

ib  
cegos

# Python Initiation





# ib Cegos en synthèse

- 11 sites en France : La Défense, Lyon, Sophia-Antipolis, Aix-en-Provence, Toulouse, Bordeaux, Nantes, Rennes, Rouen, Lille et Strasbourg
- Une certification qualité ISO 9001, la certification Qualiopi et l'agrément Data Dock
- Offre de formations informatiques à forte valeur ajoutée depuis 35 ans
- Assure chaque année la montée en compétences de plus de 25 000 spécialistes des technologies de l'information
- Le catalogue ib Cegos : <https://www.ib-formation.fr/formations>



# Bienvenue sur cette formation



## — Présentation du formateur

*Xavier TABUTEAU*





# Bienvenue sur cette formation



## — Validation des prérequis pour suivre cette formation

*Tour de table pour vérifier les prérequis*





**Avant de  
commencer**

- **Durée de la formation** : 14h
- **Objectifs** : Initiation à Python
- **Organisation**
  - ▶ **Horaires** : 9h à 17h/17h30
  - ▶ **Pauses** : 15 min en matinée et après midi (vers 10h30 et 15h30)
  - ▶ **Déjeuner** : 12h30 à 13h30



# Déroulé, structure de la formation et formalités

- Première chose à faire : signer les feuilles d'émargement et mentionner le caractère obligatoire avec de vrais signatures (ni croix ni initiales).
- Dernier jour de la formation : le centre de formation à l'obligation contractuelle de fournir vos évaluations à votre entreprise avant 15h, donc au retour de la pause déjeuner, 2 ou 3 h avant la fin de la formation, je vous ferai remplir les évaluations formateur.
- La structure d'une journée : présenter une notion théorique, suivie de la pratique (écriture du code), suivi d'un exercice. Une fois que j'ai abordé avec vous 3 ou 4 notions, 1 tp de validation des acquis qui porte sur ces notions. Ce TP sera à faire en groupe.
- J'enverrai les corrections sur les notions abordées dans un Github dont je vous donnerai le lien en début de formation.
- A la fin de chaque demi journée, je vous donnerai un lien Google Forms pour la validation des acquis et l'adaptabilité qui devra être rempli obligatoirement.
- S'il me reste du temps, je reviens sur toutes les questions hors plan de cours que les stagiaires m'ont posées durant la formation.





**ib  
cegos**

# Table des matières



# Table des matières

## — Chapitres

- ▶ Introduction
- ▶ Installations
- ▶ Premiers pas
- ▶ Les imports et modules
- ▶ Les variables
- ▶ Les structures répétitives
- ▶ Les structures conditionnelles
- ▶ Les fonctions
- ▶ TP de validation des acquis 1



# Table des matières

## — Chapitres

- ▶ Les générateurs
- ▶ Les docstrings
- ▶ Les exceptions
- ▶ Le débogage
- ▶ Les fichiers
- ▶ Les expressions régulières
- ▶ Les classes
- ▶ Les classes avancées
- ▶ TP de validations des acquis 2



# Table des matières

## — Chapitres

- ▶ Les patrons de conception
- ▶ UML
- ▶ Les modules utiles
- ▶ Les bases de données
- ▶ Le gestionnaire de paquets
- ▶ Les environnements virtuels
- ▶ L'interface graphique Tkinter
- ▶ Les tests
- ▶ L'interfaçage Python / C
- ▶ Les extractions automatiques de documentation



**ib**  
**cegos**

# Introduction



# Introduction

## – Pourquoi Python ?

- ▶ Python est un langage interprété avec un typage dynamique fort, portable, extensible, gratuit, qui permet (sans l'imposer) une approche modulaire et orientée objet de la programmation.
- ▶ Particularité importante : Python n'utilise pas d'accolades ou d'autres délimiteurs (begin/end...) pour repérer les blocs d'un programme, mais l'indentation.
- ▶ Un exemple : en fonction d'une liste de valeurs, nous souhaitons savoir celles qui sont des nombres pairs et celles qui ne le sont pas.

# Introduction

## En PHP

```
<?php
function valeurs_paires($liste_valeurs) {
    $classement = array();
    for($i=0; $i<count($liste_valeurs); $i++) {
        if($liste_valeurs[$i] % 2 == 0) {
            array_push($classement, True);
        } else {
            array_push($classement, False);
        }
    }
    return $classement;
}

echo var_dump(valeurs_paires([51, 8, 85, 9]));
?>
```

## En Python

```
def valeurs_paires(liste_valeurs):
    return [(False if v % 2 else True) for v in liste_valeurs]

print valeurs_paires([51, 8, 85, 9])
```

```
[False, True, False, False]
```

# Introduction

## – Champs d'application de Python

- ▶ Nous trouvons Python dans le Web, les multimédias, la bureautique, les utilitaires, l'intelligence artificielle, ...
- ▶ Nous le retrouvons dans tous les domaines professionnels tel que :
  - Le domaine scientifique, les finances, la programmation système, les base de données, ...
- ▶ Python à cette force de pouvoir réunir des profils d'informaticiens assez différents (administrateur système, développeur généraliste, développeur web, etc...).



# Introduction

## — Positionnement de Python :

- ▶ C'est l'un des langages les plus utilisés :

ref : <https://www.blogdumoderateur.com/langages-informatiques-populaires-janvier-2025/>

## — Evolution :

- ▶ 1994 : Python 1.0
- ▶ 2000 : Python 2.0
- ▶ 2008 : Python 3.0
- ▶ ???? : Python 4.0 ?



# Introduction

## — Objectif d'apprentissage

- ▶ Installation de Python et choix d'un IDE.
- ▶ Travailler avec les variables, conditions et boucles.
- ▶ Utilisation des modules et classes.
- ▶ Lecture et écriture de fichiers.
- ▶ Manipulation des fonctions et classes.
- ▶ Découvrir quelques librairies.



# Introduction

## — Prérequis

- ▶ Base de la programmation (Les variables, les fonctions, les expressions).
- ▶ Les concepts POO (Classe, héritage)



**ib**  
**cegos**

**Installations**



# Installations

## – Installer Python

- ▶ Python sera installé grâce à un fichier exécutable que vous pouvez télécharger à ce lien :  
<https://www.python.org/downloads/>

## – Lancer son premier programme

- ▶ Grâce à la console Python, nous pourrons exécuter notre première application.

# Installations

## — Installation de Python

Avant de se rendre pour le téléchargement, vous pouvez vérifier si Python est installé dans votre ordinateur en exécutant la commande : `python --version`

Python fait partie des principales distributions Linux et vient avec tout Mac équipé de Mac OS.

Grace au lien ci-dessous, vous êtes en mesure de sélectionner l'exécutable qui correspond à votre système.



# Installations

## — Installation de Python

- <https://www.python.org/downloads/windows/>

### Files

Version	Operating System	Description	MD5 Sum	File Size	GPG	Sigstore		
<a href="#">Gzipped source tarball</a>	Source release		1aea68575c0e97bc83ff8225977b0d46	26006589	<a href="#">SIG</a>	<a href="#">CRT</a>	<a href="#">SIG</a>	
<a href="#">XZ compressed source tarball</a>	Source release		b8094f007b3a835ca3be6bdf8116cccc	19618696	<a href="#">SIG</a>	<a href="#">CRT</a>	<a href="#">SIG</a>	
<a href="#">macOS 64-bit universal2 installer</a>	macOS	for macOS 10.9 and later	4c89649f6ca799ff29f1d1dffcb9393	40865361	<a href="#">SIG</a>	<a href="#">CRT</a>	<a href="#">SIG</a>	
<a href="#">Windows embeddable package (32-bit)</a>	Windows		7e4de22bfe1e6d333b2c691ec2c1fcee	7615330	<a href="#">SIG</a>	<a href="#">CRT</a>	<a href="#">SIG</a>	
<a href="#">Windows embeddable package (64-bit)</a>	Windows		7f90f8642c1b19cf02bce91a5f4f9263	8591256	<a href="#">SIG</a>	<a href="#">CRT</a>	<a href="#">SIG</a>	
<a href="#">Windows help file</a>	Windows		643179390f5f5d9d6b1ad66355c795bb	9355326	<a href="#">SIG</a>	<a href="#">CRT</a>	<a href="#">SIG</a>	
<a href="#">Windows installer (32-bit)</a>	Windows		58755d6906f825168999c83ce82315d7	27779240	<a href="#">SIG</a>	<a href="#">CRT</a>	<a href="#">SIG</a>	
<a href="#">Windows installer (64-bit)</a>	Windows	Recommended	bfb8467c7e3504f3800b0fe94d9a3e6	28953568	<a href="#">SIG</a>	<a href="#">CRT</a>	<a href="#">SIG</a>	

# Installations

## — Installation de Python

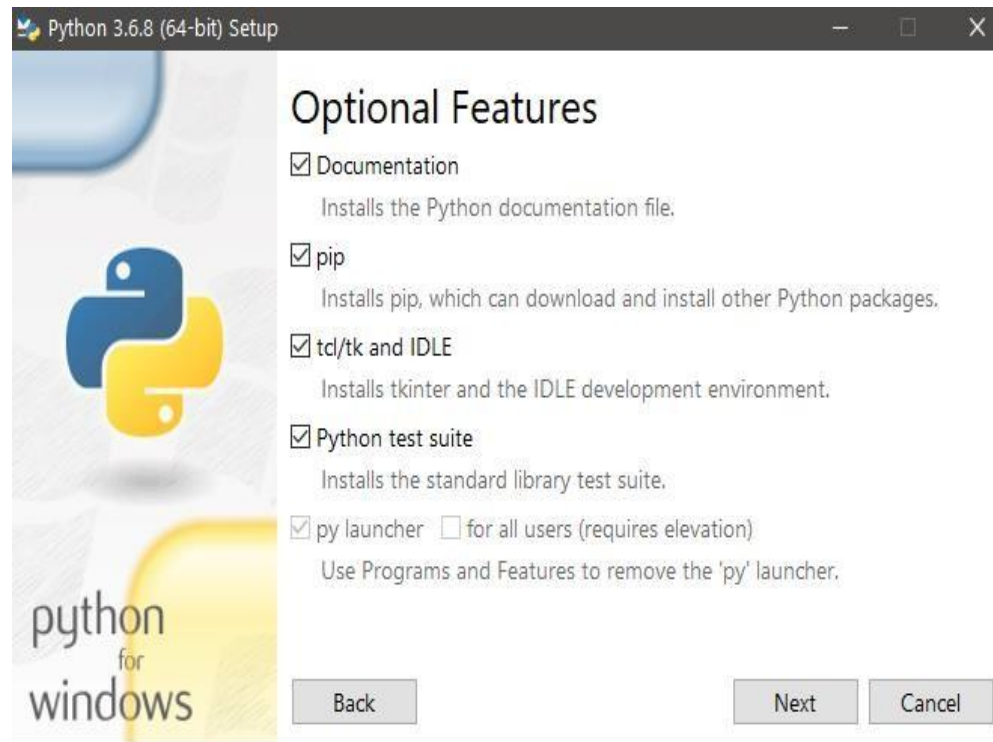
- ▶ A la première fenêtre choisir Customize Installation et cliquez sur suivant.



