

- ▶ En langage **Python**, la boucle **while**, tant que en français, est une structure de contrôle permettant d'exécuter un ensemble d'instructions de façon répétée sur la base d'une condition booléenne.
- ▶ La boucle **while** peut être vue comme une répétition de l'instruction **if**.

La boucle while

- ▶ En langage **Python**, la boucle **while**, tant que en français, est une structure de contrôle permettant d'exécuter un ensemble d'instructions de façon répétée sur la base d'une condition booléenne.
- ▶ La boucle **while** peut être vue comme une répétition de l'instruction **if**.
- ▶ La syntaxe d'une boucle **while** est :

La boucle while

- ▶ En langage **Python**, la boucle **while**, tant que en français, est une structure de contrôle permettant d'exécuter un ensemble d'instructions de façon répétée sur la base d'une condition booléenne.
- ▶ La boucle **while** peut être vue comme une répétition de l'instruction **if**.
- ▶ La syntaxe d'une boucle **while** est :

La boucle while

- ▶ En langage **Python**, la boucle **while**, tant que en français, est une structure de contrôle permettant d'exécuter un ensemble d'instructions de façon répétée sur la base d'une condition booléenne.
- ▶ La boucle **while** peut être vue comme une répétition de l'instruction **if**.
- ▶ La syntaxe d'une boucle **while** est :

```
mot réservé "while"
while condition :      une condition
                      deux points
bloc_instr           bloc d'instructions indenté sera
                      exécuté tant que condition est vraie
```

Utilisation de else avec while

- ▶ Python permet d'associer une instruction **else** à la boucle **while**.

Utilisation de else avec while

- ▶ Python permet d'associer une instruction **else** à la boucle **while**.
- ▶ Si l'instruction **else** est utilisée avec une boucle **while**, l'instruction **else** est exécutée lorsque la condition devient fausse.

Utilisation de else avec while

- ▶ Python permet d'associer une instruction **else** à la boucle **while**.
- ▶ Si l'instruction **else** est utilisée avec une boucle **while**, l'instruction **else** est exécutée lorsque la condition devient fausse.
- ▶ La syntaxe d'une telle combinaison est :

Utilisation de else avec while

- ▶ Python permet d'associer une instruction **else** à la boucle **while**.
- ▶ Si l'instruction **else** est utilisée avec une boucle **while**, l'instruction **else** est exécutée lorsque la condition devient fausse.
- ▶ La syntaxe d'une telle combinaison est :

Utilisation de else avec while

- ▶ Python permet d'associer une instruction **else** à la boucle **while**.
- ▶ Si l'instruction **else** est utilisée avec une boucle **while**, l'instruction **else** est exécutée lorsque la condition devient fausse.
- ▶ La syntaxe d'une telle combinaison est :

```
mot réservé "while"
while condition :  
    bloc_instr_1  
else :  
    bloc_instr_2
```

une condition

deux points

bloc d'instructions indenté sera exécuté tant que condition est vraie

bloc d'instructions indenté sera exécuté si condition est fausse

La boucle for

- ▶ L'instruction **for** est utilisée lorsqu'on veut répéter une séquence d'instructions un nombre prédéfini de fois : Nombre de répétitions est connu à l'avance.

La boucle for

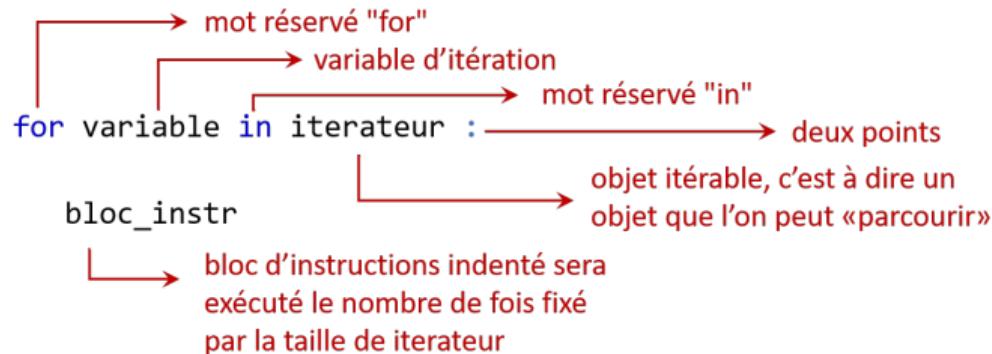
- ▶ L'instruction **for** est utilisée lorsqu'on veut répéter une séquence d'instructions un nombre prédéfini de fois : Nombre de répétitions est connu à l'avance.
- ▶ La syntaxe d'une boucle **for** est :

La boucle for

- ▶ L'instruction **for** est utilisée lorsqu'on veut répéter une séquence d'instructions un nombre prédéfini de fois : Nombre de répétitions est connu à l'avance.
- ▶ La syntaxe d'une boucle **for** est :

La boucle for

- ▶ L'instruction **for** est utilisée lorsqu'on veut répéter une séquence d'instructions un nombre prédéfini de fois : Nombre de répétitions est connu à l'avance.
- ▶ La syntaxe d'une boucle **for** est :



- ▶ Itérateur peut être **Range**, **List**, **Tuple**, **Dictionnaire**, une **Chaîne de caractères** ou un **Fichier**.

Utilisation de `else` avec `for`

- ▶ **Python** supporte l'association d'une instruction `else` au corps de la boucle `for`.

Utilisation de else avec for

- ▶ Python supporte l'association d'une instruction **else** au corps de la boucle **for**.
- ▶ Si la commande **else** est utilisée avec une boucle **for**, alors son bloc d'instructions n'est exécuté que lorsque la boucle a épuisé tous les éléments de l'itérateur. Autrement la boucle n'a pas été arrêtée.

Utilisation de else avec for

- ▶ Python supporte l'association d'une instruction **else** au corps de la boucle **for**.
- ▶ Si la commande **else** est utilisée avec une boucle **for**, alors son bloc d'instructions n'est exécuté que lorsque la boucle a épuisé tous les éléments de l'itérateur. Autrement la boucle n'a pas été arrêtée.

Utilisation de else avec for

- ▶ Python supporte l'association d'une instruction **else** au corps de la boucle **for**.
- ▶ Si la commande **else** est utilisée avec une boucle **for**, alors son bloc d'instructions n'est exécuté que lorsque la boucle a épuisé tous les éléments de l'itérateur. Autrement la boucle n'a pas été arrêtée.

```
for variable in iterateur :  
    bloc_instr_1  
else :  
    bloc_instr_2
```

mot réservé "for"
variable d'itération
mot réservé "in"
deux points
objet itérable, c'est à dire un
objet que l'on peut «parcourir»
bloc d'instructions indenté sera exécuté le
nombre de fois fixé par la taille de iterateur
bloc d'instructions indenté sera
exécuté si la boucle for prenne fin

Instructions de contrôle de boucles

- ▶ Généralement, les boucles itèrent sur un bloc de code jusqu'à ce que l'expression de test soit fausse. Toutefois, certains traitements exigent un **arrêt immédiat** de l'**itération en cours** ou même la **boucle entière** sans **aucune vérification** de l'expression de test.

Instructions de contrôle de boucles

- ▶ Généralement, les boucles itèrent sur un bloc de code jusqu'à ce que l'expression de test soit fausse. Toutefois, certains traitements exigent un **arrêt immédiat** de l'**itération en cours** ou même la **boucle entière** sans **aucune vérification** de l'expression de test.
- ▶ Les instructions de **contrôle de boucle** permettent de **modifier** l'ordre **normal** d'exécution de la séquence.

Instructions de contrôle de boucles

- ▶ Généralement, les boucles itèrent sur un bloc de code jusqu'à ce que l'expression de test soit fausse. Toutefois, certains traitements exigent un **arrêt immédiat** de l'**itération en cours** ou même la **boucle entière** sans **aucune vérification** de l'expression de test.
- ▶ Les instructions de **contrôle de boucle** permettent de **modifier** l'ordre **normal** d'exécution de la séquence.
- ▶ **Python** prend en charge les instructions suivantes :

Instructions de contrôle de boucles

- ▶ Généralement, les boucles itèrent sur un bloc de code jusqu'à ce que l'expression de test soit fausse. Toutefois, certains traitement exige un **arrêt immédiat** de **l'itération en cours** ou même la **boucle entière** sans **aucune vérification** de l'expression de test.
- ▶ Les instructions de **contrôle de boucle** permettent de **modifier** l'ordre **normal** d'exécution de la séquence.
- ▶ **Python** prend en charge les instructions suivantes :
 - * Continue,

Instructions de contrôle de boucles

- ▶ Généralement, les boucles itèrent sur un bloc de code jusqu'à ce que l'expression de test soit fausse. Toutefois, certains traitements exigent un **arrêt immédiat** de l'**itération en cours** ou même la **boucle entière** sans **aucune vérification** de l'expression de test.
- ▶ Les instructions de **contrôle de boucle** permettent de **modifier** l'ordre **normal** d'exécution de la séquence.
- ▶ **Python** prend en charge les instructions suivantes :
 - * Continue,
 - * Break,

Instruction continue

- ▶ L'instruction **continue** **ignore** toutes les instructions restantes dans l'**itération actuelle** de la boucle et **ramène** le contrôle au **début** de la boucle.

Instruction continue

- ▶ L'instruction **continue** **ignore** toutes les instructions restantes dans l'**itération actuelle** de la boucle et **ramène** le contrôle au **début** de la boucle.
- ▶ L'utilisation efficace et efficiente de l'instruction **continue** exige son appartenance au corps d'un branchement conditionnel **if ... else**.

Instruction continue

- ▶ L'instruction **continue** **ignore** toutes les instructions restantes dans l'**itération actuelle** de la boucle et **ramène** le contrôle au **début** de la boucle.
- ▶ L'utilisation efficace et efficiente de l'instruction **continue** exige son appartenance au corps d'un branchement conditionnel **if ... else**.

Instruction continue

- ▶ L'instruction `continue` **ignore** toutes les instructions restantes dans l'**itération actuelle** de la boucle et **ramène** le contrôle au **début** de la boucle.
- ▶ L'utilisation efficace et efficiente de l'instruction `continue` exige son appartenance au corps d'un branchement conditionnel `if ... else`.

```
for variable in iterateur :  
    if condition :  
        instructions_V  
        continue  
    bloc_instr
```

- ▶ L'instruction `continue` peut être utilisée dans les deux boucles `while` et `for`.

Instruction break

- ▶ L'instruction **break** **met fin** à une boucle en **cours** et **reprend** l'exécution à l'instruction **juste après** la **fin** du corps de la **boucle**, tout comme l'instruction **break** traditionnelle en C.

Instruction break

- ▶ L'instruction **break** **met fin** à une boucle en **cours** et **reprend** l'exécution à l'instruction **juste après** la **fin** du corps de la **boucle**, tout comme l'instruction **break** traditionnelle en C.
- ▶ **break** arrête une boucle **même** si la **condition** de **while** est **vraie** ou le **contenu** de l'itérateur de **for** n'est pas encore **épuisé**.

Instruction break

- ▶ L'instruction **break** met fin à une boucle en **cours** et reprend l'exécution à l'instruction **juste après** la **fin** du corps de la **boucle**, tout comme l'instruction **break** traditionnelle en C.
- ▶ **break** arrête une boucle **même** si la **condition** de **while** est **vraie** ou le **contenu** de l'itérateur de **for** n'est pas encore **épuisé**.
- ▶ L'utilisation la plus courante de **break** est lorsqu'une condition externe est déclenchée, nécessitant une sortie rapide de la boucle. L'instruction **break** peut être utilisée dans les deux boucles **while** et **for**.

Instruction break

- ▶ L'instruction **break** met fin à une boucle en **cours** et reprend l'exécution à l'instruction **juste après** la **fin** du corps de la **boucle**, tout comme l'instruction **break** traditionnelle en C.
- ▶ **break** arrête une boucle **même** si la **condition** de **while** est **vraie** ou le **contenu** de l'itérateur de **for** n'est pas encore **épuisé**.
- ▶ L'utilisation la plus courante de **break** est lorsqu'une condition externe est déclenchée, nécessitant une sortie rapide de la boucle. L'instruction **break** peut être utilisée dans les deux boucles **while** et **for**.