# Installations

## — Installation de Python

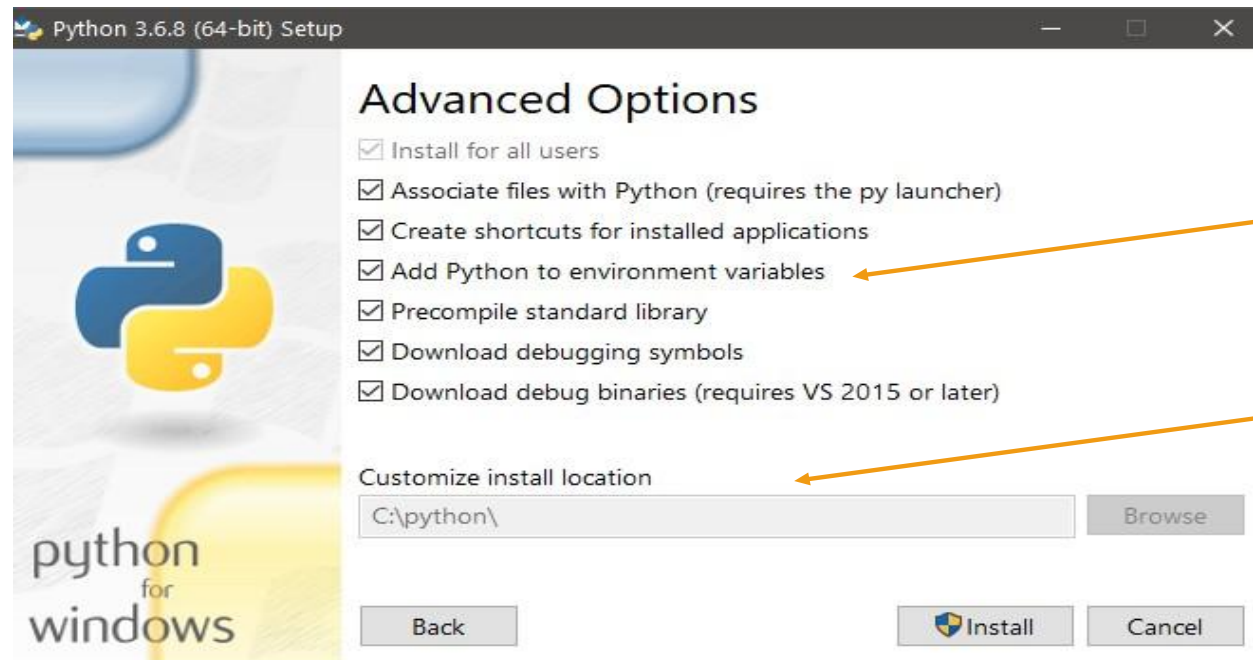
- ▶ A la fenêtre des options cocher toutes les cases puis suivant.



# Installations

## — Installation de Python

- ▶ La prochaine étape vous permettra de choisir dans quel dossier sera installé. Python et principalement de l'ajouter aux variables d'environnements.



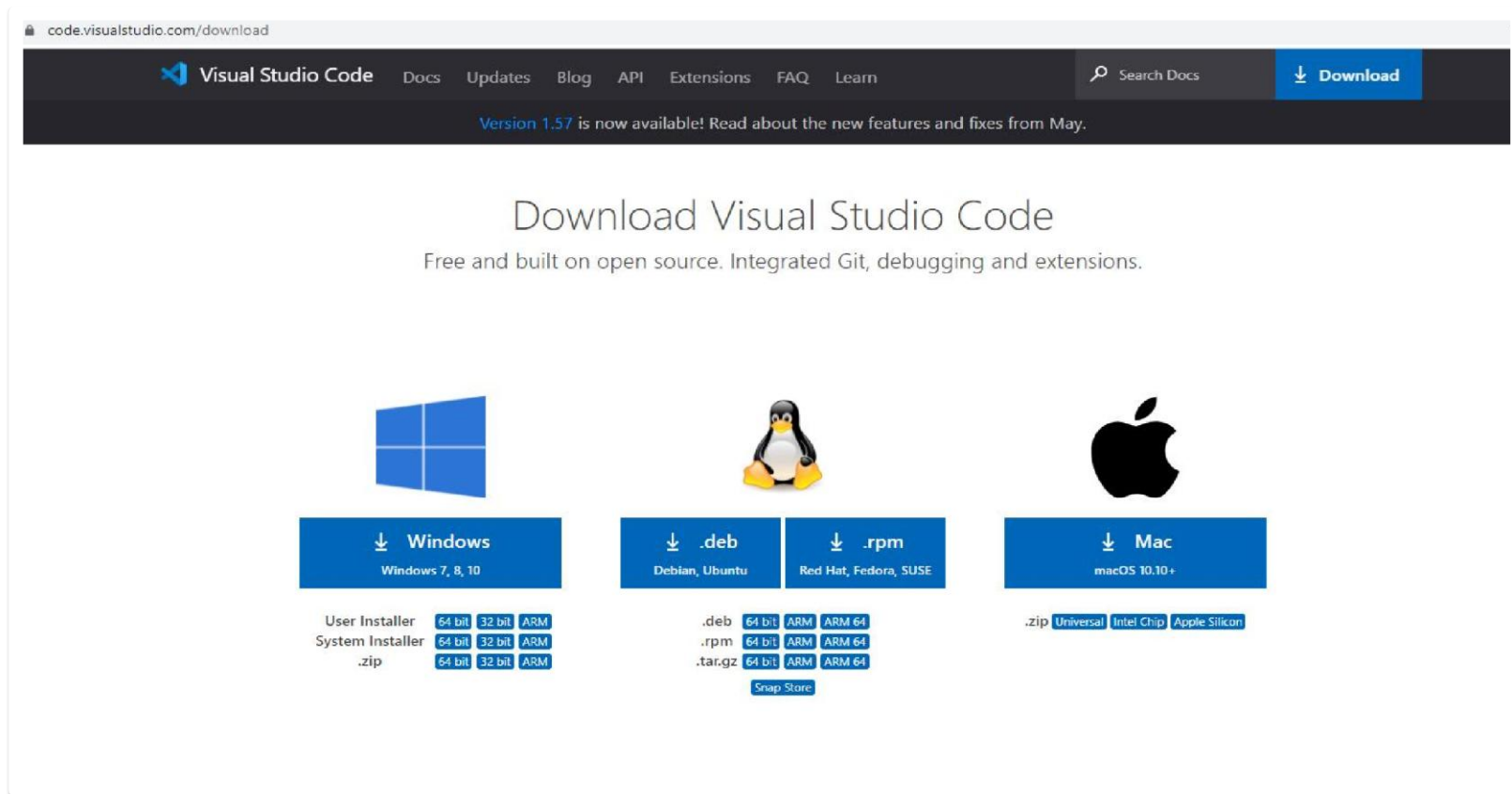
# Installations

## — Installation de VS Code

- ▶ Un IDE (Integrated Development Environment) est un regroupement d'outils utiles pour le développement d'applications (éditeur de code, debugger, builder, indexation du code pour recherches « intelligentes » dans les projets...), rassemblés dans un logiciel unique. (Eclipse, Netbeans, Xcode, Pycharm, Wingware).
- ▶ Python est avant tout un langage de script, et un simple éditeur de code avec quelques fonctions utiles peut suffire.
- ▶ Cet éditeur, qui est gratuit, nous accompagnera tout au long de notre programmation en python.
- ▶ Lien de téléchargement : <https://code.visualstudio.com/download>

# Installations

## — Installation de VS Code



The screenshot shows the Visual Studio Code download page. The browser address bar displays 'code.visualstudio.com/download'. The page header includes the Visual Studio Code logo, navigation links (Docs, Updates, Blog, API, Extensions, FAQ, Learn), a search bar, and a 'Download' button. A banner below the header announces 'Version 1.57 is now available! Read about the new features and fixes from May.' The main heading is 'Download Visual Studio Code', followed by the tagline 'Free and built on open source. Integrated Git, debugging and extensions.' Below this, three operating system sections are shown: Windows (with the Windows logo), Linux (with the Tux penguin logo), and Mac (with the Apple logo). Each section contains download links for various installers and their supported architectures.

code.visualstudio.com/download

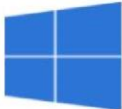
Visual Studio Code Docs Updates Blog API Extensions FAQ Learn

Search Docs Download

Version 1.57 is now available! Read about the new features and fixes from May.


### Download Visual Studio Code

Free and built on open source. Integrated Git, debugging and extensions.



↓ Windows  
Windows 7, 8, 10

User Installer 64 bit 32 bit ARM  
System Installer 64 bit 32 bit ARM  
.zip 64 bit 32 bit ARM




↓ .deb  
Debian, Ubuntu

↓ .rpm  
Red Hat, Fedora, SUSE

.deb 64 bit ARM ARM 64  
.rpm 64 bit ARM ARM 64  
.tar.gz 64 bit ARM ARM 64

Snap Store



↓ Mac  
macOS 10.10+

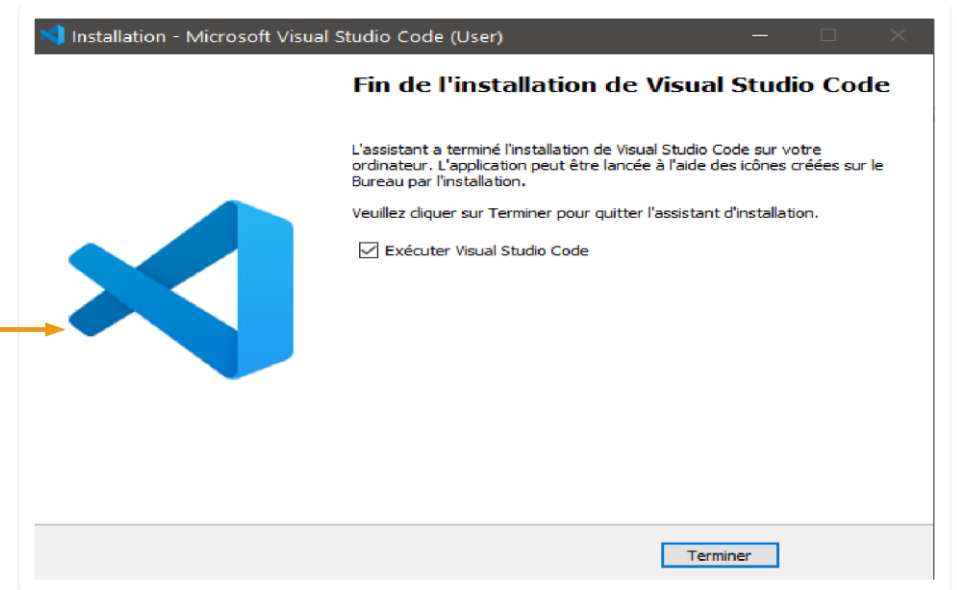
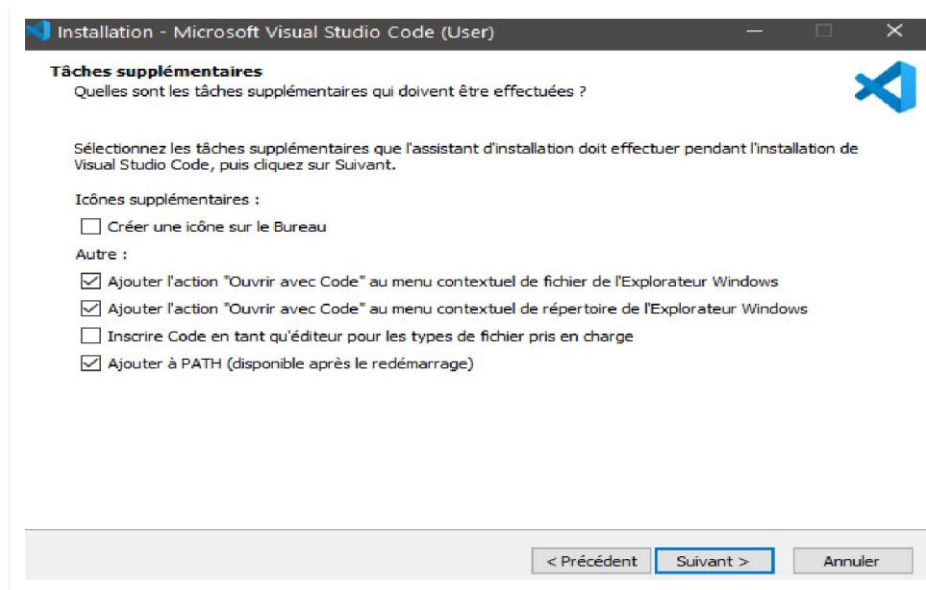
.zip Universal Intel Chip Apple Silicon