Instruction break

- ▶ L'instruction **break** met fin à une boucle en **cours** et **reprend** l'exécution à l'instruction **juste après** la **fin** du corps de la **boucle**, tout comme l'instruction **break** traditionnelle en C.
- ▶ **break** arrête une boucle **même** si la **condition** de **while** est **vraie** ou le **contenu** de l'iterator de **for** n'est pas encore **épuisé**.
- ▶ L'utilisation la plus courante de **break** est lorsqu'une condition externe est déclenchée, nécessitant une sortie rapide de la boucle. L'instruction **break** peut être utilisée dans les deux boucles **while** et **for**.

```
while conditionW :  
    if condition :  
        instructions_V  
        break  
  
    bloc_instr
```

Table des matières

- ① Avant-propos
- ② Environnement Python
- ③ Bases d'utilisation
- ④ Outils de contrôle de flux
- ⑤ **Les collections de données**
 - Les listes en Python
 - Les tuples en Python
 - Les dictionnaires en Python
- ⑥ Les fonctions en Python
- ⑦ Les modules/librairies en Python
- ⑧ Les fichiers entrées/sorties en Python

Types de collecte de données

- ▶ **Python** offre une gamme de 4 types de données composées, appelées séquences :

Types de collecte de données

▶ **Python** offre une gamme de 4 types de données composées, appelées séquences :

List est une collection qui est ordonnée et modifiable. Elle permet de dupliquer les membres.

Types de collecte de données

▶ **Python** offre une gamme de 4 types de données composées, appelées séquences :

List est une collection qui est ordonnée et modifiable. Elle permet de dupliquer les membres.

Tuple est une collection qui est ordonnée et non modifiable. Permet de dupliquer des membres.

Types de collecte de données

▶ **Python** offre une gamme de 4 types de données composées, appelées séquences :

List est une collection qui est ordonnée et modifiable. Elle permet de dupliquer les membres.

Tuple est une collection qui est ordonnée et non modifiable. Permet de dupliquer des membres.

Set est une collection qui n'est ni ordonnée ni indexée. Pas de membres en double.

Types de collecte de données

▶ **Python** offre une gamme de 4 types de données composées, appelées séquences :

List est une collection qui est ordonnée et modifiable. Elle permet de dupliquer les membres.

Tuple est une collection qui est ordonnée et non modifiable. Permet de dupliquer des membres.

Set est une collection qui n'est ni ordonnée ni indexée. Pas de membres en double.

Dictionary est une collection non ordonnée, modifiable et indexée. Pas de doublons.

Types de collecte de données

▶ **Python** offre une gamme de 4 types de données composées, appelées séquences :

List est une collection qui est ordonnée et modifiable. Elle permet de dupliquer les membres.

Tuple est une collection qui est ordonnée et non modifiable. Permet de dupliquer des membres.

Set est une collection qui n'est ni ordonnée ni indexée. Pas de membres en double.

Dictionary est une collection non ordonnée, modifiable et indexée. Pas de doublons.

▶ Le choix du bon type pour un ensemble de données particulier pourrait contribuer à l'augmentation de l'efficacité du traitement ou de la sécurité des données.

Table des matières

- ① Avant-propos
- ② Environnement Python
- ③ Bases d'utilisation
- ④ Outils de contrôle de flux
- ⑤ Les collections de données
 - Les listes en Python
 - Les tuples en Python
 - Les dictionnaires en Python
- ⑥ Les fonctions en Python
- ⑦ Les modules/librairies en Python
- ⑧ Les fichiers entrées/sorties en Python

Listes

- ▶ Une liste est une structure de données qui contient une série d'éléments éventuellement hétérogènes ; "de types différents".

Listes

- ▶ Une liste est une structure de données qui contient une série d'éléments éventuellement hétérogènes ; "de types différents".
- ▶ **Python** permet la déclaration d'une liste par une série de valeurs séparées par des virgules, et le tout encadré par des crochets.

Listes

- ▶ Une liste est une structure de données qui contient une série d'éléments éventuellement hétérogènes ; "de types différents".
- ▶ **Python** permet la déclaration d'une liste par une série de valeurs séparées par des virgules, et le tout encadré par des crochets.

Listes

- ▶ Une liste est une structure de données qui contient une série d'éléments éventuellement hétérogènes ; "de types différents".
- ▶ Python permet la déclaration d'une liste par une série de valeurs séparées par des virgules, et le tout encadré par des crochets.

```
»» Liste1 = ["Python", "langage", 1996, 2020]
```

```
»» print(Liste1)
```

```
['Python', 'langage', 1996, 2020]
```

```
»» Liste2 = [-12, 6, 1996, 2020]
```

```
»» Liste3 = ["a", "u", "r", "g", "s", "m"]
```

```
»» Liste4 = [2, "z", 4+3j, [2.56, "w", 8j]]
```

Manipulation des Listes

- ▶ Accès aux éléments :

Manipulation des Listes

▶ Accès aux éléments :

- * Pour accéder aux valeurs des listes, il faut utiliser les crochets plus l'indice ou les indices, commençant par l'indice "0", pour obtenir la valeur qu'il s'y trouve.

Manipulation des Listes

▶ Accès aux éléments :

- * Pour accéder aux valeurs des listes, il faut utiliser les crochets plus l'indice ou les indices, commençant par l'indice "0", pour obtenir la valeur qu'il s'y trouve.

Manipulation des Listes

▶ Accès aux éléments :

- Pour accéder aux valeurs des listes, il faut utiliser les crochets plus l'indice ou les indices, commençant par l'indice "0", pour obtenir la valeur qu'il s'y trouve.

```
»» Liste1 = ["Python", "langage", 1996, 2020]
```

```
»» Liste4 = [2, "z", 4+3j, [2.56, "w", 8j]]
```

```
»» print("Liste1[0]: ", Liste1[0])
```

Liste1[0]: Python

```
»» print("Liste4[3]: ", Liste4[3])
```

Liste4[3]: [2.56, 'w', 8j]

Manipulation des Listes

- ▶ Découpage en tranche :

Manipulation des Listes

▶ Découpage en tranche :

- * Comme les chaînes de caractères, les listes peuvent être indexées et découpées :

Manipulation des Listes

▶ Découpage en tranche :

- * Comme les chaînes de caractères, les listes peuvent être indexées et découpées :

Manipulation des Listes

▶ Découpage en tranche :

- * Comme les chaînes de caractères, les listes peuvent être indexées et découpées :

» carree = [1, 4, 9, 16, 25, 36, 49, 64]

» print(carree[3:8])

[16, 25, 36, 49]

» print(carree[2:])

[9, 16, 25, 36, 49, 64]

» print(carree[:5])

[1, 4, 9, 16, 25]

Manipulation des Listes

▶ Suppression :

Manipulation des Listes

▶ Suppression :

- ✿ On peut supprimer une entrée de la liste en référençant son index (`del`) ou bien son contenu (`remove`):

Manipulation des Listes

▶ Suppression :

- ✿ On peut supprimer une entrée de la liste en référençant son index (`del`) ou bien son contenu (`remove`):

Manipulation des Listes

▶ Suppression :

- ✿ On peut supprimer une entrée de la liste en référençant son index (`del`) ou bien son contenu (`remove`):

```
»»> presents = ["Ali", "Mohammed", "Anas", "Wiam", "Yosra", "Majd"]
```

```
»»> del(presents[0])
```

```
»»> print(presents)
```

```
['Mohammed', 'Anas', 'Wiam', 'Yosra', 'Majd']
```

```
»»> presents = ["Ali", "Mohammed", "Anas", "Wiam", "Yosra", "Majd"]
```

```
»»> print(presents.remove('Yosra'))
```

```
['Ali', 'Mohammed', 'Anas', 'Wiam', 'Majd']
```

Manipulation des Listes

▶ Ajout :

Manipulation des Listes

▶ Ajout :

- * On peut ajouter un élément à la fin d'une liste en utilisant la méthode ([append](#)) :

Manipulation des Listes

▶ Ajout :

- * On peut ajouter un élément à la fin d'une liste en utilisant la méthode ([append](#)) :

Manipulation des Listes

▶ Ajout :

- * On peut ajouter un élément à la fin d'une liste en utilisant la méthode (`append`) :
 - »» Age = [54, 89, 14, 25, 64, 71]
 - »» Age.append("Omar")
 - »» `print(Age)`
 - [54, 89, 14, 25, 64, 71, 'Omar']
- * La méthode (`insert`) permet d'insérer une entrée à une position donnée dans la liste :

Manipulation des Listes

▶ Ajout :

- ✿ On peut ajouter un élément à la fin d'une liste en utilisant la méthode (`append`) :
 - »» Age = [54, 89, 14, 25, 64, 71]
 - »» Age.append("Omar")
 - »» `print(Age)`
 - [54, 89, 14, 25, 64, 71, 'Omar']
- ✿ La méthode (`insert`) permet d'insérer une entrée à une position donnée dans la liste :

Manipulation des Listes

▶ Ajout :

- * On peut ajouter un élément à la fin d'une liste en utilisant la méthode (`append`) :

»» Age = [54, 89, 14, 25, 64, 71]

»» Age.append("Omar")

»» `print(Age)`

[54, 89, 14, 25, 64, 71, 'Omar']

- * La méthode (`insert`) permet d'insérer une entrée à une position donnée dans la liste :

»» Age = [54, 89, 14, 25, 64, 71]

»» Age.insert(2,"Omar")

»» `print(Age)`

[54, 89, 'Omar', 14, 25, 64, 71]

Manipulation des Listes

- ▶ Modification de la valeur d'un élément et index d'un élément :

Manipulation des Listes

▶ Modification de la valeur d'un élément et index d'un élément :

- ✳ Les données dans une liste sont mutables, donc on peut changer leurs valeurs :

Manipulation des Listes

▶ Modification de la valeur d'un élément et index d'un élément :

- ✳ Les données dans une liste sont mutables, donc on peut changer leurs valeurs :

Manipulation des Listes

▶ Modification de la valeur d'un élément et index d'un élément :

- ✿ Les données dans une liste sont mutables, donc on peut changer leurs valeurs :

```
» Infos = ["Licence", "IRM", 2015, 2021]
```

```
» Infos[1] = 5
```

```
» print(Infos)
```

```
['Licence', 5, 2015, 2021]
```

- ✿ On peut trouver l'index d'un élément par la méthode (`index`) :

Manipulation des Listes

▶ Modification de la valeur d'un élément et index d'un élément :

- ✿ Les données dans une liste sont mutables, donc on peut changer leurs valeurs :

```
» Infos = ["Licence", "IRM", 2015, 2021]
```

```
» Infos[1] = 5
```

```
» print(Infos)
```

```
['Licence', 5, 2015, 2021]
```

- ✿ On peut trouver l'index d'un élément par la méthode (`index`) :

Manipulation des Listes

▶ Modification de la valeur d'un élément et index d'un élément :

- ✿ Les données dans une liste sont mutables, donc on peut changer leurs valeurs :

```
» Infos = ["Licence", "IRM", 2015, 2021]
```

```
» Infos[1] = 5
```

```
» print(Infos)
```

```
['Licence', 5, 2015, 2021]
```

- ✿ On peut trouver l'index d'un élément par la méthode (`index`) :

```
» print(Infos.index(2021))
```

3

Manipulation des Listes

- ▶ Itération à travers une liste :

Manipulation des Listes

► Itération à travers une liste :

- ★ En utilisant une boucle **for**, on peut parcourir tous les éléments d'une liste un à un :

Manipulation des Listes

► Itération à travers une liste :

- ★ En utilisant une boucle **for**, on peut parcourir tous les éléments d'une liste un à un :

Manipulation des Listes

► Itération à travers une liste :

- * En utilisant une boucle **for**, on peut parcourir tous les éléments d'une liste un à un :

»» Couleurs = ["Rouge", "Bleu", "Vert", "Magenta", "Cyan", "Yellow"]

»» **for** cont **in** Couleurs :

```
    print("La couleur :", cont)
```

La couleur : Rouge

La couleur : Blue

La couleur : Vert

La couleur : Magenta

La couleur : Cyan

La couleur : Yellow

Fonctions et méthodes intégrées

Soit une liste notée **List**, ci-après d'autres **fonctions** pour mieux la manipuler :

Fonction	Description
min(List)	Valeur minimum dans la liste
max(List)	valeur maximum dans la liste
sum(List)	somme des éléments de la liste
len(List)	Renvoie le nombre d'éléments de la liste
cmp(List1, List2)	Compare les éléments des deux listes.
list(seq)	Génère une liste à partir des éléments de seq
enumerate(List)	Permet de récupérer la position et la valeur correspondante lorsqu'on itère sur une liste

Fonctions et méthodes intégrées

Soit une liste notée **L**, ci-après d'autres **méthodes** pour mieux la manipuler :

Méthode	Description
<code>L.extend(T)</code>	Ajoute les éléments de T à la fin de L (équivalent à <code>L+=T</code>)
<code>L.pop()</code>	Supprime le dernier élément de L, et le renvoie
<code>L.pop(i)</code>	Supprime l'élément d'indice i de L, en décalant les suivants vers la gauche.
<code>L.index(x)</code>	Retourne l'indice de la première occurrence de x dans L si x est présent
<code>L.count(x)</code>	Retourne le nombre d'occurrences de x dans L
<code>L.sort()</code>	Trie la liste L dans l'ordre croissant (en place)
<code>L.reverse()</code>	Renverse la liste (en place)
<code>L.clear()</code>	Supprime tous les éléments de la liste. Équivalent à <code>del L[:]</code>
<code>L.copy()</code>	Renvoie une copie de la liste

Listes de listes

- ▶ On utilisera couramment des listes qui contiennent des listes, en particulier pour représenter des matrices.