# Installations

## — Installation de VS Code

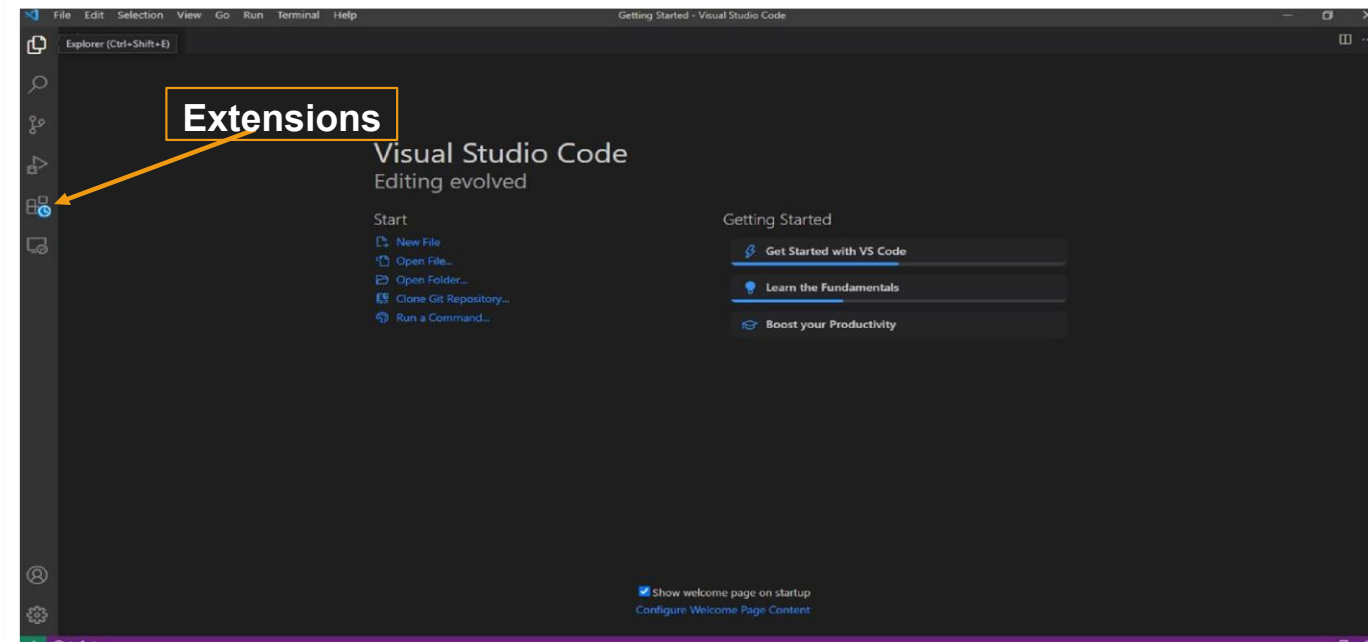
- ▶ Le processus d'installation de VS Code est très simple et n'exigera pas trop de configurations.
- ▶ Une fois la tâche finie, vous pouvez déjà lancer l'éditeur.



# Installations

## — Installation de VS Code

- Afin de faciliter la saisie de nos codes Python, nous allons installer l'extension « Python ».
- 1. Placer votre curseur à gauche et cliquer sur l'option « Extensions ».

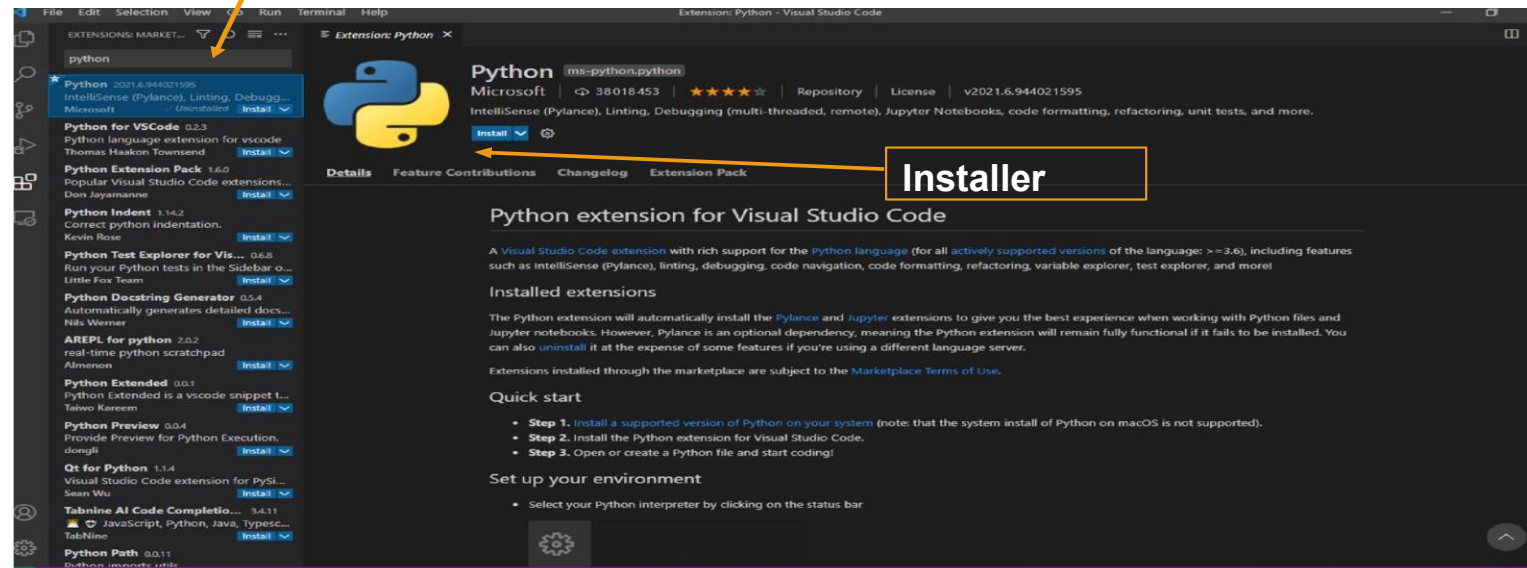


# Installations

## – Installation de VS Code

- ▶ 2. Dans la zone de recherche (coin supérieur gauche), vous cherchez et trouvez l'extension « Python » et ensuite vous l'installez.

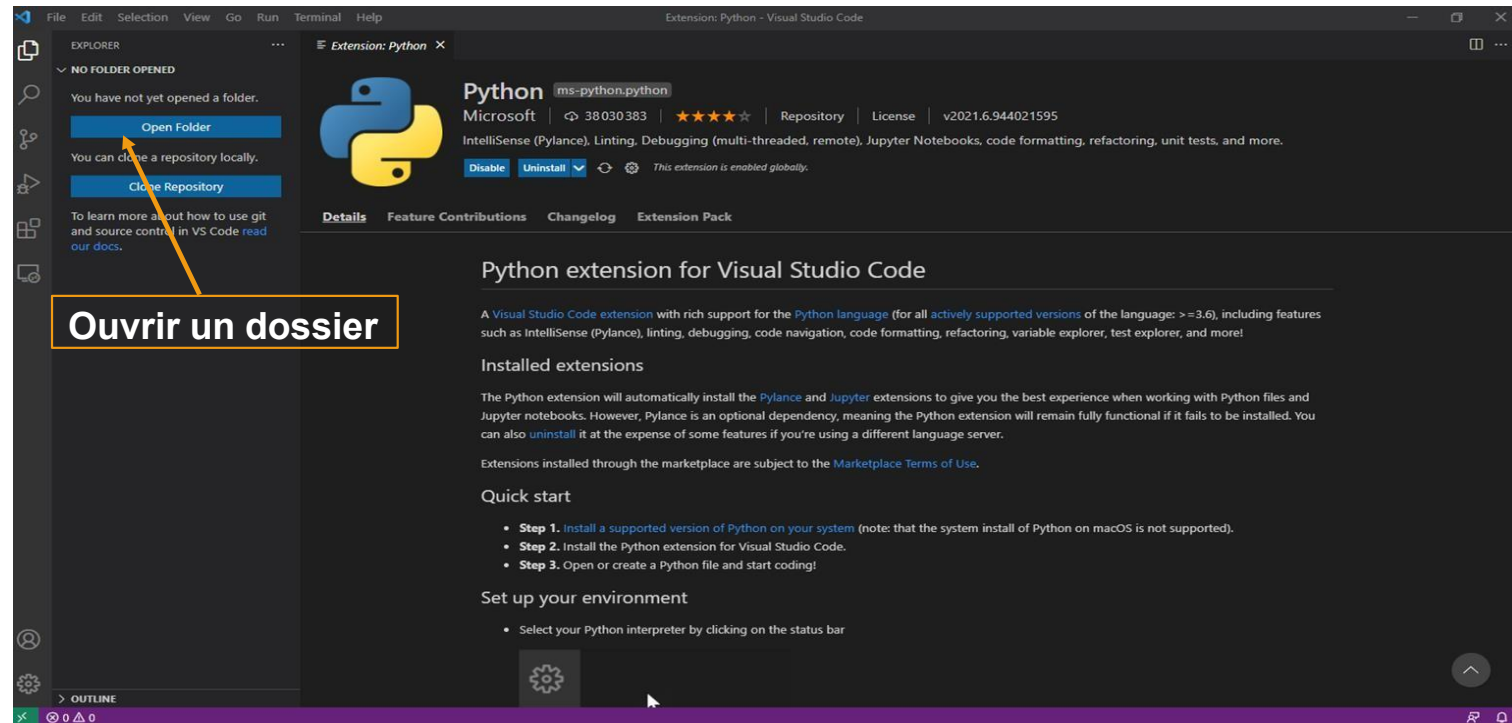
Rechercher et sélectionner



# Installations

## — Installation de VS Code

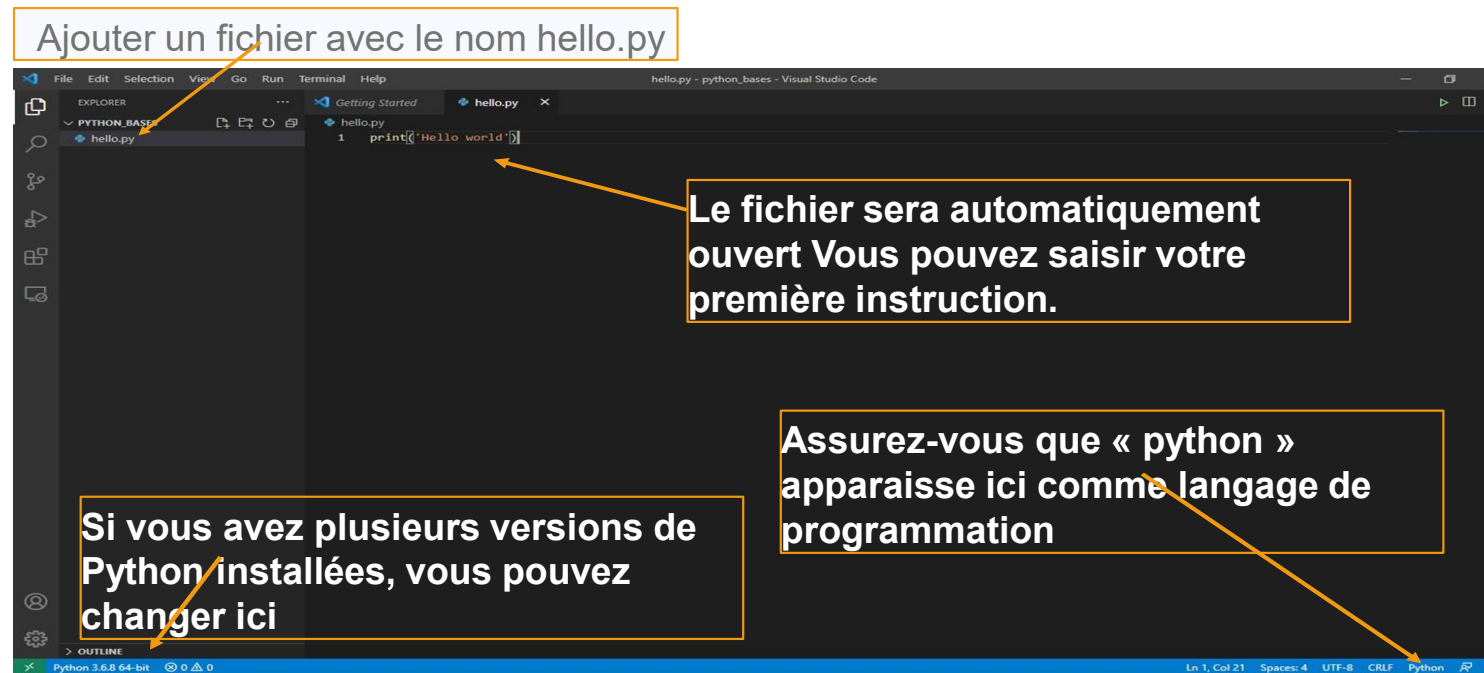
- Nous allons créer notre premier fichier capable d'exécuter du code python. D'abord, il faudra sélectionner ou créer un dossier.



# Installations

## — Installation de VS Code

- Une fois le dossier ouvert, nous créons un fichier avec l'extension python « .py ». Ensuite, nous nous rassurons si le fichier est bien pris en charge par l'extension « python » que nous avons installé préalablement.





# Installations

## – Comment exécuter Python ?

- ▶ Invite de commande

- ▶ Console Python

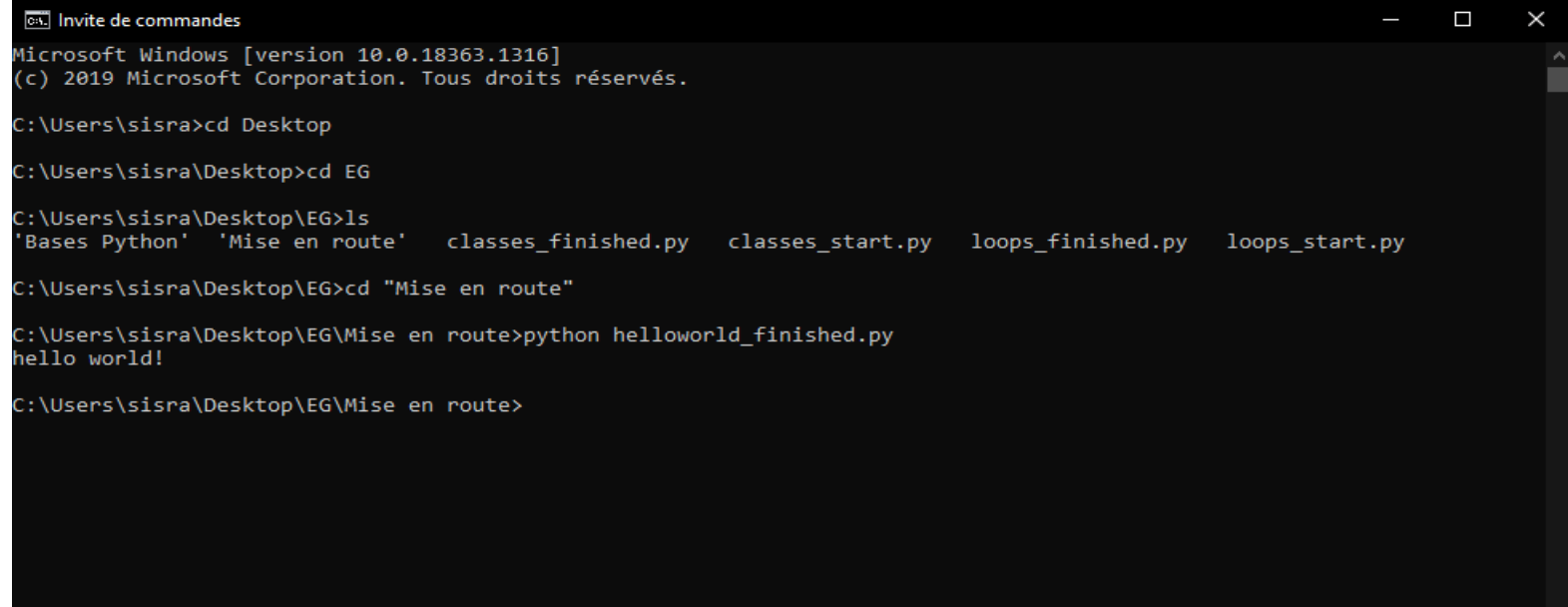
- ▶ VS Code



# Installations

## — Invite de commande

- Nous pouvons tester différentes commandes du langage python grâce à l'invite de commande. En ouvrant le terminal sur notre machine, il suffit de se rendre dans le répertoire où se trouve notre fichier .py puis saisir la commande python suivi du nom du fichier.



```
Invite de commandes
Microsoft Windows [version 10.0.18363.1316]
(c) 2019 Microsoft Corporation. Tous droits réservés.

C:\Users\sisra>cd Desktop
C:\Users\sisra\Desktop>cd EG
C:\Users\sisra\Desktop\EG>ls
'Bases Python'  'Mise en route'  classes_finished.py  classes_start.py  loops_finished.py  loops_start.py
C:\Users\sisra\Desktop\EG>cd "Mise en route"
C:\Users\sisra\Desktop\EG\Mise en route>python helloworld_finished.py
hello world!
C:\Users\sisra\Desktop\EG\Mise en route>
```

# Installations

## – Console Python

- Nous pouvons tester différentes commandes du langage python grâce à la console. En ouvrant le terminal sur notre machine, il suffit de saisir la commande « python » pour accéder à la console Python.

Lancer la console Python

```
C:\>python
Python 3.6.8 (tags/v3.6.8:3c6b436a57, Dec 24 2018, 00:16:47) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print('hello world')
hello world
>>> |
```

Instruction python pour afficher du texte

Résultat de l'instruction

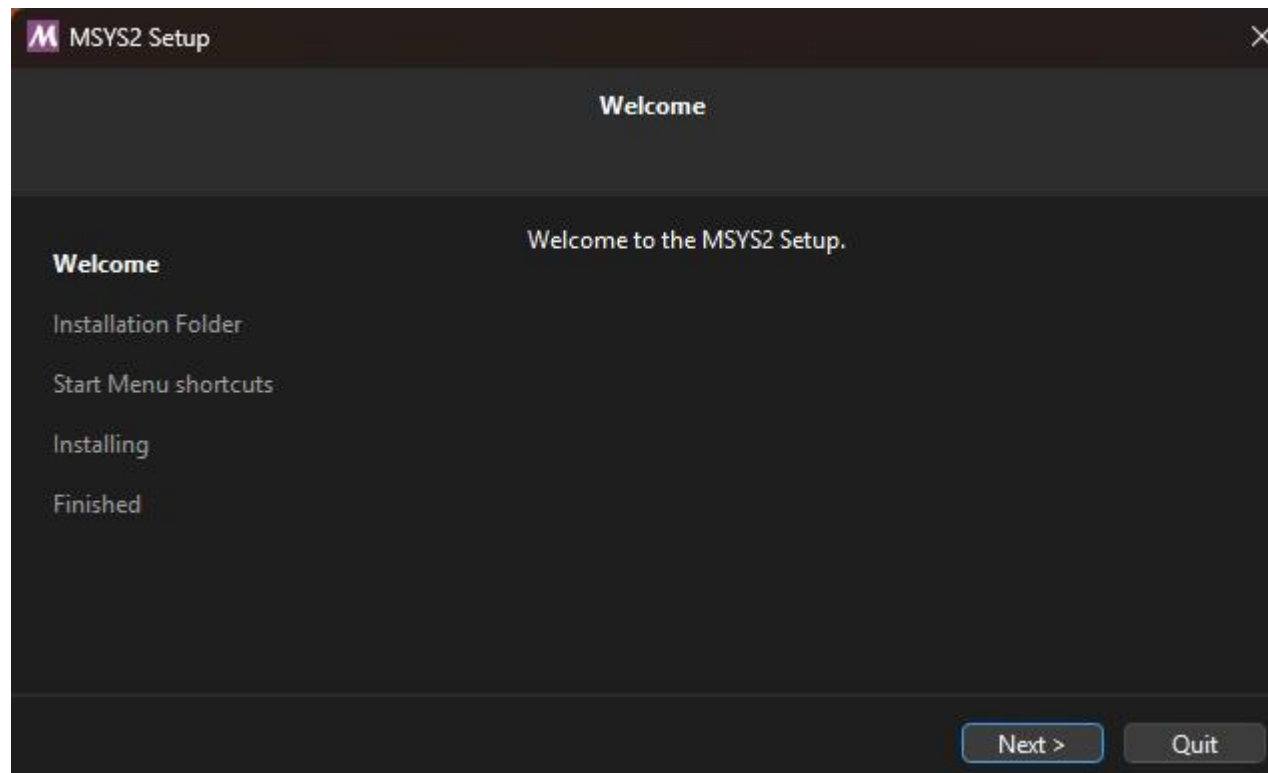
# Installations

## — Installation du Compilateur C

- ▶ Pour invoquer des fonctions codées en C avec Python, nous aurons besoin d'un compilateur C sur notre ordinateur.
- ▶ Pour Windows : <https://www.msys2.org>
- ▶ Pour mac ou linux, il est possible qu'il y en ait déjà un d'installé.
- ▶ Télécharger l'installateur avec le lien ci-dessus et exécuter l'installateur en mode administrateur.
- ▶ Cliquer sur le bouton « Next »

# Installations

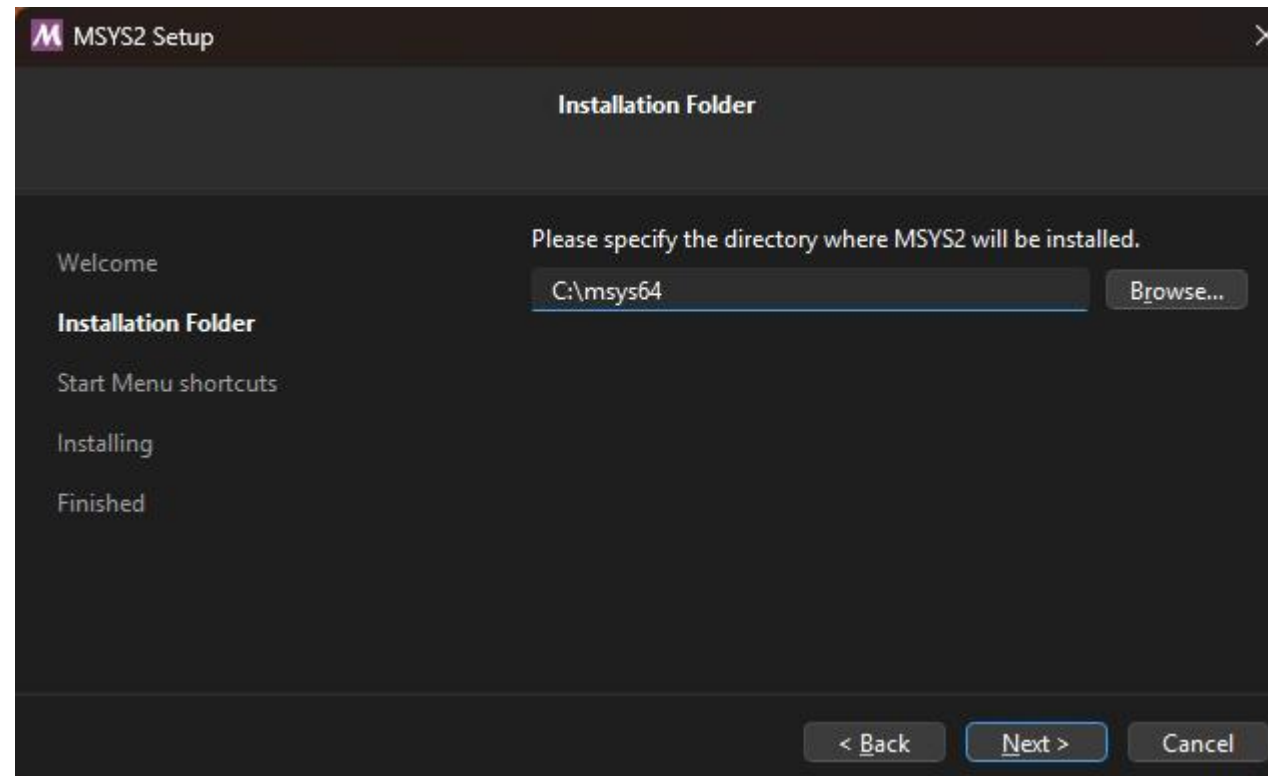
- Installation du Compilateur C
  - Cliquer sur le bouton « Next »