Listes de listes

- ▶ On utilisera couramment des listes qui contiennent des listes, en particulier pour représenter des matrices.

- ▶ La matrice $B = \begin{pmatrix} 20 & -5 & 11 \\ 9 & 12 & 7 \\ 8 & 14 & 2 \end{pmatrix}$, peut être déclarée en python par :

»» B = $\left[[20, -5, 11], [8, 12, 7], [8, 14, 2] \right]$

Listes de listes

- ▶ On utilisera couramment des listes qui contiennent des listes, en particulier pour représenter des matrices.

- ▶ La matrice $B = \begin{pmatrix} 20 & -5 & 11 \\ 9 & 12 & 7 \\ 8 & 14 & 2 \end{pmatrix}$, peut être déclarée en python par :

»» B = [[20, -5, 11], [8, 12, 7], [8, 14, 2]]

- ▶ On parle alors de listes en deux dimensions, dont l'accès à l'élément i de la liste se fait alors par la syntaxe $B[i]$.

»»**print** B[0]

[20, -5, 11]

»»**print** B[1]

[9, 12, 7]

»»**print** B[2]

[8, 14, 2]

Listes de listes

► Le même résultats peut être obtenu en faisant recours au traitement répétitif.

```
»»> for i in range(3) :  
    print(B[i])
```

[20, -5, 11]

[9, 12, 7]

[8, 14, 2]

Listes de listes

- Le même résultats peut être obtenu en faisant recours au traitement répétitif.

```
»» for i in range(3) :  
    print(B[i])
```

[20, -5, 11]
[9, 12, 7]
[8, 14, 2]

- L'accès à l'élément j de la i-ième liste se fait alors par la syntaxe B[i][j].

```
»» print B[2][0]
```

8

```
»» print B[1][1]
```

12

```
»» print B[1][2]
```

7

Listes de listes

- ▶ Via un traitement répétitif imbriqué, on peut accéder **successivement** à tous les éléments de la matrice B .

```
»»> for i in range(3) :  
    for j in range(3) :  
        print(B[i][j])
```

20

-5

11

9

12

7

8

14

2

Listes en compréhensions (intentions)

- ▶ La construction des listes en **python** fait souvent appel à des boucles, ce qui amène à des syntaxes un peu lourdes.



Listes en compréhensions (intentions)

- ▶ La construction des listes en **python** fait souvent appel à des boucles, ce qui amène à des syntaxes un peu lourdes.
- ▶ La compréhension de liste, ou les listes en intentions, est un mécanisme très compacte et très puissant de construction des listes en **python**.

Listes en compréhensions (intentions)

- ▶ La construction des listes en **python** fait souvent appel à des boucles, ce qui amène à des syntaxes un peu lourdes.
- ▶ La compréhension de liste, ou les listes en intentions, est un mécanisme très compacte et très puissant de construction des listes en **python**.
- ▶ Elle consiste à écrire la boucle dans la liste elle-même selon la syntaxe suivante :

Listes en compréhensions (intentions)

- ▶ La construction des listes en **python** fait souvent appel à des boucles, ce qui amène à des syntaxes un peu lourdes.
- ▶ La compréhension de liste, ou les listes en intentions, est un mécanisme très compacte et très puissant de construction des listes en **python**.
- ▶ Elle consiste à écrire la boucle dans la liste elle-même selon la syntaxe suivante :

Listes en compréhensions (intentions)

- ▶ La construction des listes en **python** fait souvent appel à des boucles, ce qui amène à des syntaxes un peu lourdes.
- ▶ La compréhension de liste, ou les listes en intentions, est un mécanisme très compacte et très puissant de construction des listes en **python**.
- ▶ Elle consiste à écrire la boucle dans la liste elle-même selon la syntaxe suivante :

```
liste = [ expression_sur_elm for elm in collection if condition_sur_elm ]
```

Est une expression qui dépend en général de *elm* et dont la valeur est placée dans *liste*

Elément courant qui parcourt la liste *collection*
Appelé aussi variable de contrôle de la liste en compréhension

Condition sur les éléments de la liste *collection*

Assez souvent une liste

La paire de crochets, les mots-clés **for** et **in** sont obligatoires

Listes en compréhensions (intentions)

- ▶ La construction d'une liste des 5 premiers cubes parfaits par ajout et en utilisant la notion de liste en intention.

Listes en compréhensions (intentions)

- ▶ La construction d'une liste des 5 premiers cubes parfaits par ajout et en utilisant la notion de liste en intention.
 - ✳ Par append :

Listes en compréhensions (intentions)

- ▶ La construction d'une liste des 5 premiers cubes parfaits par ajout et en utilisant la notion de liste en intention.
 - ✳ Par append :

Listes en compréhensions (intentions)

- ▶ La construction d'une liste des 5 premiers cubes parfaits par ajout et en utilisant la notion de liste en intention.

- ✳ Par append :

```
»> Cinq_cub = list()  
»> for elemt in range(5) :  
    Cinq_cub.append(elemt**3)  
»> print(Cinq_cub)
```

```
[0, 1, 8, 27, 64]
```

- ✳ Par intentions :

Listes en compréhensions (intentions)

- ▶ La construction d'une liste des 5 premiers cubes parfaits par ajout et en utilisant la notion de liste en intention.

- ✳ Par append :

```
»> Cinq_cub = list()  
»> for elemt in range(5) :  
    Cinq_cub.append(elemt**3)  
»> print(Cinq_cub)
```

```
[0, 1, 8, 27, 64]
```

- ✳ Par intentions :



Listes en compréhensions (intentions)

- ▶ La construction d'une liste des 5 premiers cubes parfaits par ajout et en utilisant la notion de liste en intention.

- ✳ Par append :

```
»> Cinq_cub = list()  
»> for elemt in range(5) :  
    Cinq_cub.append(elemt**3)  
»> print(Cinq_cub)
```

```
[0, 1, 8, 27, 64]
```

- ✳ Par intentions :

```
»> Cinq_cub = [elemt ** 3 for elemt in range(5)]  
»> print(Cinq_cub)
```

```
[0, 1, 8, 27, 64]
```

Listes en compréhensions (intentions)

- ▶ Construction d'une liste en intention des entiers inférieurs à 10 et non divisibles par 3.

```
»> List_cond = [nbre for nbre in range(10) if nbre % 3 != 0]
```

```
»> print(Liste)
```

```
[1, 2, 4, 5, 7, 8]
```

Listes en compréhensions (intentions)

- ▶ Construction d'une liste en intention des entiers inférieurs à 10 et non divisibles par 3.

```
»> List_cond = [nbre for nbre in range(10) if nbre % 3 != 0]
»> print(Cinq_cub)
[1, 2, 4, 5, 7, 8]
```

- ▶ Construction d'une liste des caractères des mots dans une autre liste.

```
»> liste_init = ["hello", "the", "world"]
»> nouvelle_liste = [c for mot in liste_init for c in mot]
»> print(nouvelle_liste)
['h', 'e', 'l', 'l', 'o', 't', 'h', 'e', 'w', 'o', 'r', 'l', 'd']
```

Table des matières

- ① Avant-propos
- ② Environnement Python
- ③ Bases d'utilisation
- ④ Outils de contrôle de flux
- ⑤ **Les collections de données**
 - Les listes en Python
 - Les tuples en Python
 - Les dictionnaires en Python
- ⑥ Les fonctions en Python
- ⑦ Les modules/librairies en Python
- ⑧ Les fichiers entrées/sorties en Python

Tuples

- ▶ Un tuple ressemble beaucoup à une liste, mais on ne peut ni modifier ses éléments, ni lui en ajouter ou en enlever.

Tuples

- ▶ Un tuple ressemble beaucoup à une liste, mais on ne peut ni modifier ses éléments, ni lui en ajouter ou en enlever.
- ▶ On parle de structure immuable (ou statique, ou encore non mutable).

Tuples

- ▶ Un tuple ressemble beaucoup à une liste, mais on ne peut ni modifier ses éléments, ni lui en ajouter ou en enlever.
- ▶ On parle de structure immuable (ou statique, ou encore non mutable).
- ▶ En contrepartie de cette rigidité, les tuples sont très compacts (ils occupent peu de mémoire) et l'accès à leurs éléments est rapide.

Tuples

- ▶ Un tuple ressemble beaucoup à une liste, mais on ne peut ni modifier ses éléments, ni lui en ajouter ou en enlever.
- ▶ On parle de structure immuable (ou statique, ou encore non mutable).
- ▶ En contrepartie de cette rigidité, les tuples sont très compacts (ils occupent peu de mémoire) et l'accès à leurs éléments est rapide.
- ▶ Pour la syntaxe, on crée un tuple en écrivant ses éléments, séparés par des virgules et encadrés par des parenthèses.

Tuples

- ▶ Un tuple ressemble beaucoup à une liste, mais on ne peut ni modifier ses éléments, ni lui en ajouter ou en enlever.
- ▶ On parle de structure immuable (ou statique, ou encore non mutable).
- ▶ En contrepartie de cette rigidité, les tuples sont très compacts (ils occupent peu de mémoire) et l'accès à leurs éléments est rapide.
- ▶ Pour la syntaxe, on crée un tuple en écrivant ses éléments, séparés par des virgules et encadrés par des parenthèses.
- ▶ S'il n'y a pas d'ambiguïté, les parenthèses peuvent être omises (en pratique, dès que le nombre d'éléments du tuple est au moins 2).

Tuples

- ▶ Un tuple ressemble beaucoup à une liste, mais on ne peut ni modifier ses éléments, ni lui en ajouter ou en enlever.
- ▶ On parle de structure immuable (ou statique, ou encore non mutable).
- ▶ En contrepartie de cette rigidité, les tuples sont très compacts (ils occupent peu de mémoire) et l'accès à leurs éléments est rapide.
- ▶ Pour la syntaxe, on crée un tuple en écrivant ses éléments, séparés par des virgules et encadrés par des parenthèses.
- ▶ S'il n'y a pas d'ambiguïté, les parenthèses peuvent être omises (en pratique, dès que le nombre d'éléments du tuple est au moins 2).
- ▶ Un tuple constitué d'un seul élément a doit être écrit **a**, ou **(a,)**. Le tuple sans élément se note **()**.

Opérations sur les tuples

- ▶ On accède aux valeurs d'un tuple, en utilisant les crochets et les indices des valeurs souhaitées.

```
»> Tupl1 = "Arabe", "Français", "Anglais", "Allemand"
```

```
»> print("Tupl1: ", Tupl1)
```

```
Tupl1: ('Arabe', 'Français', 'Anglais', 'Allemand')
```

```
»> print("Tupl1[1:2]: ", Tupl1[1:2])
```

```
Tupl1[1:2]: ('Français',)
```

Opérations sur les tuples

- ▶ Les tuples sont immuables, on ne peut pas modifier les valeurs de leurs éléments.