# Installations

## — Installation du Compilateur C

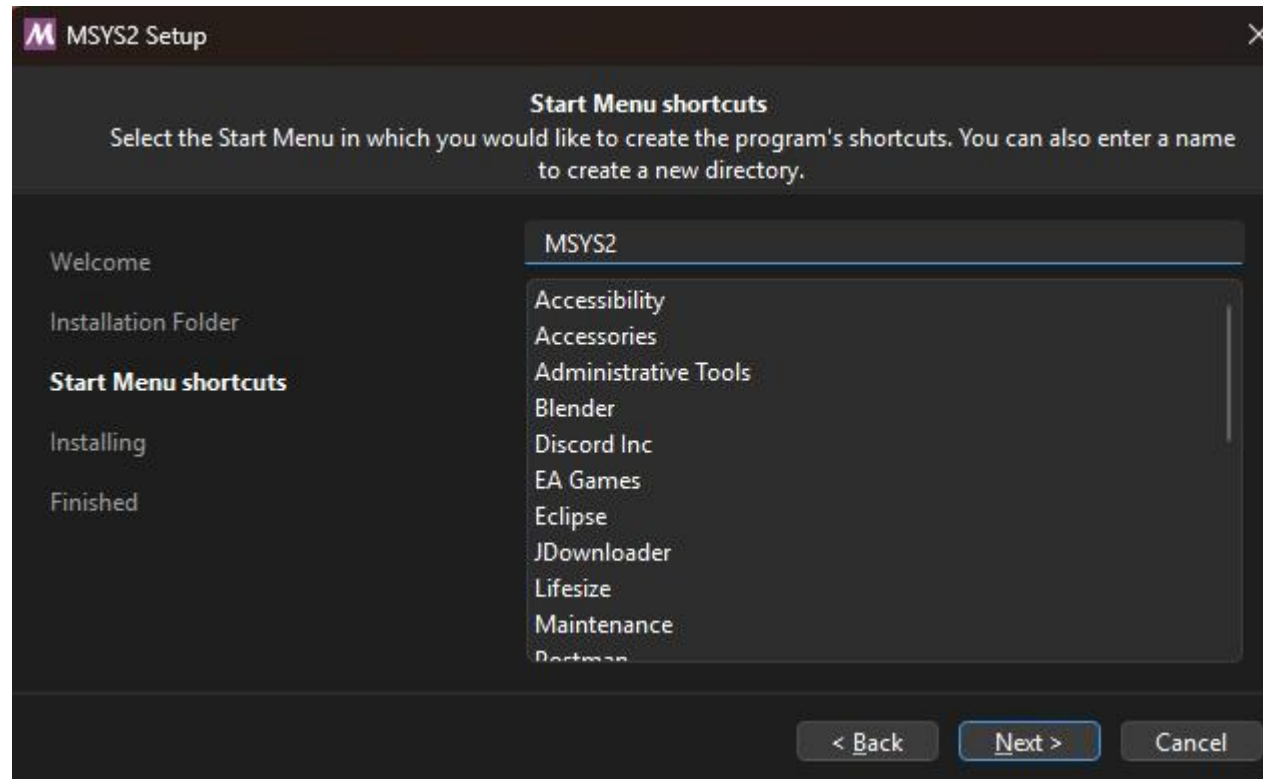
- Choisir l'emplacement pour l'installation ou laisser par défaut et cliquer sur le bouton « Next ».



# Installations

## — Installation du Compilateur C

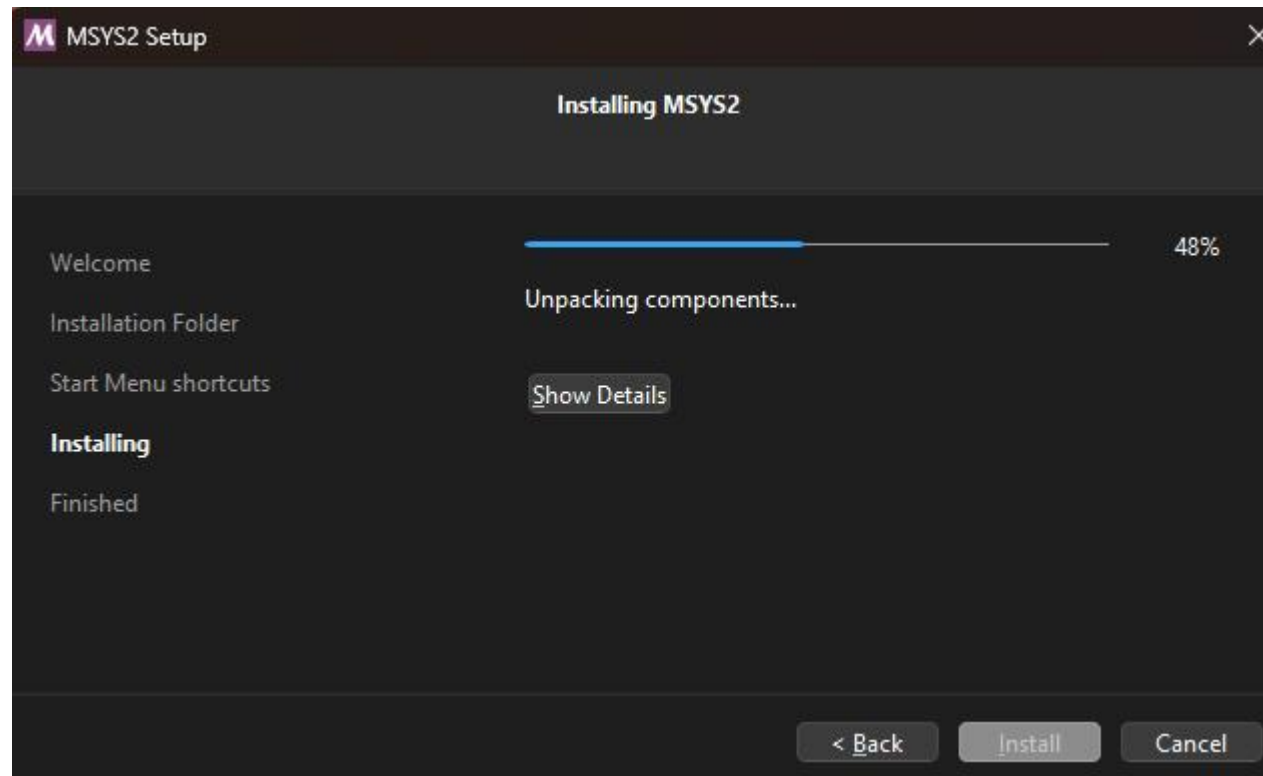
- Pour les raccourcis du menu démarrer, laisser par défaut et cliquer sur « Next ».



# Installations

## — Installation du Compilateur C

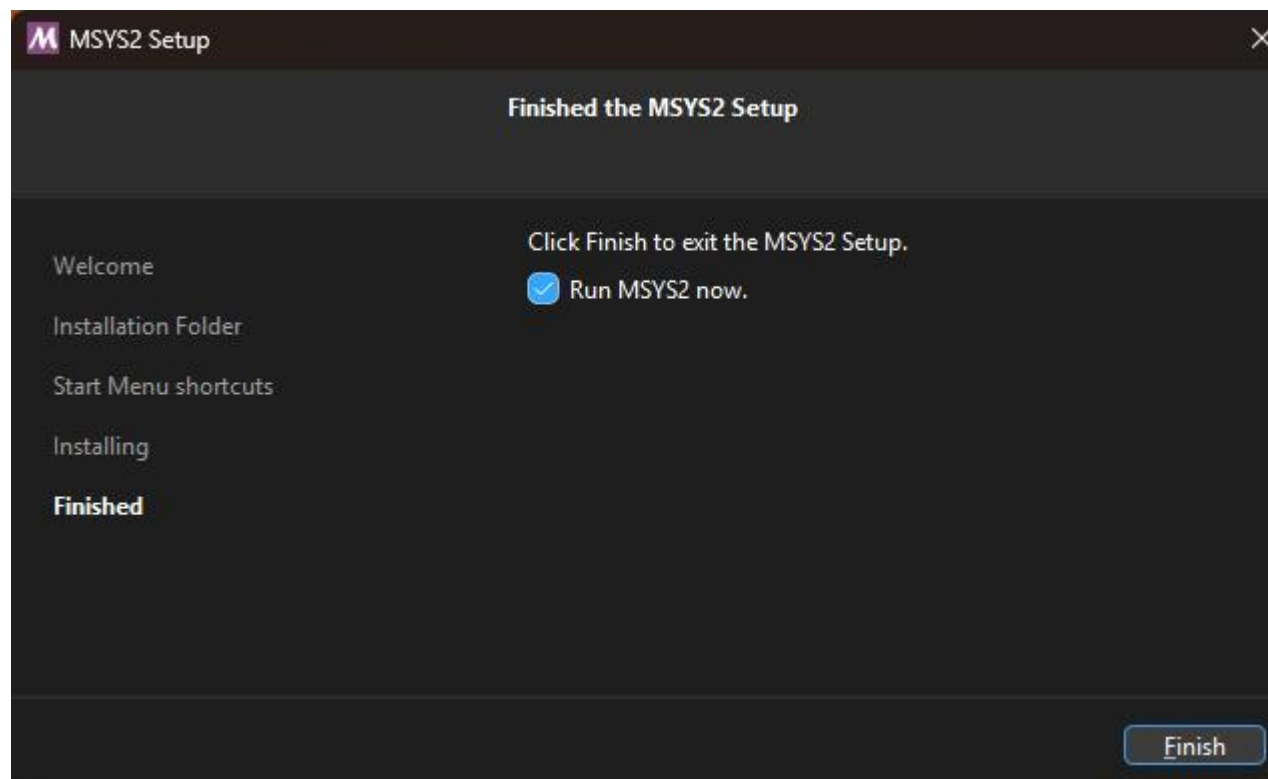
- Attendre pendant l'installation.



# Installations

## — Installation du Compilateur C

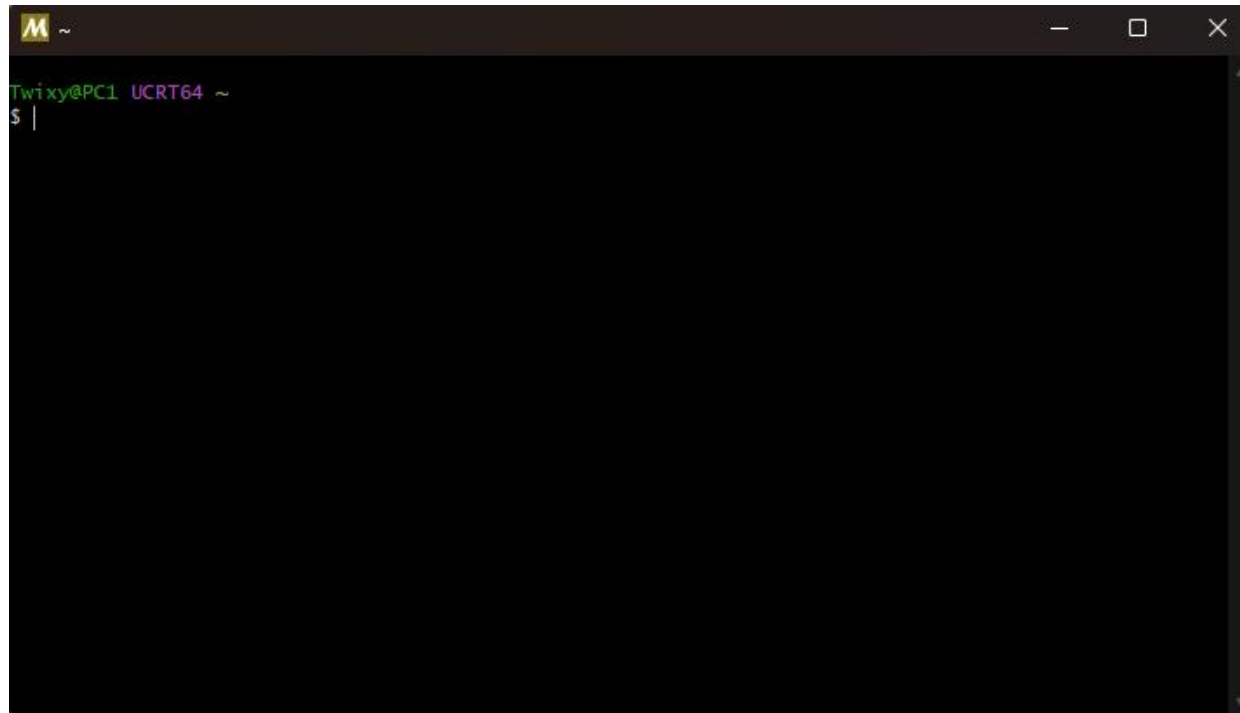
- Laisser la case cochée et cliquer sur « Finish ».





# Installations

- Installation du Compilateur C
  - Une fenêtre noire s'ouvre.



# Installations

## — Installation du Compilateur C

- ▶ Ecrire la ligne de commande suivante dans cette fenêtre pour installer ucrt-w64 GCC :
  - `pacman -S mingw-w64-ucrt-x86_64-gcc`
- ▶ Valider et attendre la fin des téléchargements et installations. Exécuter cette ligne de commande afin de mettre à jour tous les packages y compris la fenêtre :
  - `pacman -Suy`
- ▶ Mettre en variable d'environnement l'emplacement de `gcc.exe` (`c:\msys64\ucrt64\bin`) et redémarrer le PC.

ib  
cegos

**Premiers pas**



# Premiers pas

## – Utilisons la console Python

### ► Calculer avec Python

- `>>> 15 + 4`                      `>>> 5 ** 2`
- `>>> 2 - 9`                        `>>> 20 / 3`
- `>>> 4 * 3`                        `>>> 5 // 3`

### ► Données, Variables, Affectation

- `>>> a = 15`
- `>>> nom = "Kaiser"`

# Premiers pas

## – Utilisons la console Python

### ► Affichage de la valeur d'une variable

- `>>> print(a)`
- `>>> print(nom)`

### ► Affectation multiple

- `>>> x = y = 7`      `# Affectation parallèle`
- `>>> a, b = 5, 18`    `# Affectation multiple`

# Premiers pas

## – Utilisons la console Python

### ► Composition

- `>>> print(17 + 3)`      `# Ceci est un affichage`
- `>>> print("somme est de" , 15 * 3 + 4)`

# Premiers pas

## — Petites choses diverses à savoir pour débiter en Python

### ► Commentaire en python

- Les commentaires en Python commencent avec un caractère dièse, #, et s'étendent jusqu'à la fin de la ligne. Un commentaire peut apparaître au début d'une ligne ou à la suite d'un espace ou de code, mais pas à l'intérieur d'une chaîne de caractères littérale.
- Il y a aussi les commentaires multi lignes commençant et finissant par triple quotes simples ou doubles. On les appelle les docstrings.

### ► Quelques mots sur l'encodage des sources et UTF-8

- Pour pouvoir utiliser des caractères « spéciaux » (accents notamment) dans vos programmes (même dans les commentaires), vous devez dire à Python, de manière explicite, que vous souhaitez utiliser le codage de caractères UTF-8.

# -\*- coding: UTF-8 -\*-

# Premiers pas

## – Petites choses diverses à savoir pour débiter en Python

► `if __name__ == '__main__': # kesako ?`

- `main()` n'existe pas en Python, comme on peut le trouver en C ou java par exemple.
- Il y a néanmoins un cas de figure où le fait de ne pas avoir ce genre de fonction peut être problématique : quand on inclut un module dans un autre, Python réalise un import de tout le contenu du module. Le problème, c'est que si on y place des instructions à l'extérieur de toute fonction ou méthode, elles seront exécutées systématiquement, même lors de l'inclusion du module, ce qui n'est pas terrible : on souhaite généralement importer les fonctions et classes, mais pas lancer les instructions.
- C'est ici qu'intervient le test `if __name__ == '__main__': # (Programme Principal)`



# Premiers pas

## – Petites choses diverses à savoir pour débiter en Python

### ► Python Extension Proposal : PEP-8

- PEP 8 (pour Python Extension Proposal) est un ensemble de règles qui permet d'homogénéiser le code et d'appliquer de bonnes pratiques. L'exemple le plus connu est la guerre entre les développeurs à savoir s'il faut indenter son code avec des espaces ou avec des tabulations. La PEP8 tranche : ce sont les espaces qui gagnent, au nombre de 4.

# Premiers pas

## — Petites choses diverses à savoir pour débiter en Python

### ► Python Extension Proposal : PEP-8

#### **Encodage**

A préciser en première ligne de code si besoin. Par défaut l'UTF-8 est utilisé.

#### **Import**

A mettre en début de programme après l'encodage.

#### **Indentation Lignes**

Les lignes ne doivent pas dépasser 79 caractères. Séparer les fonctions et les classes à la racine d'un module par 2 lignes vides. Les méthodes par 1 ligne vide.

#### **Les espaces**

Les opérateurs doivent être entourés d'espaces. On ne met pas d'espace à l'intérieur des parenthèses, crochets ou accolades.

# Exercice dans la console Python

- Assigner les valeurs respectives 3, 5, 7 à trois variables a, b, c.
- Effectuer l'opération `a - b // c`. (division entière)
- Interpréter le résultat obtenu ligne par ligne.



**ib  
cegos**

# Importation de modules



# Importation de modules

- Un module est un fichier python que l'on veut importer.
- Un package est un dossier dans lequel on stock des fichiers pythons (modules).
- PYTHONPATH est une variable d'environnement qui définit les chemins supplémentaires où Python cherche des modules lorsqu'on les importe. Il est possible de modifier cette variable directement dans le système d'exploitation de façon constante ou de la modifier dans un script Python de façon temporaire.

# Importation de modules

## — Ordonner les lignes d'import :

- ▶ import de module standard
- ▶ import d'une partie du contenu d'un module standard
- ▶ import de module tierce
- ▶ import d'une partie du contenu d'un module tierce
- ▶ import de module personnel
- ▶ import d'une partie du contenu d'un module personnel

# Importation de modules

► import os

# import module de la librairie standard

► import sys

# on groupe car même type

► from itertools import islice

# import le contenu d'une partie d'un module

► from collections import namedtuple

# on groupe car même type

► import requests

# import librairie tierce partie

► import arrow

# on groupe car même type

► from django.conf import settings

# contenu d'une partie d'un module tierce

► from django.shortcuts import redirect

# on groupe car même type

# Importation de modules

- ▶ # import une fonction du projet
- ▶ `from myPackage.mySubPackage.myModule import myFunc`
- ▶ # import une classe du projet
- ▶ `from myPackage.mySubPackage.myModule import MyClass`



ib  
cegos

# Les variables



# Les variables

## – Règles de nommage

- ▶ Python ne possède pas de syntaxe particulière pour créer ou déclarer une variable.
- ▶ Il existe quelques règles usuelles pour la dénomination des variables.
- ▶ Les variables vont pouvoir stocker différents types de valeurs comme des nombres, des chaînes de caractères, des booléens, et plus encore.
- ▶ La casse est significative dans les noms de variables.
- ▶ Il y a 3 règles à respecter pour nommer une variable :
  - Le nom doit débuter par une lettre ou un underscore « \_ ».
  - Le nom d'une variable doit contenir que des caractères alphanumériques courant, sans espace ni caractères spéciaux ou accents.
  - On ne peut pas utiliser certains mots qui possède déjà une signification en Python. Ce sont les mots réservés.

# Les variables

## — Les types

### ► Integer

- `>>> a = 15`
- `>>> type(a)`

### ► String

- `>>> texte1 = 'Les œufs durs.'`
- `>>> texte2 = ' "Oui", répondit-il,'`
- `>>> texte3 = "j'aime bien "`
- `>>> print(texte1, texte2, texte3)`

### ► Float

- `>>> b = 16.0`
- `>>> type(b)`

### ► Boolean

- `>>> c = True`
- `>>> type(c)`

# Les variables

## – Les chaînes de caractères

- ▶ Une donnée de type string peut se définir en première approximation comme une suite quelconque de caractères. Dans un script python, on peut délimiter une telle suite de caractères, soit par des apostrophes (simple quotes), soit par des guillemets (double quotes).

## – Le caractère spécial « \ » (antislash)

- ▶ En premier lieu, il permet d'écrire sur plusieurs lignes une commande qui serait trop longue pour tenir sur une seule (cela vaut pour n'importe quel type de commande).
- ▶ À l'intérieur d'une chaîne de caractères, l'antislash permet d'insérer un certain nombre de codes spéciaux (sauts à la ligne, apostrophes, guillemets, etc.).
- ▶ Exemples :

```
>>> print("ma\tpetite\nchaine") # affiche ma      petite
                                # chaîne
```

# Les variables

## – Notion de formatage

- ▶ `%s` : string `>>> n = "Celine"`
- ▶ `%d` : decimal integer `>>> a = 42`
- ▶ `%f` : float `>>> print("nom : %s - age %d" %(n, a))`
- ▶ `%g` : generic number
  
- ▶ `.format()` `>>> print("nom : {1} - age : {0}".format(n, a))`
- ▶ `fstring` `>>> print(f"nom : {n} - age : {a}")`

# Les variables

## – Typage dynamique fort

- ▶ Python est fortement typé dynamiquement.
- ▶ Un typage fort signifie que le type d'une valeur ne change pas de manière inattendue. Chaque changement de type nécessite une conversion explicite.
- ▶ En Python, le typage dynamique signifie que le type des variables est déterminé automatiquement au moment de l'exécution (et non lors de l'écriture du code). En d'autres termes, il n'y a pas besoin de déclarer le type d'une variable à l'avance, Python le déduit automatiquement en fonction de la valeur qu'on lui assigne. Les valeurs attribuées n'ont pas de limite contrairement aux autres langages. La seule limite est la mémoire du PC.

# Les variables

## – Typage dynamique fort

```
>>> points = 3.2 # points est du type float (nombre décimal)
>>> print("Tu as " + points + " points !") # Génère une erreur de typage
>>> points = int(points) # points est maintenant du type int (entier),
                        # sa valeur est arrondie à l'unité inférieure (ici 3)
>>> print("Tu as " + points + " points !") # Génère une erreur de typage
>>> points = str(points) # points est maintenant du type str (string)
>>> print("Tu as " + points + " points !") # Plus d'erreur de typage, affiche "Tu as 3 points !"
```

# Les variables

## — Les collections

- ▶ Les chaînes de caractères que nous avons abordées constituaient un premier exemple de données composites. On appelle ainsi les structures de données qui sont utilisées pour regrouper de manière structurée des ensembles de valeurs.

### ▶ Listes

```
#Déclarer une list avec la possibilité de modifier son contenu  
list_data = [1, 2, 3, 4]
```

### ▶ Tuples

```
#Déclarer un tuple sans la possibilité de modifier son contenu  
tuple_data = (1, 2, 3, 4)
```



# Les variables

## — Les collections

### ► Sets

```
# Un set ne contient pas de doublon.  
# ici, le deuxième élément "pierre" sera ignoré.  
mon_set = {"pascal", "pierre", "paul", "pierre"}
```

### ► Dictionnaires

```
#Construire un dictionnaire  
dico = {}  
dico['computer'] = 'ordinateur'  
dico['mouse'] = 'souris'  
dico['keyboard'] = 'clavier'  
print("dico")  
dico = {'computer': 'ordinateur', 'keyboard': 'clavier', 'mouse': 'souris'}
```

# Exercice « ex\_collection.py »

## – Énoncé :

- ▶ Soit une chaîne de caractères comprenant, trois champs séparés par des caractères '--', comprenant à son tour trois champs séparés par des caractères ';' (un numéro d'étudiant, un nom et un prénom).
  - ▶ Faites un algorithme qui retourne un dictionnaire dont les clés sont les numéros d'étudiants et les valeurs sont, pour chaque numéro d'étudiant, une chaîne correspondant à la concaténation des prénom et nom de la personne.
- ▶ 213615200;BESNIER;JEAN--213565488;DUPOND;MARC--214665555;DURAND;JULIE



**ib**  
**cegos**

# Les blocs d'instructions



# Les blocs d'instructions

Avec Python, elles sont définies par la mise en page.