» `Tupl1[1:2] = "Chinois"`

TypeError

Traceback (most recent call last)

```
<ipython-input-7-3b544f54ff3a> in <module>
```

```
---> 1 Tupl1[1:2] = 'Chinois'
```

```
TypeError: 'tuple' object does not support item assignment
```

Opérations sur les tuples

- Il n'est pas possible de retirer les différents éléments d'un tuple. Cependant, on peut supprimer le tuple complètement par la fonction `del`.

```
»> Ce_tuple = ("Pomme", "Banane", "Cerise")
»> del Ce_tuple
»> print(Ce_tuple)    # Ceci entraînera une erreur car le tuple n'existe plus
```

NameError

<ipython-input-10-ec3397ca843a> in <module>

```
    1 Ce_tuple = ("Pomme", "Banane", "Cerise")
    2 del Ce_tuple
--> 3 print(Ce_tuple)
```

NameError: name 'Ce_tuple' is not defined

Traceback (most recent call last)

Table des matières

- ① Avant-propos
- ② Environnement Python
- ③ Bases d'utilisation
- ④ Outils de contrôle de flux
- ⑤ **Les collections de données**
 - Les listes en Python
 - Les tuples en Python
 - Les dictionnaires en Python
- ⑥ Les fonctions en Python
- ⑦ Les modules/librairies en Python
- ⑧ Les fichiers entrées/sorties en Python

Dictionnaires

- Le dictionnaire en **python** est une collection non ordonnée d'articles. Chaque élément d'un dictionnaire possède une paire (*clé : valeur*).

Dictionnaires

- ▶ Le dictionnaire en **python** est une collection non ordonnée d'articles. Chaque élément d'un dictionnaire possède une paire (*clé : valeur*).
- ▶ La création d'un dictionnaire passe par le placement de ses éléments à l'intérieur d'accolades (**{}**) séparées par des virgules.

Dictionnaires

- ▶ Le dictionnaire en **python** est une collection non ordonnée d'articles. Chaque élément d'un dictionnaire possède une paire (*clé : valeur*).
- ▶ La création d'un dictionnaire passe par le placement de ses éléments à l'intérieur d'accolades (**{}**) séparées par des virgules.

- ▶ Le dictionnaire en **python** est une collection non ordonnée d'articles. Chaque élément d'un dictionnaire possède une paire (*clé : valeur*).
- ▶ La création d'un dictionnaire passe par le placement de ses éléments à l'intérieur d'accolades (`{}`) séparées par des virgules.

Remarques

- * Les valeurs peuvent être de n'importe quel type de données et peuvent se répéter.

- ▶ Le dictionnaire en **python** est une collection non ordonnée d'articles. Chaque élément d'un dictionnaire possède une paire (*clé : valeur*).
- ▶ La création d'un dictionnaire passe par le placement de ses éléments à l'intérieur d'accolades (`{}`) séparées par des virgules.

Remarques

- * Les valeurs peuvent être de n'importe quel type de données et peuvent se répéter.
- * Les clés doivent être de type immuable (chaîne, nombre ou n-uplet avec éléments immuables) et doivent être uniques.

- ▶ Le dictionnaire en **python** est une collection non ordonnée d'articles. Chaque élément d'un dictionnaire possède une paire (*clé : valeur*).
- ▶ La création d'un dictionnaire passe par le placement de ses éléments à l'intérieur d'accolades (`{}`) séparées par des virgules.

Remarques

- * Les valeurs peuvent être de n'importe quel type de données et peuvent se répéter.
- * Les clés doivent être de type immuable (chaîne, nombre ou n-uplet avec éléments immuables) et doivent être uniques.
- ▶ Les dictionnaires n'obéissent pas à la notion d'ordre (i.e. pas d'indice), d'où l'accès à ses valeurs se fait par le biais des clés correspondantes.

- ▶ Le dictionnaire en **python** est une collection non ordonnée d'articles. Chaque élément d'un dictionnaire possède une paire (*clé : valeur*).
- ▶ La création d'un dictionnaire passe par le placement de ses éléments à l'intérieur d'accolades (`{}`) séparées par des virgules.

Remarques

- * Les valeurs peuvent être de n'importe quel type de données et peuvent se répéter.
- * Les clés doivent être de type immuable (chaîne, nombre ou n-uplet avec éléments immuables) et doivent être uniques.
- ▶ Les dictionnaires n'obéissent pas à la notion d'ordre (i.e. pas d'indice), d'où l'accès à ses valeurs se fait par le biais des clés correspondantes.

- ▶ Le dictionnaire en **python** est une collection non ordonnée d'articles. Chaque élément d'un dictionnaire possède une paire (*clé : valeur*).
- ▶ La création d'un dictionnaire passe par le placement de ses éléments à l'intérieur d'accolades (`{}`) séparées par des virgules.

Remarques

- * Les valeurs peuvent être de n'importe quel type de données et peuvent se répéter.
- * Les clés doivent être de type immuable (chaîne, nombre ou n-uplet avec éléments immuables) et doivent être uniques.
- ▶ Les dictionnaires n'obéissent pas à la notion d'ordre (i.e. pas d'indice), d'où l'accès à ses valeurs se fait par le biais des clés correspondantes.

Création de dictionnaires

► On peu créer et afficher un dictionnaire par initialisation puis affectation :

» animal = { }

» animal["nom"] = "girafe"

» animal["taille"] = 5.0

» animal["poids"] = 1100

» print(animal)

{'nom': 'girafe', 'taille': 5.0, 'poids': 1100}

Création de dictionnaires

► On peut créer et afficher un dictionnaire par initialisation puis affectation :

```
»» animal = {}  
»» animal["nom"] = "girafe"  
»» animal["taille"] = 5.0  
»» animal["poids"] = 1100  
»» print(animal)  
{'nom': 'girafe', 'taille': 5.0, 'poids': 1100}
```

► On peut aussi initialiser toutes les clés et les valeurs d'un dictionnaire en une seule opération :

```
»» Ouvrier = {"nom": "Mohammed", "prenom": "Jellouli", "age": 48}  
»» print(Ouvrier)  
{'nom': 'Mohammed', 'prenom': 'Jellouli', 'age': 48}
```

Opérations sur un dictionnaire

▶ Accès aux éléments :

```
»> Ouvrier = {"nom" :"Mohammed", "prenom" :"Jellouli", "Age" : 48}
```

```
»> print("Ouvrier['nom']: ", Ouvrier['Nom'])
```

```
»> print("Ouvrier['Age']: ", Ouvrier['age'])
```

```
Ouvrier['Nom']: Mohammed
```

```
Ouvrier['Age']: 48
```

Opérations sur un dictionnaire

▶ Accès aux éléments :

```
»> Ouvrier = {"nom" :"Mohammed", "prenom" :"Jellouli", "Age" :48}
```

```
»> print("Ouvrier['nom']: ", Ouvrier['Nom'])
```

```
»> print("Ouvrier['Age']: ", Ouvrier['age'])
```

Ouvrier['Nom']: Mohammed

Ouvrier['Age']: 48

▶ Longueur du dictionnaire, fonction `len` :

```
»> eleve = {"nom" :"Mohammed", "prenom" :"Jellouli", "niveau" :"Bac" }
```

```
»> print("Le dictionnaire contient ", (len(eleve)), "éléments")
```

Le dictionnaire contient 3 éléments

Modification et ajout d'éléments à un dictionnaire

- Le contenu d'un dictionnaire peut être modifié en remplaçant la valeur associée à une clé par une autre valeur. On utilise pour cela la syntaxe d'affectation :

dictionnaire[clé]=nouvelle_valeur

Remarque : Si clé existe, sa valeur sera modifiée en nouvelle_valeur. Si clé n'existe pas, un nouvel élément est créé dans le dictionnaire.

- Modification des éléments :

»» eleve = {"nom":"Jellouli","prenom":"Mohammed","niveau":"Bac"}

»» eleve[prenom] = "Ali"

»» print(eleve)

{"nom":"Jellouli","prenom":"Ali","niveau":"Bac"}

Modification et ajout d'éléments à un dictionnaire

- Le contenu d'un dictionnaire peut être modifié en remplaçant la valeur associée à une clé par une autre valeur. On utilise pour cela la syntaxe d'affectation :

dictionnaire[clé]=nouvelle_valeur

Remarque : Si clé existe, sa valeur sera modifiée en nouvelle_valeur. Si clé n'existe pas, un nouvel élément est créé dans le dictionnaire.

★ Ajout des éléments :

```
>>> dico_es_fr = {"si":"oui", "hoy":"aujourd'hui", "porque":"pourquoi"}  
>>> dico_es_fr["parfait"] = "perfecto"  
>>> print(dico_es_fr)  
{"si":"oui", "hoy":"aujourd'hui", "porque":"pourquoi", "perfecto":"parfait"}
```

Suppression des éléments du dictionnaire

► On peut supprimer une entrée du dictionnaire par le biais de la fonction `del()` :

```
»> dico_es_fr = {"si":"oui","hoy":"aujourd'hui","porque":"pourquoi"}  
»> del(dico_es_fr["hoy"])  
»> print(dico_es_fr)  
{'si': 'oui', 'porque': 'pourquoi'}
```

Suppression des éléments du dictionnaire

► On peut supprimer une entrée du dictionnaire par le biais de la fonction `del()` :

```
»> dico_es_fr = {"si":"oui","hoy":"aujourd'hui","porque":"pourquoi"}  
»> del(dico_es_fr["hoy"])  
»> print(dico_es_fr)  
{'si': 'oui', 'porque': 'pourquoi'}
```

► On peut vider un dictionnaire avec la méthode `clear()` :

```
»> alph_num = {"zéro":0,"un":1,"deux":2,"trois":3}  
»> print(alph_num)  
{"zéro":0,"un":1,"deux":2,"trois":3}  
»> alph_num.clear()  
»> print(alph_num)  
{ }
```

Fonctions intégrées pour les dictionnaires

- ▶ Les fonctions intégrées suivantes sont couramment utilisées avec les dictionnaires pour effectuer différentes tâches.

Fonctions	Description
all()	Retourne True si toutes les clés du dictionnaire sont True (ou si le dictionnaire est vide).
any()	Retourne Vrai si une des clés du dictionnaire est vraie. Si le dictionnaire est vide, renvoie False.)
copy()	Renvoie une copie superficielle du dictionnaire
cmp()	Compare les éléments de deux dictionnaires. (Non disponible en Python 3)
fromkeys(seq[, v])	Retourne un nouveau dictionnaire avec les clés de seq et une valeur égale à v (par défaut : aucune)
get(key[,d])	Renvoie la valeur de la clé. Si la clé n'existe pas, renvoie d (valeur par défaut : Aucun).

Méthodes sur les dictionnaires en Python

- D'autres méthodes sont présentées dans le tableau ci-dessous.

Méthode	Description
len()	Retourne la longueur (le nombre d'éléments) dans le dictionnaire.
pop(key[,d])	Retire l'article avec la clé et renvoie sa valeur ou d si la clé n'est pas trouvée. Si d n'est pas fourni et que la clé n'est pas trouvée, cela provoque une erreur de clé.
sorted()	Retournez une nouvelle liste triée selon les clés du dictionnaire.
update([other])	Met à jour le dictionnaire avec les paires clé/valeur d'autres clés, en écrasant les clés existantes.
values()	Retourne un nouvel objet des valeurs du dictionnaire

Table des matières

- ① Avant-propos
- ② Environnement Python
- ③ Bases d'utilisation
- ④ Outils de contrôle de flux
- ⑤ Les collections de données
- ⑥ **Les fonctions en Python**
- ⑦ Les modules/librairies en Python
- ⑧ Les fichiers entrées/sorties en Python

Utilité des fonctions

- ▶ En programmation, la notion de *fonction* peut être appliquée pour améliorer :

Utilité des fonctions

- ▶ En programmation, la notion de *fonction* peut être appliquée pour améliorer :
 - * la modularité : rendre le code *plus lisible et plus clair* en le *fractionnant en blocs logiques*.

Utilité des fonctions

- ▶ En programmation, la notion de *fonction* peut être appliquée pour améliorer :
 - * la modularité : rendre le code *plus lisible et plus clair* en le *fractionnant en blocs logiques*.
 - * la réutilisation du code : pour *réaliser plusieurs fois la même opération* au sein d'un programme.

Utilité des fonctions

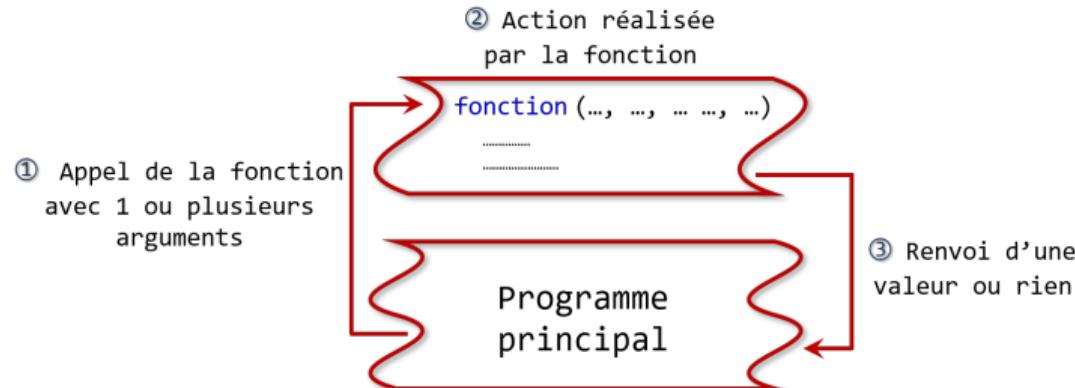
- ▶ En programmation, la notion de *fonction* peut être appliquée pour améliorer :
 - * la modularité : rendre le code *plus lisible et plus clair* en le *fractionnant en blocs logiques*.
 - * la réutilisation du code : pour *réaliser plusieurs fois la même opération* au sein d'un programme.
- ▶ **Python** présente de nombreuses fonctions intégrées, telles que `range()`, `len()`, `print()`, ...

Utilité des fonctions

- ▶ En programmation, la notion de *fonction* peut être appliquée pour améliorer :
 - * la modularité : rendre le code *plus lisible et plus clair* en le *fractionnant en blocs logiques*.
 - * la réutilisation du code : pour *réaliser plusieurs fois la même opération* au sein d'un programme.
- ▶ **Python** présente de nombreuses fonctions intégrées, telles que `range()`, `len()`, `print()`, ...
- ▶ **Python** permet aussi de créer ses propres fonctions, qui sont appelées des fonction définie par l'utilisateur.

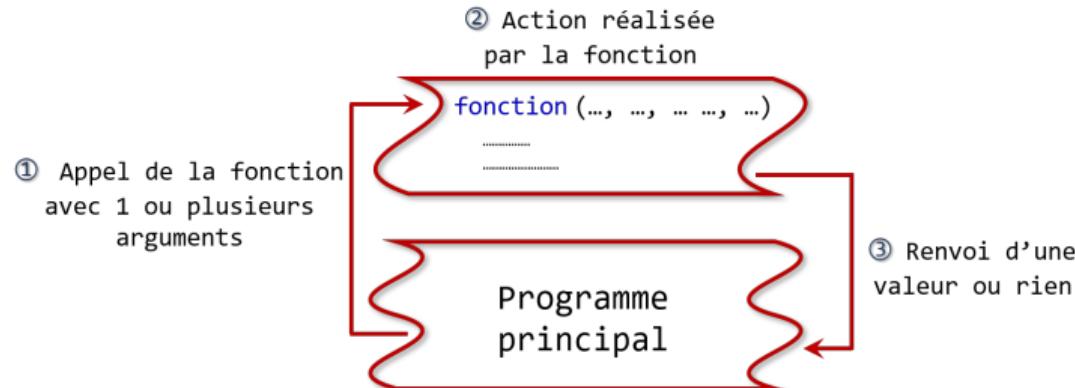
Utilité des fonctions

- En général, une fonction est une sorte de boîte noire dont le code ne fonctionne que lorsqu'il est appelé.



Utilité des fonctions

- En général, une fonction est une sorte de boîte noire dont le code ne fonctionne que lorsqu'il est appelé.



- À laquelle on fait passer aucune, une ou plusieurs variable(s) entre parenthèses. Ces variables sont appelées arguments. Elle effectue une action et renvoie un objet **python** ou rien du tout.

Utilité d'une fonction

- ▶ L'algorithme utilisé au sein de la fonction n'intéresse pas directement l'utilisateur.

Utilité d'une fonction

- ▶ L'algorithme utilisé au sein de la fonction n'intéresse pas directement l'utilisateur.
- ▶ Chaque fonction effectue en général une tâche unique et précise.

Utilité d'une fonction

- ▶ L'algorithme utilisé au sein de la fonction n'intéresse pas directement l'utilisateur.
- ▶ Chaque fonction effectue en général une tâche unique et précise.
- ▶ Si une fonction s'avère trop complexe, il est plus judicieux de la décomposer en plusieurs fonctions (qui peuvent éventuellement s'appeler les unes les autres).

Utilité d'une fonction

- ▶ L'algorithme utilisé au sein de la fonction n'intéresse pas directement l'utilisateur.
- ▶ Chaque fonction effectue en général une tâche unique et précise.
- ▶ Si une fonction s'avère trop complexe, il est plus judicieux de la décomposer en plusieurs fonctions (qui peuvent éventuellement s'appeler les unes les autres).
- ▶ Cette modularité améliore la qualité générale et la lisibilité du code.

Utilité d'une fonction

- ▶ L'algorithme utilisé au sein de la fonction n'intéresse pas directement l'utilisateur.
- ▶ Chaque fonction effectue en général une tâche unique et précise.
- ▶ Si une fonction s'avère trop complexe, il est plus judicieux de la décomposer en plusieurs fonctions (qui peuvent éventuellement s'appeler les unes les autres).
- ▶ Cette modularité améliore la qualité générale et la lisibilité du code.
- ▶ Le terme programme principal (main en anglais) pour désigner l'endroit depuis lequel on appelle une fonction (on verra plus tard que l'on peut en fait appeler une fonction de n'importe où).

Définition d'une fonction

- ▶ Pour définir une fonction, **python** utilise le mot-clé **def**. Si on souhaite que la fonction retourne quelque chose, il faut utiliser le mot-clé **return**.

Définition d'une fonction

- ▶ Pour définir une fonction, **python** utilise le mot-clé **def**. Si on souhaite que la fonction retourne quelque chose, il faut utiliser le mot-clé **return**.
- ▶ La syntaxe **python** pour la définition d'une fonction est la suivante :

Définition d'une fonction

- ▶ Pour définir une fonction, **python** utilise le mot-clé **def**. Si on souhaite que la fonction retourne quelque chose, il faut utiliser le mot-clé **return**.
- ▶ La syntaxe **python** pour la définition d'une fonction est la suivante :

Définition d'une fonction

- ▶ Pour définir une fonction, **python** utilise le mot-clé **def**. Si on souhaite que la fonction retourne quelque chose, il faut utiliser le mot-clé **return**.
- ▶ La syntaxe **python** pour la définition d'une fonction est la suivante :

```
def Nom_function(liste des arguments) :  
    bloc_instr_1  
    return[valeur];
```

mot réservé "def"
n'importe quel nom pour la fonction
paramètres et arguments entrants
deux points
contenu de la fonction à partir de
deux points et l'indentation. Il est
exécuté lorsque la fonction est
appelée par un autre programme
fin de la fonction, le cas échéant retourner une
valeur à l'appelant

Exemples de fonction

```
>>>def hello() :  
    print("Bonjour à partir de la fonction")  
>>>hello()  
Bonjour à partir de la fonction
```

Exemples de fonction

```
>>>def hello() :  
    print("Bonjour à partir de la fonction")
```

```
>>>hello()
```

Bonjour à partir de la fonction

```
>>>var = hello()  
>>>print(var)
```

Bonjour à partir de la fonction

Exemples de fonction

```
>>>def hello() :  
    print("Bonjour à partir de la fonction")
```

```
>>>hello()
```

Bonjour à partir de la fonction

```
>>>var = hello()  
>>>print(var)
```

Bonjour à partir de la fonction

La définition de la fonction est ici

```
>>>def Mes_infos( nom, age) :  
    print("Nom : ", nom, " Age : ", age)  
    return;
```

Exemples de fonction

```
>>>def hello() :  
    print("Bonjour à partir de la fonction")
```

```
>>>hello()
```

Bonjour à partir de la fonction

```
>>>var = hello()  
>>>print(var)
```

Bonjour à partir de la fonction

La définition de la fonction est ici

```
>>>def Mes_infos( nom, age) :  
    print("Nom : ", nom, " Age : ", age)  
    return;
```

Maintenant on peut appeler la fonction Mes_infos

```
>>>Mes_infos( age = 50, nom = "Mustapha")
```

Nom : Mustapha Age : 30

Renvoie de plus d'une valeur

- ▶ Un énorme avantage en **python** est que les fonctions sont capables de renvoyer plusieurs objets à la fois :

Renvoie de plus d'une valeur

- ▶ Un énorme avantage en **python** est que les fonctions sont capables de renvoyer plusieurs objets à la fois :
 - ✳️ Sous forme de Liste :

Renvoie de plus d'une valeur

- ▶ Un énorme avantage en **python** est que les fonctions sont capables de renvoyer plusieurs objets à la fois :
 - ✳️ Sous forme de Liste :

Renvoie de plus d'une valeur

- ▶ Un énorme avantage en **python** est que les fonctions sont capables de renvoyer plusieurs objets à la fois :
 - * Sous forme de Liste :

```
>>>def car_cub_quad(x) :  
    return[x**2, x**3, x**4];  
>>>print(car_cub_quad(5))  
[25, 125, 625]
```

- * Sous forme d'un Tuple ou d'une série de deux éléments (ou plus) :

Renvoie de plus d'une valeur

- ▶ Un énorme avantage en **python** est que les fonctions sont capables de renvoyer plusieurs objets à la fois :
 - * Sous forme de Liste :

```
>>>def car_cub_quad(x) :  
    return[x**2, x**3, x**4];  
  
>>>print(car_cub_quad(5))  
[25, 125, 625]
```

- * Sous forme d'un Tuple ou d'une série de deux éléments (ou plus) :

Renvoie de plus d'une valeur

- ▶ Un énorme avantage en **python** est que les fonctions sont capables de renvoyer plusieurs objets à la fois :

- * Sous forme de Liste :

```
>>>def car_cub_quad(x) :  
    return[x**2, x**3, x**4];  
>>>print(car_cub_quad(5))  
[25, 125, 625]
```

- * Sous forme d'un Tuple ou d'une série de deux éléments (ou plus) :

```
>>>def car_cub_quad(x) :  
    return x**2, x**3, x**4  
>>>print(car_cub_quad(5))  
(25, 125, 625)
```

Les fonctions anonymes

- En **python**, une fonction *anonyme* signifie que la fonction à définir est sans nom.

Les fonctions anonymes

- ▶ En **python**, une fonction *anonyme* signifie que la fonction à définir est sans nom.
- ▶ Comme il a été défini ci-dessus, le mot clé **def** est utilisé pour définir les fonctions normales, alors que pour créer des fonctions anonymes le mot clé **lambda** est réservé à cette fin.