Vous devez utiliser les sauts à la ligne et l'indentation, mais en contrepartie vous n'avez pas à vous préoccuper d'autres symboles délimiteurs de blocs.

En définitive, Python vous force donc à écrire du code lisible, et à prendre de bonnes habitudes que vous conserverez lorsque vous utiliserez d'autres langages.

Python ne peut exécuter un programme que si sa syntaxe est parfaitement correcte. Dans le cas contraire, le processus s'arrête et vous obtenez un message d'erreur.

# Les blocs d'instructions

- Exemple

```
def power(num, x=1):  
    result = 1  
    for i in range(x):  
        result = result * num  
    return result
```

# Les structures répétitives



# Les structures répétitives

- L'instruction while

```
# boucle while  
x = 0  
while (x < 5):  
    print(x)  
    x += 1
```

- L'instruction for

```
# boucle for  
for x in range(5, 10):  
    print(x)
```

# Les structures conditionnelles





# Les structures conditionnelles

## — Instruction conditionnelle If

- ▶ Les boucles et conditions sont imbriquables.
- ▶ Les parenthèses ne sont pas obligatoires mais donne de la visibilité.
- ▶ Lorsqu'une seule instruction est exécutée, on peut la mettre sur la même ligne que le if, elif ou else.

```
# if, elif, else
if x < y:
    st = "x est plus petit que y"
elif x == y:
    st = "x est égale à y"
else:
    st = "x est plus grand que y"
print(st)
```

# Les structures conditionnelles

## — Opérateurs de comparaison et booléen

$X == Y$  # égale

$X != Y$  # différent

$X < Y$  # inférieur

$X > Y$  # supérieur

$X >= Y$  # supérieur ou égale

$X <= Y$  # inférieur ou égale

$X \text{ or } Y$  # ou

$X \text{ and } Y$  # et

$\text{not } X$  # négation de  $X$

- Les booléens ont la valeur True ou False avec une majuscule.

# Les structures conditionnelles

## – Condition ternaire

- ▶ Permet d'affecter une valeur selon une condition de type if else en une seule ligne de code.

## – Compréhension de liste

- ▶ La compréhension de liste permet de créer une liste à partir d'une boucle for combiné ou non avec une instruction de condition if (ou if else) sur une seule ligne.

# Les structures conditionnelles

- La structure match / case

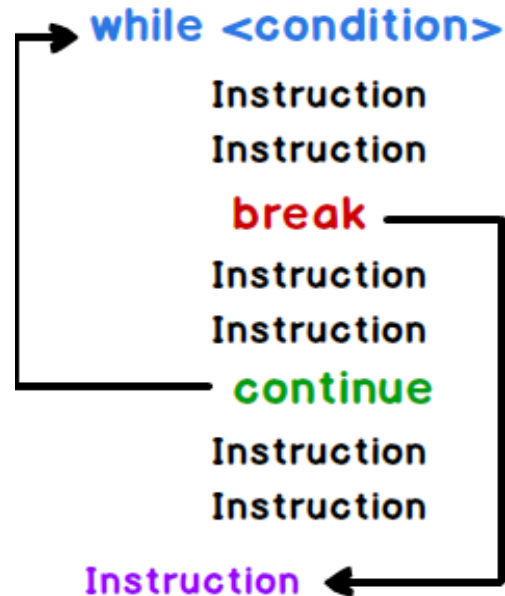
- La structure conditionnelle match / case est apparue depuis seulement la version 3.10.

```
# match case
match (x):
    case 1:
        y = 0
    case 2:
        y -= 1
    case _:
        y += x + 3
```

# Les structures conditionnelles

## – Les instructions continue et break

- ▶ Break est utilisé pour quitter une boucle for/while en cours d'exécution.
- ▶ Continue est utilisé pour ignorer la suite du bloc actuel et revenir à l'instruction for/while.



# Exercice « ex\_liste\_1 »

## — Énoncé :

- ▶ Écrivez un programme qui recherche le plus grand élément présent dans une liste donnée.
- ▶ Par exemple, si on l'appliquait à la liste [32, 5, 12, 8, 3, 75, 2, 15], ce programme devrait afficher :
  - le plus grand élément de cette liste à la valeur 75.

# Exercice « ex\_liste\_2 »

## — Énoncé :

- ▶ Ecrivez un programme qui donne la somme de tous les nombres supérieurs à 10 se trouvant dans une liste.
- ▶ Si on l'appliquait à la liste [32, 5, 12, 8, 3, 75, 2, 15], ce programme devrait afficher :
  - la somme demandée est 134

# Exercice « ex\_listes.py »

## — Enoncé :

- Ecrivez un programme qui analyse un par un tous les éléments d'une liste de mots (par exemple : ['Jean', 'Maximilien', 'Brigitte', 'Sonia', 'Jean-Pierre', 'Sandra']) pour générer deux nouvelles listes. L'une contiendra les mots comportant moins de 6 caractères, l'autre les mots comportant 6 caractères ou davantage.



ib  
cegos

# Les fonctions



# Les fonctions

## — Quelques fonctions prédéfinies / natives (builtin)

- ▶ `abs(x)`      # retourne la valeur absolue de x.
- ▶ `all(iterable)`    # retourne True si toutes les valeurs de l'itérable sont True.
- ▶ `any(iterable)`    # retourne True si au moins une valeur de l'itérable est True.
- ▶ `bin(int)`      # convertit nombre entier en chaîne de caractère binaire.
- ▶ `hex(int)`      # convertit nombre en valeur hexadécimale.
- ▶ `len(obj)`      # retourne la longueur de l'objet.
- ▶ `list(obj)`      # cast l'objet au format liste.
- ▶ `map(fct, obj)`    # applique une transformation à tous les éléments d'un itérable.
- ▶ `filter(fct, list)`    # filtre avec un prédicat les éléments d'un itérable et retourne un booléen.

# Les fonctions

## — Fonctions natives de l'objet str

- ▶ `str.capitalize()`    # retourne la string avec une majuscule en début de phrase
- ▶ `str.title()`    # retourne la string avec une majuscule en début de chaque mot
- ▶ `str.upper()`    # retourne la string en majuscule
- ▶ `str.lower()`    # retourne la string en minuscule
- ▶ `str.strip()`    # retourne la string str en supprimant les espaces avant et après le texte
- ▶ `str.count(x)`    # retourne le nombre d'occurrence de x dans str
- ▶ `str.endswith("x")` # retourne True si la str fini par 'x'
- ▶ `str.startswith("x")` # retourne True si la str commence par 'x'
- ▶ `str.find(x)`    # retourne l'index de la première occurrence de x (-1 si n'existe pas)

# Exercice « ex\_chaines.py »

## — Enoncé :

- ▶ chaine = "Retrouvez tous les mots dans une chaine de caractères"
- ▶ 1 - Retrouvez tous les mots dans une chaine de caractères sous forme de liste.
- ▶ 2 - Retrouver tous les mots qui se terminent par un caractère donné.
- ▶ 3 - Retrouver tous les mots qui commencent par un caractère donné.
- ▶ 4 - Retrouver tous les mots qui contiennent au moins 4 caractères.
- ▶ 5 - Retrouver tous les mots qui possèdent exactement n caractères.

# Exercice « ex\_fonctions\_natives.py »

## — Énoncé :

- ▶ Ecrire une boucle de programme qui demande à l'utilisateur d'entrer des notes d'élèves. La boucle se terminera seulement si l'utilisateur entre une valeur négative.
- ▶ Avec les notes ainsi entrées, construire progressivement une liste.
- ▶ Après chaque entrée d'une nouvelle note (et donc à chaque itération de la boucle), afficher le nombre de notes entrées, la note la plus élevée, la note la plus basse, la moyenne de toutes les notes (informations stockées dans un dictionnaire).

# Les fonctions

## — Fonction print()

- ▶ Elle permet d'afficher n'importe quel nombre de valeurs fournies en arguments (c'est-à-dire entre les parenthèses). Par défaut, ces valeurs seront séparées les unes des autres par un espace, et le tout se terminera par un saut à la ligne.

- `>>> print("Bonjour", "à", "tous", sep = "*")` Bonjour\*à\*tous
- `>>> print("Bonjour", "à", "tous", sep = "")` Bonjouràtous

# Les fonctions

## — Fonction input() : Interaction avec l'utilisateur

► Cette fonction provoque une interruption dans le programme courant. L'utilisateur est invité à entrer des caractères au clavier et à terminer avec <Enter>. Lorsque cette touche est enfoncée, l'exécution du programme se poursuit, et la fonction fournit en retour une chaîne de caractères correspondant à ce que l'utilisateur a saisi.

- `prenom = input("Entrez votre prénom : ")`
- `print("Bonjour,", prenom)`
- OU
- `print("Veuillez entrer un nombre positif quelconque : ")`
- `ch = input()`
- `nn = int(ch) # conversion de la chaîne en un nombre entier`
- `print("Le carré de", nn, "vaut", nn ** 2)`

# Exercice « ex\_input.py »

## — Énoncé :

- ▶ Ecrivez un programme qui convertisse en mètres par seconde une vitesse fournie par l'utilisateur en Km/h.
- ▶ Demander à l'utilisateur de rejouer. « Voulez-vous rejouer ? » Réponse possible oui ou non.



# Exercice « ex\_boucles\_imbriquees.py »

## — Énoncé :

- ▶ Ecrivez un programme qui affiche les 20 premiers résultats de la table de multiplication par un nombre entré par l'utilisateur.
- ▶ Demander à l'utilisateur de rejouer. « Voulez-vous rejouer ? » Réponse possible oui ou non.

# Les fonctions

## – Les fonctions

- L'approche efficace d'un problème complexe consiste souvent à le décomposer en plusieurs sous-problèmes plus simples, d'autre part, il arrivera souvent qu'une même séquence d'instructions doive être utilisée à plusieurs reprises dans un programme.

```
def nomDeLaFonction(liste de paramètres):  
    ...  
    bloc d'instructions  
    ...
```

## – Les paramètres des fonctions

- Une fonction peut avoir 0 ou plusieurs paramètres. Ces paramètres peuvent être des variables (int, string, list...) mais aussi d'autres fonctions.

# Les fonctions

## – Variables locales vs variables globales

- ▶ Lorsque nous définissons des variables à l'intérieur du corps d'une fonction, ces variables ne sont accessibles qu'à la fonction elle-même. On dit que ces variables sont des variables locales à la fonction.
- ▶ Les variables définies à l'extérieur d'une fonction sont des variables globales. Leur contenu est « visible » de l'intérieur d'une fonction, mais la fonction ne peut pas le modifier.

# Les fonctions

## — Fonction récursive

- ▶ Une fonction récursive est une fonction qui s'appelle elle-même. Attention à bien mettre une condition pour sortir de l'appel récursif, sinon on provoque une boucle infinie.
- ▶ Exemple la fonction factorielle(x)

```
def factorielle(n):  
    if n < 2:  
        return 1  
    else:  
        return n * factorielle(n-1)
```

```
def factorielle(n):  
    return 1 if n < 2 else n * factorielle(n-1)
```

# Exercice « ex\_fonction.py »

## — Énoncé :

- ▶ Définissez une fonction `maximum(n1, n2, n3)` qui renvoie le plus grand de 3 nombres `n1`, `n2`, `n3` fournis en arguments. Par exemple, l'exécution de l'instruction :
  - `print(maximum(2, 5, 4))` doit donner le résultat : 5.

# Exercice « ex\_fonctions.py »

## — Énoncé :

- ▶ Ecrire une fonction cube qui retourne le cube de son argument.
- ▶ Ecrire une fonction volumeSphere qui calcule le volume d'une sphère de rayon r fourni en argument et qui utilise la fonction cube.
- ▶ Tester la fonction volumeSphere par un appel dans le programme principal.

# Les fonctions

## — Fonction Lambda

- Pour Python, c'est la seule façon d'écrire une fonction anonyme. Ceci est particulièrement utile pour la programmation fonctionnelle. En effet, une fonction peut être directement écrite dans un appel de fonction sans avoir à la définir au préalable.