Les fonctions anonymes

- ▶ En **python**, une fonction *anonyme* signifie que la fonction à définir est sans nom.
- ▶ Comme il a été défini ci-dessus, le mot clé **def** est utilisé pour définir les fonctions normales, alors que pour créer des fonctions anonymes le mot clé **lambda** est réservé à cette fin.
- ▶ La syntaxe des fonctions **lambda** ne contient qu'une seule déclaration, qui est la suivante :

Les fonctions anonymes

- ▶ En **python**, une fonction *anonyme* signifie que la fonction à définir est sans nom.
- ▶ Comme il a été défini ci-dessus, le mot clé **def** est utilisé pour définir les fonctions normales, alors que pour créer des fonctions anonymes le mot clé **lambda** est réservé à cette fin.
- ▶ La syntaxe des fonctions **lambda** ne contient qu'une seule déclaration, qui est la suivante :

Les fonctions anonymes

- ▶ En **python**, une fonction *anonyme* signifie que la fonction à définir est sans nom.
- ▶ Comme il a été défini ci-dessus, le mot clé **def** est utilisé pour définir les fonctions normales, alors que pour créer des fonctions anonymes le mot clé **lambda** est réservé à cette fin.
- ▶ La syntaxe des fonctions **lambda** ne contient qu'une seule déclaration, qui est la suivante :

lambda arguments : expression

The diagram illustrates the structure of a lambda expression with the following labels:

- mot réservé « lambda »
- liste des arguments d'entrée
- deux points
- instruction à exécuter sur les arguments

Red arrows point from these labels to the corresponding parts of the lambda expression: 'mot réservé « lambda »' points to the word 'lambda'; 'liste des arguments d'entrée' points to the 'arguments' part; 'deux points' points to the colon (':'); and 'instruction à exécuter sur les arguments' points to the 'expression' part.

Les fonctions anonymes

► Ci-après quelques exemples montreront le fonctionnement des fonctions `lambda`.

Définition de la fonction

»» som = `lambda` parm1, parm2 : parm1 + parm2

Les fonctions anonymes

► Ci-après quelques exemples montreront le fonctionnement des fonctions `lambda`.

Définition de la fonction

»» som = `lambda` parm1, parm2 : parm1 + parm2

Les fonctions anonymes

► Ci-après quelques exemples montreront le fonctionnement des fonctions `lambda`.

Définition de la fonction

»» som = `lambda` parm1, parm2 : parm1 + parm2

La fonction peut être appelée de la sorte

»» `print`("La valeur totale est : ", som(10, 20))

La valeur totale est : 30

Les fonctions anonymes

► Ci-après quelques exemples montreront le fonctionnement des fonctions **lambda**.

Définition de la fonction

```
»> som = lambda parm1, parm2 : parm1 + parm2
```

La fonction peut être appelée de la sorte

```
»> print("La valeur totale est : ", som( 10, 20 ))
```

La valeur totale est : 30

Un autre exemple

```
»> a, b = 4, -2
```

```
»> puiss_xy = lambda x, y : x ** y
```

```
»> print(a , "puissance" , b, " : " , puiss_xy(a, b))
```

4 puissance -2 : 0.0625

Table des matières

- ① Avant-propos
- ② Environnement Python
- ③ Bases d'utilisation
- ④ Outils de contrôle de flux
- ⑤ Les collections de données
- ⑥ Les fonctions en Python
- ⑦ Les modules/librairies en Python
- ⑧ Les fichiers entrées/sorties en Python

Définition d'un module

- ▶ Les modules sont des programmes **Python** contenant un ensemble de fonctions et de variables prédéfinies et fonctionnelles que l'on est amené à réutiliser souvent dans votre code.

Définition d'un module

- ▶ Les modules sont des programmes **Python** contenant un ensemble de fonctions et de variables prédéfinies et fonctionnelles que l'on est amené à réutiliser souvent dans votre code.
- ▶ Les développeurs de **Python** proposent des solutions standardisées à de nombreux problèmes du quotidien du développeur se présentant sous forme de modules natifs (écrits en C) et qui effectuent une quantité phénoménale de tâches.

Définition d'un module

- ▶ Les modules sont des programmes **Python** contenant un ensemble de fonctions et de variables prédéfinies et fonctionnelles que l'on est amené à réutiliser souvent dans votre code.
- ▶ Les développeurs de **Python** proposent des solutions standardisées à de nombreux problèmes du quotidien du développeur se présentant sous forme de modules natifs (écrits en C) et qui effectuent une quantité phénoménale de tâches.
- ▶ La plupart de ces modules sont déjà installés dans les versions standards de Python. Vous pouvez accéder à une documentation exhaustive sur le site de Python.

<https://docs.python.org/3/library/index.html>

Quelques modules courants

Il existe une série de modules que vous serez probablement amenés à utiliser si vous programmez en **Python**. En voici une liste non exhaustive. Pour la liste complète, reportez-vous à la page des modules sur le site de Python :

- * **math** : Fonctions et constantes mathématiques de base (sin, cos, exp, pi...).

Quelques modules courants

Il existe une série de modules que vous serez probablement amenés à utiliser si vous programmez en **Python**. En voici une liste non exhaustive. Pour la liste complète, reportez-vous à la page des modules sur le site de Python :

- * **math** : Fonctions et constantes mathématiques de base (sin, cos, exp, pi...).
- * **os** : Dialogue avec le système d'exploitation.

Quelques modules courants

Il existe une série de modules que vous serez probablement amenés à utiliser si vous programmez en **Python**. En voici une liste non exhaustive. Pour la liste complète, reportez-vous à la page des modules sur le site de Python :

- * **math** : Fonctions et constantes mathématiques de base (sin, cos, exp, pi...).
- * **os** : Dialogue avec le système d'exploitation.
- * **random** : Génération de nombres aléatoires.

Quelques modules courants

Il existe une série de modules que vous serez probablement amenés à utiliser si vous programmez en **Python**. En voici une liste non exhaustive. Pour la liste complète, reportez-vous à la page des modules sur le site de Python :

- ✳ **math** : Fonctions et constantes mathématiques de base (sin, cos, exp, pi...).
- ✳ **os** : Dialogue avec le système d'exploitation.
- ✳ **random** : Génération de nombres aléatoires.
- ✳ **time** : Accès à l'heure de l'ordinateur et aux fonctions gérant le temps.

Quelques modules courants

Il existe une série de modules que vous serez probablement amenés à utiliser si vous programmez en **Python**. En voici une liste non exhaustive. Pour la liste complète, reportez-vous à la page des modules sur le site de Python :

- * **math** : Fonctions et constantes mathématiques de base (sin, cos, exp, pi...).
- * **os** : Dialogue avec le système d'exploitation.
- * **random** : Génération de nombres aléatoires.
- * **time** : Accès à l'heure de l'ordinateur et aux fonctions gérant le temps.
- * **urllib** : Récupération de données sur internet depuis Python.

Quelques modules courants

Il existe une série de modules que vous serez probablement amenés à utiliser si vous programmez en **Python**. En voici une liste non exhaustive. Pour la liste complète, reportez-vous à la page des modules sur le site de Python :

- * **math** : Fonctions et constantes mathématiques de base (sin, cos, exp, pi...).
- * **os** : Dialogue avec le système d'exploitation.
- * **random** : Génération de nombres aléatoires.
- * **time** : Accès à l'heure de l'ordinateur et aux fonctions gérant le temps.
- * **urllib** : Récupération de données sur internet depuis Python.
- * **Tkinter** : Création d'interfaces graphiques.

Importation de modules

- ▶ L'utilisation des différents éléments d'un module passe par son importation.

Importation de modules

► L'utilisation des différents éléments d'un module passe par son importation.

► La syntaxe :

» **import Nom_module**

permet d'importer toute une série de fonctions organisées par thèmes.

Importation de modules

- ▶ L'utilisation des différents éléments d'un module passe par son importation.
- ▶ La syntaxe :
 - » `import Nom_module`

permet d'importer toute une série de fonctions organisées par thèmes.
- ▶ Il existe un autre moyen d'importer une ou plusieurs fonctions d'un module à l'aide du mot-clé `from` :

Importation de modules

▶ L'utilisation des différents éléments d'un module passe par son importation.

▶ La syntaxe :

»» `import Nom_module`

permet d'importer toute une série de fonctions organisées par thèmes.

▶ Il existe un autre moyen d'importer une ou plusieurs fonctions d'un module à l'aide du mot-clé `from` :

▶ La syntaxe :

»» `from Nom_module import *`

permet d'importer toutes les fonctions du module.

Importation de modules

▶ L'utilisation des différents éléments d'un module passe par son importation.

▶ La syntaxe :

»» **import Nom_module**

permet d'importer toute une série de fonctions organisées par thèmes.

▶ Il existe un autre moyen d'importer une ou plusieurs fonctions d'un module à l'aide du mot-clé **from** :

▶ La syntaxe :

»» **from Nom_module import ***

permet d'importer toutes les fonctions du module.

▶ La syntaxe :

»» **from Nom_module import Nom_fonct1, ..., Nom_fonctN**

permet d'importer simultanément *N* fonctions du module.

Création de modules

- ▶ Certaines fonctions peuvent être judicieusement réutilisées dans un autre programme **Python**, d'où l'intérêt de créer un module.

Création de modules

- ▶ Certaines fonctions peuvent être judicieusement réutilisées dans un autre programme **Python**, d'où l'intérêt de créer un module.
- ▶ En général, les modules sont regroupés autour d'un thème précis ; exemple : math, time, wave, ...

Création de modules

- ▶ Certaines fonctions peuvent être judicieusement réutilisées dans un autre programme **Python**, d'où l'intérêt de créer un module.
- ▶ En général, les modules sont regroupés autour d'un thème précis ; exemple : math, time, wave, ...
- ▶ Pour créer un module en **Python**, il suffit d'écrire l'ensemble de fonctions (et/ou de constantes) qui le constituent dans un fichier (comme n'importe quel script **Python**) portant le nom du module suivi du suffixe .py.

Création de modules

- ▶ Certaines fonctions peuvent être judicieusement réutilisées dans un autre programme **Python**, d'où l'intérêt de créer un module.
- ▶ En général, les modules sont regroupés autour d'un thème précis ; exemple : math, time, wave, ...
- ▶ Pour créer un module en **Python**, il suffit d'écrire l'ensemble de fonctions (et/ou de constantes) qui le constituent dans un fichier (comme n'importe quel script **Python**) portant le nom du module suivi du suffixe .py.
- ▶ Depuis un (autre) programme en **Python**, il suffit alors d'utiliser la primitive **import** pour pouvoir utiliser ces fonctions.

Création de modules

- Soit le module contenant les différentes opérations élémentaires de calcul :

calc.py

```
def add(x,y) :
```

```
    return x + y
```

```
def sub(x, y) :
```

```
    return x - y
```

```
def prod(x, y) :
```

```
    return x * y
```

```
def div(x, y) :
```

```
    return x / y
```

Utilisation de son propre module

- ▶ Pour appeler une fonction ou une variable de ce module, il faut que le fichier calc.py soit dans le répertoire courant (dans lequel on travaille). Ensuite, il suffit d'importer le module et toutes ses fonctions (et constantes), puis utiliser les fonctions comme avec un module classique.

Utilisation de son propre module

- ▶ Pour appeler une fonction ou une variable de ce module, il faut que le fichier calc.py soit dans le répertoire courant (dans lequel on travaille). Ensuite, il suffit d'importer le module et toutes ses fonctions (et constantes), puis utiliser les fonctions comme avec un module classique.

Utilisation de son propre module

- ▶ Pour appeler une fonction ou une variable de ce module, il faut que le fichier calc.py soit dans le répertoire courant (dans lequel on travaille). Ensuite, il suffit d'importer le module et toutes ses fonctions (et constantes), puis utiliser les fonctions comme avec un module classique.
- ▶ Exécution :

```
»» import calc as op
```

Utilisation de son propre module

- ▶ Pour appeler une fonction ou une variable de ce module, il faut que le fichier calc.py soit dans le répertoire courant (dans lequel on travaille). Ensuite, il suffit d'importer le module et toutes ses fonctions (et constantes), puis utiliser les fonctions comme avec un module classique.
- ▶ Exécution :

```
»» import calc as op
```

Utilisation de son propre module

- ▶ Pour appeler une fonction ou une variable de ce module, il faut que le fichier calc.py soit dans le répertoire courant (dans lequel on travaille). Ensuite, il suffit d'importer le module et toutes ses fonctions (et constantes), puis utiliser les fonctions comme avec un module classique.
- ▶ Exécution :

```
>>> import calc as op  
>>> a = 10  
>>> b = 20
```



Utilisation de son propre module

- ▶ Pour appeler une fonction ou une variable de ce module, il faut que le fichier calc.py soit dans le répertoire courant (dans lequel on travaille). Ensuite, il suffit d'importer le module et toutes ses fonctions (et constantes), puis utiliser les fonctions comme avec un module classique.
- ▶ Exécution :

```
>>> import calc as op  
>>> a = 10  
>>> b = 20  
>>> addition = op.add(a,b)
```

Utilisation de son propre module

- ▶ Pour appeler une fonction ou une variable de ce module, il faut que le fichier calc.py soit dans le répertoire courant (dans lequel on travaille). Ensuite, il suffit d'importer le module et toutes ses fonctions (et constantes), puis utiliser les fonctions comme avec un module classique.
- ▶ Exécution :

```
>>> import calc as op  
>>> a = 10  
>>> b = 20  
>>> addition = op.add(a,b)  
>>> print(addition)
```

Utilisation de son propre module

- ▶ Pour appeler une fonction ou une variable de ce module, il faut que le fichier calc.py soit dans le répertoire courant (dans lequel on travaille). Ensuite, il suffit d'importer le module et toutes ses fonctions (et constantes), puis utiliser les fonctions comme avec un module classique.
- ▶ Exécution :

```
>>> import calc as op  
>>> a = 10  
>>> b = 20  
>>> addition = op.add(a,b)  
>>> print(addition)
```

30

Table des matières

- ① Avant-propos
- ② Environnement Python
- ③ Bases d'utilisation
- ④ Outils de contrôle de flux
- ⑤ Les collections de données
- ⑥ Les fonctions en Python
- ⑦ Les modules/librairies en Python
- ⑧ Les fichiers entrées/sorties en Python
 - Les fichiers
 - Manipulation des fichiers

Opérations standards I/O

- ▶ La façon la plus simple de produire un résultat est d'utiliser l'instruction d'impression à l'écran : `print()`.

Opérations standards I/O

- ▶ La façon la plus simple de produire un résultat est d'utiliser l'instruction d'impression à l'écran : `print()`.
- ▶ Cette fonction convertit les expressions passées en une chaîne de caractères et écrit le résultat sur la sortie standard, qui par défaut l'écran.

Opérations standards I/O

- ▶ La façon la plus simple de produire un résultat est d'utiliser l'instruction d'impression à l'écran : `print()`.
- ▶ Cette fonction convertit les expressions passées en une chaîne de caractères et écrit le résultat sur la sortie standard, qui par défaut l'écran.
- ▶ **Python** permet de *faire interagir l'utilisateur* en lui permettant de rentrer une ligne de texte à partir d'une entrée standard, qui par défaut provient du clavier.

Opérations standards I/O

- ▶ La façon la plus simple de produire un résultat est d'utiliser l'instruction d'impression à l'écran : `print()`.
- ▶ Cette fonction convertit les expressions passées en une chaîne de caractères et écrit le résultat sur la sortie standard, qui par défaut l'écran.
- ▶ Python permet de *faire interagir l'utilisateur* en lui permettant de rentrer une ligne de texte à partir d'une entrée standard, qui par défaut provient du clavier.
- ▶ Pour ce faire, on dispose de la fonction intégrée : `input()`.

Interaction avec l'utilisateur : la fonction **input()**

- ▶ Dans certaines situations, un programme informatique à besoin d'une intervention de l'utilisateur (entrée d'un paramètre, clic sur un bouton, etc).

Interaction avec l'utilisateur : la fonction **input()**

- ▶ Dans certaines situations, un programme informatique à besoin d'une intervention de l'utilisateur (entrée d'un paramètre, clic sur un bouton, etc).
- ▶ Dans les scripts en mode texte, la méthode la plus simple consiste à employer la fonction intégrée **input()**.

Interaction avec l'utilisateur : la fonction **input()**

- ▶ Dans certaines situations, un programme informatique à besoin d'une intervention de l'utilisateur (entrée d'un paramètre, clic sur un bouton, etc).
- ▶ Dans les scripts en mode texte, la méthode la plus simple consiste à employer la fonction intégrée **input()**.
- ▶ Elle provoque une interruption dans le programme courant et invite l'utilisateur à entrer des caractères au clavier et à terminer avec <>.

Interaction avec l'utilisateur : la fonction **input()**

- ▶ Dans certaines situations, un programme informatique à besoin d'une intervention de l'utilisateur (entrée d'un paramètre, clic sur un bouton, etc).
- ▶ Dans les scripts en mode texte, la méthode la plus simple consiste à employer la fonction intégrée **input()**.
- ▶ Elle provoque une interruption dans le programme courant et invite l'utilisateur à entrer des caractères au clavier et à terminer avec <>.
- ▶ L'exécution du programme se poursuit, et cette fonction lit une seule ligne de texte, sous la forme d'une chaîne, et va être par la suite assignée à une variable quelconque.

Interaction avec l'utilisateur : la fonction **input()**

- ▶ Dans certaines situations, un programme informatique à besoin d'une intervention de l'utilisateur (entrée d'un paramètre, clic sur un bouton, etc).
- ▶ Dans les scripts en mode texte, la méthode la plus simple consiste à employer la fonction intégrée **input()**.
- ▶ Elle provoque une interruption dans le programme courant et invite l'utilisateur à entrer des caractères au clavier et à terminer avec <>.
- ▶ L'exécution du programme se poursuit, et cette fonction lit une seule ligne de texte, sous la forme d'une chaîne, et va être par la suite assignée à une variable quelconque.
- ▶ Cette fonction peut être appelée en laissant l'entre parenthèses vide comme on peut aussi y placer un message explicatif destiné à l'utilisateur.

Utilisation de la fonction **input()**

```
# Input avec parenthèses vides  
=> print("Veuillez entrer un nombre :")  
=> val = input()  
Veuillez entrer un nombre : 56
```

Utilisation de la fonction **input()**

```
# Input avec parenthèses vides
>>> print("Veuillez entrer un nombre :")
>>> val = input()
Veuillez entrer un nombre : 56
>>> type(val)
<class str>
>>> int(val)
>>> type(val)
<class int>
```

Utilisation de la fonction **input()**

```
# Input avec parenthèses vides
```

```
»» print("Veuillez entrer un nombre :")
```

```
»» val = input()
```

```
Veuillez entrer un nombre : 56
```

```
»» type(val)
```

```
<class str>
```

```
»» int(val)
```

```
»» type(val)
```

```
<class int>
```

```
# Input avec message
```

```
»» val = float(input("Veuillez entrer un nombre :"))
```

```
Veuillez entrer un nombre : 23
```

Utilisation de la fonction **input()**

```
# Input avec parenthèses vides
```

```
»» print("Veuillez entrer un nombre :")
```

```
»» val = input()
```

```
Veuillez entrer un nombre : 56
```

```
»» type(val)
```

```
<class str>
```

```
»» int(val)
```

```
»» type(val)
```

```
<class int>
```

```
# Input avec message
```

```
»» val = float(input("Veuillez entrer un nombre :"))
```

```
Veuillez entrer un nombre : 23
```

```
»» type(val)
```

```
<class 'float'>
```

Table des matières

- ① Avant-propos
- ② Environnement Python
- ③ Bases d'utilisation
- ④ Outils de contrôle de flux
- ⑤ Les collections de données
- ⑥ Les fonctions en Python
- ⑦ Les modules/librairies en Python
- ⑧ Les fichiers entrées/sorties en Python
 - Les fichiers
 - Manipulation des fichiers

Notion de fichier

- ▶ Comme la mémoire vive (RAM) est volatile, les données doivent être stockées de façon permanente pour une utilisation future.

Notion de fichier

- ▶ Comme la mémoire vive (RAM) est volatile, les données doivent être stockées de façon permanente pour une utilisation future.
- ▶ Les fichiers sont des emplacements conçus sur le disque pour stocker de façon permanente des données.

Notion de fichier

- ▶ Comme la mémoire vive (RAM) est volatile, les données doivent être stockées de façon permanente pour une utilisation future.
- ▶ Les fichiers sont des emplacements conçus sur le disque pour stocker de façon permanente des données.
- ▶ Ainsi, en **python**, une opération sur un fichier se déroule dans l'ordre suivant :

Notion de fichier

- ▶ Comme la mémoire vive (RAM) est volatile, les données doivent être stockées de façon permanente pour une utilisation future.
- ▶ Les fichiers sont des emplacements conçus sur le disque pour stocker de façon permanente des données.
- ▶ Ainsi, en **python**, une opération sur un fichier se déroule dans l'ordre suivant :
 - ✳ Ouvrir un fichier.

Notion de fichier

- ▶ Comme la mémoire vive (RAM) est volatile, les données doivent être stockées de façon permanente pour une utilisation future.
- ▶ Les fichiers sont des emplacements conçus sur le disque pour stocker de façon permanente des données.
- ▶ Ainsi, en **python**, une opération sur un fichier se déroule dans l'ordre suivant :
 - ✳ Ouvrir un fichier.
 - ✳ Lire ou écrire (Effectuer des opérations de lecture **ou** d'écriture).

Notion de fichier

- ▶ Comme la mémoire vive (RAM) est volatile, les données doivent être stockées de façon permanente pour une utilisation future.
- ▶ Les fichiers sont des emplacements conçus sur le disque pour stocker de façon permanente des données.
- ▶ Ainsi, en **python**, une opération sur un fichier se déroule dans l'ordre suivant :
 - ✳ Ouvrir un fichier.
 - ✳ Lire ou écrire (Effectuer des opérations de lecture **ou** d'écriture).
 - ✳ Fermer le fichier (libérer les ressources systèmes liées au fichier).

Table des matières

- ① Avant-propos
- ② Environnement Python
- ③ Bases d'utilisation
- ④ Outils de contrôle de flux
- ⑤ Les collections de données
- ⑥ Les fonctions en Python
- ⑦ Les modules/librairies en Python
- ⑧ Les fichiers entrées/sorties en Python
 - Les fichiers
 - Manipulation des fichiers

Opérations sur les fichiers

▶ Ouverture de fichier

Opérations sur les fichiers

▶ Ouverture de fichier

- * Python a une fonction intégrée `open()` pour ouvrir un fichier.

Opérations sur les fichiers

▶ Ouverture de fichier

- ✿ Python a une fonction intégrée `open()` pour ouvrir un fichier.
- ✿ Elle renvoie un descripteur de fichier qui sera utilisé pour *lire* ou *modifier* le fichier en conséquence.



▶ Ouverture de fichier

- ✿ Python a une fonction intégrée `open()` pour ouvrir un fichier.
- ✿ Elle renvoie un descripteur de fichier qui sera utilisé pour *lire* ou *modifier* le fichier en conséquence.
- ✿ La syntaxe est la suivante :

▶ Ouverture de fichier

- ✿ Python a une fonction intégrée `open()` pour ouvrir un fichier.
- ✿ Elle renvoie un descripteur de fichier qui sera utilisé pour *lire* ou *modifier* le fichier en conséquence.
- ✿ La syntaxe est la suivante :

Opérations sur les fichiers

▶ Ouverture de fichier

- ✿ Python a une fonction intégrée `open()` pour ouvrir un fichier.
- ✿ Elle renvoie un descripteur de fichier qui sera utilisé pour *lire* ou *modifier* le fichier en conséquence.
- ✿ La syntaxe est la suivante :

```
descripteur = open(nom_fich, mode)
```

Mot réservé « open »

Chaîne spécifiant le type d'opération à effectuer sur le fichier

Nom ou le chemin du fichier



Opérations sur les fichiers

- ▶ Fermeture de fichier

Opérations sur les fichiers

▶ Fermeture de fichier

- * Une fois que le travail sur le fichier est achevé ou qu'on veuille l'ouvrir en un autre mode, le fichier doit être fermé en faisant recours à la méthode `close()` comme suit :

Opérations sur les fichiers

▶ Fermeture de fichier

- * Une fois que le travail sur le fichier est achevé ou qu'on veuille l'ouvrir en un autre mode, le fichier doit être fermé en faisant recours à la méthode `close()` comme suit :

Opérations sur les fichiers

▶ Fermeture de fichier

- * Une fois que le travail sur le fichier est achevé ou qu'on veuille l'ouvrir en un autre mode, le fichier doit être fermé en faisant recours à la méthode `close()` comme suit :

descripteur.`close()`



Mot réservé « close »

Mode d'ouverture

► Le tableau suivant répertorie les différents modes disponibles.

Mode	Description
'r'	Ouvrez un fichier en lecture. (par défaut)
'w'	Ouvrez un fichier pour l'écriture. Dans ce mode, si le fichier spécifié n'existe pas, il sera créé. Si le fichier existe, alors ses données sont détruites.
'x'	Ouvrez un fichier pour une création exclusive. Si le fichier existe déjà, l'opération échoue.
'a'	Ouvrez un fichier en mode ajout. Si le fichier n'existe pas, ce mode le créera. Si le fichier existe déjà, les nouvelles données seront ajoutées à la fin du fichier.
'+'	Ouvrir un fichier pour la mise à jour (lecture et écriture)

Opérations sur les fichiers

Attention!!!

Lorsqu'on travaille avec des fichiers en mode texte, il est vivement recommandé de spécifier le type de codage.

Opérations sur les fichiers

Attention!!!

Lorsqu'on travaille avec des fichiers en mode texte, il est vivement recommandé de spécifier le type de codage.

```
»> fid = open("Etudiants.txt", "r", encoding = 'utf-8')
```

Opérations sur les fichiers

Attention!!!

Lorsqu'on travaille avec des fichiers en mode texte, il est vivement recommandé de spécifier le type de codage.

```
»> fid = open("Etudiants.txt", "r", encoding = 'utf-8')
```

Remarque

Python propose une construction appelée *gestionnaire de contexte* permettant de gérer l'ouverture et la fermeture d'un fichier dans un bloc d'instructions en utilisant l'instruction *with*.

Opérations sur les fichiers

Attention!!!

Lorsqu'on travaille avec des fichiers en mode texte, il est vivement recommandé de spécifier le type de codage.

```
»> fid = open("Etudiants.txt", "r", encoding = 'utf-8')
```

Remarque

Python propose une construction appelée *gestionnaire de contexte* permettant de gérer l'ouverture et la fermeture d'un fichier dans un bloc d'instructions en utilisant l'instruction *with*.

```
»> with open("Etudiants.txt", "r", encoding = 'utf-8') as fid :  
    # Bloc relatif au traitement sur le fichier  
»> # Suite du programme
```

Opérations sur les fichiers

Informations sur fichier

- file est un descripteur, diverses informations sur le fichier peuvent être obtenues :

Méthode	Description
file.closed	Retourne vrai si le dossier est clos, faux sinon.
file.mode	Retourne le mode d'accès avec lequel le fichier a été ouvert.
file.name	Renvoie le nom du fichier.

Opérations sur les fichiers

Informations sur fichier

- file est un descripteur, diverses informations sur le fichier peuvent être obtenues :

Méthode	Description
file.closed	Retourne vrai si le dossier est clos, faux sinon.
file.mode	Retourne le mode d'accès avec lequel le fichier a été ouvert.
file.name	Renvoie le nom du fichier.

Opérations sur les fichiers

Informations sur fichier

- file est un descripteur, diverses informations sur le fichier peuvent être obtenues :

Méthode	Description
file.closed	Retourne vrai si le dossier est clos, faux sinon.
file.mode	Retourne le mode d'accès avec lequel le fichier a été ouvert.
file.name	Renvoie le nom du fichier.

```
»> fich = open("foot.txt", "w")
»> print("Nom du fichier : ", fich.name)
Nom du fichier : foot.txt
```

Opérations sur les fichiers

Informations sur fichier

- file est un descripteur, diverses informations sur le fichier peuvent être obtenues :

Méthode	Description
file.closed	Retourne vrai si le dossier est clos, faux sinon.
file.mode	Retourne le mode d'accès avec lequel le fichier a été ouvert.
file.name	Renvoie le nom du fichier.

```
>>> fich = open("foot.txt", "w")
>>> print("Nom du fichier : ", fich.name)
Nom du fichier : foot.txt
>>> print("Fermé ou pas : ", fich.closed)
Fermé ou pas : False
```

Opérations sur les fichiers

Informations sur fichier

- file est un descripteur, diverses informations sur le fichier peuvent être obtenues :

Méthode	Description
file.closed	Retourne vrai si le dossier est clos, faux sinon.
file.mode	Retourne le mode d'accès avec lequel le fichier a été ouvert.
file.name	Renvoie le nom du fichier.

```
>>> fich = open("foot.txt", "w")
>>> print("Nom du fichier : ", fich.name)
Nom du fichier : foot.txt
>>> print("Fermé ou pas : ", fich.closed)
Fermé ou pas : False
>>> print("Mode d'ouverture : ", fich.mode)
Mode d'ouverture : w
```

Opérations sur les fichiers

▶ Écriture dans un fichier

Opérations sur les fichiers

▶ Écriture dans un fichier

- ✿ Si on considère un fichier décrit par son descripteur *fich*, la méthode *fich.write(chaine)* écrit la chaîne passée en argument dans le fichier à la position pointée par le pointeur d'accès.

Opérations sur les fichiers

▶ Écriture dans un fichier

- ✿ Si on considère un fichier décrit par son descripteur *fich*, la méthode *fich.write(chaine)* écrit la chaîne passée en argument dans le fichier à la position pointée par le pointeur d'accès.
- ✿ Le pointeur d'accès est automatiquement déplacé après les données écrites.

Opérations sur les fichiers

▶ Écriture dans un fichier

- ✿ Si on considère un fichier décrit par son descripteur *fich*, la méthode *fich.write(chaine)* écrit la chaîne passée en argument dans le fichier à la position pointée par le pointeur d'accès.
- ✿ Le pointeur d'accès est automatiquement déplacé après les données écrites.

Opérations sur les fichiers

▶ Écriture dans un fichier

- ✿ Si on considère un fichier décrit par son descripteur *fich*, la méthode *fich.write(chaine)* écrit la chaîne passée en argument dans le fichier à la position pointée par le pointeur d'accès.
- ✿ Le pointeur d'accès est automatiquement déplacé après les données écrites.

»» with `open("Etudiants.txt", "w", encoding = 'utf-8')` as fid :

```
fid.write(" Mostafa \n ")
fid.write(" Ismail \n ")
fid.write(" Dounia \n ")
```

autres instruction

Opérations sur les fichiers

▶ Écriture dans un fichier

- ★ Si on considère un fichier décrit par son descripteur *fich*, la méthode *fich.write(chaine)* écrit la chaîne passée en argument dans le fichier à la position pointée par le pointeur d'accès.
- ★ Le pointeur d'accès est automatiquement déplacé après les données écrites.

»» with `open("Etudiants.txt", "w", encoding = 'utf-8')` as fid :

```
fid.write(" Mostafa \n ")
fid.write(" Ismail \n ")
fid.write(" Dounia \n ")
```

autres instruction

Remarque

On doit inclure les caractères de nouvelle ligne, \n, afin de distinguer les différentes lignes dans le fichier.

Opérations sur les fichiers

▶ Lecture d'un fichier

Opérations sur les fichiers

▶ Lecture d'un fichier

- * Nous pouvons lire un nombre défini (*size*) de données en utilisant la méthode *read(size)*. Si *size* n'est pas spécifié, la lecture se fait jusqu'à la fin du fichier.

Opérations sur les fichiers

▶ Lecture d'un fichier

- * Nous pouvons lire un nombre défini (*size*) de données en utilisant la méthode *read(size)*. Si *size* n'est pas spécifié, la lecture se fait jusqu'à la fin du fichier.

Opérations sur les fichiers

▶ Lecture d'un fichier

- * Nous pouvons lire un nombre défini (*size*) de données en utilisant la méthode `read(size)`. Si *size* n'est pas spécifié, la lecture se fait jusqu'à la fin du fichier.

» fid = `open("Test.txt", "r", encoding = 'utf-8')`

» fid.`read(4)` # Il va lire les quatre premières données du fichier.

Ceci

» fid.`read(4)` # Il va lire les quatre données qui suivent du fichier.

est

» fid.`read()` # Lecture de la totalité jusqu'à la fin du fichier.

premier fichier

Portant sur plusieurs lignes

» fid.`read()` # Une lecture ultérieure renvoie à une chaîne vide.

Opérations sur les fichiers

▶ Lecture d'un fichier

Pr. Abdellah ADIB

Opérations sur les fichiers

▶ Lecture d'un fichier

- * On peut aussi utiliser la méthode `readline()` pour lire les lignes individuelles d'un fichier. Cette méthode lit un fichier jusqu'à la nouvelle ligne, y compris le caractère de nouvelle ligne.

Opérations sur les fichiers

▶ Lecture d'un fichier

- * On peut aussi utiliser la méthode `readline()` pour lire les lignes individuelles d'un fichier. Cette méthode lit un fichier jusqu'à la nouvelle ligne, y compris le caractère de nouvelle ligne.

Opérations sur les fichiers

▶ Lecture d'un fichier

- * On peut aussi utiliser la méthode `readline()` pour lire les lignes individuelles d'un fichier. Cette méthode lit un fichier jusqu'à la nouvelle ligne, y compris le caractère de nouvelle ligne.

```
>>> fid = open("Test.txt", "r", encoding = 'utf-8')
```

```
>>> fid.readline() # Il va lire la première ligne du fichier.
```

Ceci est mon premier fichier

```
>>> fid.readline() # Il va lire la deuxième ligne du fichier.
```

Portant sur plusieurs lignes

```
>>> fid.readline() # Rien à lire car la fin du fichier.
```

Opérations sur les fichiers

▶ Lecture d'un fichier

Opérations sur les fichiers

▶ Lecture d'un fichier

- * Enfin, la méthode *readlines()* renvoie une liste des lignes restantes de l'ensemble du fichier. Comme toutes les autres méthodes de lecture, elle renvoie des valeurs vides lorsque la fin du fichier (EOF) est atteinte.

▶ Lecture d'un fichier

- * Enfin, la méthode *readlines()* renvoie une liste des lignes restantes de l'ensemble du fichier. Comme toutes les autres méthodes de lecture, elle renvoie des valeurs vides lorsque la fin du fichier (EOF) est atteinte.

Opérations sur les fichiers

▶ Lecture d'un fichier

- Enfin, la méthode `readlines()` renvoie une liste des lignes restantes de l'ensemble du fichier. Comme toutes les autres méthodes de lecture, elle renvoie des valeurs vides lorsque la fin du fichier (EOF) est atteinte.

```
>>> fid = open("Test.txt", "r", encoding = 'utf-8')
>>> fid.readlines() # Il va lire toutes les lignes du fichier.
Ceci est mon premier fichier
Portant sur plusieurs lignes
>>> fid.readline() # Rien à lire car la fin du fichier.
```

Opérations sur les fichiers

- ▶ Position dans le fichier (curseur)

Opérations sur les fichiers

▶ Position dans le fichier (curseur)

- * On peut accéder au curseur (position) actuelle (en nombre d'octets) dans le fichier en traitement en utilisant la méthode `tell()`. De même, la méthode `seek()` permet de modifier le curseur (position).

Opérations sur les fichiers

▶ Position dans le fichier (curseur)

- * On peut accéder au curseur (position) actuelle (en nombre d'octets) dans le fichier en traitement en utilisant la méthode `tell()`. De même, la méthode `seek()` permet de modifier le curseur (position).

Opérations sur les fichiers

▶ Position dans le fichier (curseur)

- On peut accéder au curseur (position) actuelle (en nombre d'octets) dans le fichier en traitement en utilisant la méthode `tell()`. De même, la méthode `seek()` permet de modifier le curseur (position).

```
>>> fid = open("Test.txt", "r", encoding = 'utf-8')  
>>> fid.readline() # Il va lire la première ligne du fichier.
```

Ceci est mon premier fichier

```
>>> current = fid.tell() # Obtenir la position actuelle dans fichier.  
30
```

```
>>> fid.seek(0) # Ramène le curseur dans le fichier à sa position initiale.
```

0

```
>>> print(fid.read())
```

Ceci est mon premier fichier

Portant sur plusieurs lignes



UNIVERSITE HASSAN II DE CAS BLANCA
FACUITE DES SCIENCES ET TECHNIQUES

جامعة الحسن الثاني بالدار البيضاء
كلية العلوم والتكنولوجيات