- `>>> list(map(lambda x: x ** 2, range(10)))`
- `[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]`

# Les fonctions

## — Fonction Lambda

► Bien que les fonctions lambda soient utilisées dans le but de créer des fonctions anonymes, on peut décider tout de même de leur donner un nom :

- `>>> f = lambda x: x ** 2`
- `>>> f(5)`
- `25`
- `>>> list(map(f, range(10)))`
- `[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]`

► Syntaxe pour passer une variable à l'exécution : `lambda x: monCalculAvecX`

► Syntaxe pour passer une variable à la création : `lambda x = x: monCalculAvecX`



# TP validation des acquis 1



# TP validation des acquis 1

- Créer un petit programme interactif de gestion de notes pour un élève. Le programme devra permettre de :
  - ▶ Demander à l'utilisateur combien de notes il veut entrer.
  - ▶ Saisir ces notes une par une (entre 0 et 20) et les ajouter dans une liste.
  - ▶ Calculer la moyenne de ces notes via une fonction nommée "calcul\_moyenne".
  - ▶ Afficher un message différent selon la moyenne :
    - Moins de 10 : "En difficulté"
    - Entre 10 et 15 : "Peut mieux faire"
    - 15 ou plus : "Très bien"

# TP validation des acquis 1

## – Résultat attendu :

Combien de notes voulez-vous entrer ? 3

Entrez la note 1 : 12

Entrez la note 2 : 8

Entrez la note 3 : 15

Moyenne : 11.67

Appréciation : Peut mieux faire

ib  
cegos

# Les docstrings



# Les docstrings

## – L'utilisation du docstring

- ▶ C'est toujours une bonne idée d'écrire de la documentation pour vos fonctions, classes et modules. Et l'une des grandes caractéristiques de Python est qu'il vous permet de le faire dans du code. La documentation du code de votre programme, ainsi que du code de Python, est directement disponible lorsque votre programme est exécuté ou utilisé par un autre développeur.

- `>>> print(any.__doc__)`

## – Cela fonctionne pour les fonctions mais aussi pour les modules et les classes.

- ▶ `>>> import collections`

- ▶ `>>> print(collections.__doc__)`

```
def maFonction(arg1, arg2=None):
    """maFonction(arg1, arg2=None) --> Ne fait rien de spéciale.

    Parameters:
    arg1: premier argument standard.
    arg2: deuxieme argument avec valeur par default à None.
    """
    print(arg1, arg2)

def main():
    print(maFonction.__doc__)
```

# Les docstrings

- Les chaînes de documentation doivent toujours être placées entre des triples guillemets ou triples apostrophes, même si elles ne font qu'une ligne.
- La première ligne de votre docstring doit résumer la fonction, la classe ou le module et son objectif principal.
- Pour les paquets et les modules, listez les classes et les sous-modules importants qu'ils contiennent, ainsi que les exceptions personnalisées que le développeur doit connaître.
- Pour les classes, listez les méthodes et les informations importantes.

# Les docstrings

- Pour les fonctions, il y a une variété d'éléments importants à lister :
- Assurez-vous que votre docstring liste et explique chacun des arguments (un par ligne), y compris les facultatifs.
- Si la fonction renvoie une valeur, expliquez-la dans la docstring. Sinon, la convention habituelle consiste à économiser de l'espace en ne l'indiquant pas. Si la fonction soulève des exceptions, veuillez à les mentionner également.

# Les docstrings

- Il existe plusieurs standards de docstrings :

- ▶ PEP257 (Style officiel de Python)

- Première phrase : résumé en une ligne (phrase complète).
    - Une ligne vide.
    - Description détaillée des paramètres et du retour.
    - Pas de sections particulières.



# Les docstrings

## — Il existe plusieurs standards de docstrings :

### ► Style Google

- Première phrase : résumer en une ligne.
- Utilise des sections nommées Args, Attributs, Returns et Raises suivi de « : ».
- Des lignes vides entre chaque section.
- Les variables sont décrites avec leurs types entre parenthèses sur une ligne. La description est séparée du reste par un « : ».

# Les docstrings

## — Il existe plusieurs standards de docstrings :

### ► Style Numpy

- Première phrase : résumé en une ligne.
- Utilise les sections Parameters et Returns.
- Une ligne vide entre chaque section.
- Chaque section est soulignée par des tirets.
- Les variables sont indiquées avec leur type sur une ligne (séparé par un « : »), et une ligne décalée d'une indentation en dessous pour leur description.



**ib  
cegos**

# Les exceptions



# Les exceptions

Toutes les erreurs qui se produisent lors de l'exécution d'un programme Python sont représentées par une exception. Une exception est un objet qui contient des informations sur le contexte de l'erreur. Lorsqu'une exception survient et qu'elle n'est pas traitée alors elle produit une interruption du programme et elle affiche sur la sortie standard un message ainsi que la pile des appels (stacktrace). La pile des appels, présente dans l'ordre, la liste des fonctions et des méthodes qui étaient en cours d'appel au moment où l'exception est survenue.

# Les exceptions

Parfois un programme est capable de traiter le problème à l'origine de l'exception. Par exemple si le programme demande à l'utilisateur de saisir un nombre et que celui-ci saisit une valeur erronée, le programme peut simplement demander à l'utilisateur de saisir une autre valeur plutôt que de faire échouer le programme.

```
try:  
    pass  
except e:  
    pass  
else:  
    pass  
finally:  
    pass
```

# Exercice « ex\_exception.py »

## – Énoncé :

- ▶ Ecrire un programme qui demande à l'utilisateur de saisir des entiers un par un (on saisira le mot fin pour finir la saisie des nombres) puis à l'aide de parcours successifs de la liste effectuer les actions suivantes :
- ▶ 1) Afficher la liste.
- ▶ 2) Afficher la liste en colonne de manière à afficher l'index et le contenu.
- ▶ 3) Créer une nouvelle liste qui sera chaque élément de la liste multiplié par 3 en utilisant une fonction lambda.
- ▶ 4) Obtenir le plus grand nombre de la liste.
- ▶ 5) Obtenir le plus petit nombre de la liste.
- ▶ 6) Obtenir la quantité de nombre pair présents dans la liste.
- ▶ 7) Calculer la somme de tous les nombres impairs de la liste.
- ▶ Le programme doit gérer les exceptions au niveau de la saisie des données de l'utilisateur.

ib  
cegos

**Le débogage**



# Le débogage

## — Module pdb

► Pour déboguer lorsque nous ne pouvons pas le faire via un IDE telle de VS Code ou PyCharm, nous pouvons déboguer via la console Python (cf : <https://docs.python.org/fr/3/library/pdb.html>).

► Avant la v3.7 nous devons écrire dans notre code les lignes suivantes :

- `import pdb` # import du module pdb
- `pdb.set_trace()` # instruction du module pdb

► A partir de la version 3.7 nous devons écrire :

- `breakpoint()` # équivalent au `pdb.set_trace()` mais intégré à python

► Le programme s'arrête quand `set_trace()` ou `breakpoint()` est trouvé.



# Le débogage

► A partir de ce moment on utilise des commandes dans la console du terminal telle que :

- `c(ontinue)` : Continuer l'exécution
- `q(uit)` : Quitter le débogueur / l'exécution
- `n(ext)` : Passer à la ligne suivante dans la même fonction
- `s(tep)` : Passer à la ligne suivante dans cette fonction ou une fonction appelée
- `l(list)` : Liste le code à la position courante
- `u(p)` : Monte à la pile d'appel
- `d(own)` : Descend à la pile d'appel
- `bt` : Affiche la pile d'appel
- `a(rgs)` : Affiche la liste d'arguments de la fonction courante.
- `!instruction_python` : Remplacer `instruction_python` par l'instruction voulue pour l'exécuter.
- `b(reak)` : Sans argument, liste tous les arrêts, incluant pour chaque point d'arrêt, le nombre de fois qu'un point d'arrêt a été atteint, le nombre d'ignore, et la condition associée le cas échéant.

# Le débogage

- ▶ A partir de la console python on peut exécuter la commande :
  - `pdb.run('mymodule.test()')`      # fichier mymodule.py, fonction test()
- ▶ Il est aussi possible de déboguer à partir de l'invite de commande :
  - `python -m pdb myscript.py`
- ▶ L'utilisation de l'instruction `pdb.post_mortem()` dans un `except` permet de faire un débogage après qu'une exception est produite.

ib  
cegos

**Les fichiers**



# Les fichiers

## — Le module OS

► Le module OS contient de nombreuses fonctions intéressantes pour l'accès au système d'exploitation.

- `>>> import os`
- `>>> os.getcwd()`
- `>>> os.chdir("C:\\Users\\User\\Documents")`
- `>>> os.path.dirname(__file__)`

# Les fichiers

## — Les fichiers

- ▶ Avec Python, l'accès aux fichiers est assuré par l'intermédiaire d'un objet-interface particulier, que l'on appelle objet-fichier. On crée cet objet à l'aide de la fonction intégrée `open()`. Celle-ci renvoie un objet doté de méthodes spécifiques, qui vous permettront de lire et écrire dans le fichier.

### Ecriture

```
>>> objetfichier = open('Monfichier','a')
>>> objetfichier.write('Bonjour, fichier !')
>>> objetfichier.write("Quel beau temps!")
>>> objetfichier.close()
```

### Lecture

```
>>> of = open('Monfichier', 'r')
>>> t = of.read()
>>> print(t)
Bonjour, fichier !Quel beau temps !
>>> of.close()
```

# Les fichiers

- La fonction `open()` attend deux arguments minimums, qui doivent tous deux être des chaînes de caractères. Le premier argument est le nom du fichier à ouvrir, et le second est le mode d'ouverture.
  - ▶ 'a' : indique qu'il faut ouvrir ce fichier en mode « ajout » (append), ce qui signifie que les données à enregistrer doivent être ajoutées à la fin du fichier, à la suite de celles qui s'y trouvent éventuellement déjà.
  - ▶ 'w' : utilisé aussi pour l'écriture mais lorsqu'on utilise ce mode « écriture » (write), Python crée toujours un nouveau fichier (vide), et l'écriture des données commence à partir du début de ce nouveau fichier. S'il existe déjà un fichier de même nom, celui-ci est effacé au préalable.
  - La méthode `write()` réalise l'écriture proprement dite. Les données à écrire doivent être fournies en argument. Chaque nouvel appel de `write()` (en mode a) continue l'écriture à la suite de ce qui est déjà enregistré.

# Les fichiers

## ► 'r' : Ouverture en lecture.

- La méthode `read()` (en mode `r`) permet de lire le contenu d'un fichier dans son ensemble.
- La méthode `readline()` (en mode `r`) lit qu'une ligne par appel de cette instruction.
- La méthode `readlines()` (en mode `r`) lit les lignes individuellement dans une liste.
- La méthode `seek(x)` (en mode `r`) permet de replacer le curseur à la position `x` voulue.
- La méthode `close()` referme le fichier dans n'importe qu'elle mode.

## ► 'r+' : c'est un mode lecture et écriture. Le fichier doit exister au préalable sinon une exception se produit. Le curseur est placé en début de fichier et le fichier n'est pas effacé.

# Exercice « ex\_fichier.py »

## — Enoncé :

- Ecrivez un script qui génère automatiquement un fichier texte contenant les tables de multiplication de 2 à 30 (chacune d'entre elles incluant les termes de 1 à 20 seulement).



# Exercice « ex\_multi\_fichiers.py »

## – Enoncé :

- ▶ A partir de deux fichiers préexistants A et B, construisez un fichier C qui contienne alternativement un élément de A, un élément de B, un élément de A... et ainsi de suite jusqu'à atteindre la fin de l'un des deux fichiers originaux.
- ▶ Complétez ensuite C avec les éléments restant sur l'autre.

# Exercice « ex\_fichier\_mini\_bdd.py »

## — Énoncé :

- ▶ Mini Système BDD à partir d'un fichier qu'on nommera utilisateurs.txt et qui se situera dans le même dossier que ce script.
- ▶ Dans un premier temps :
  - Créer un dictionnaire qui permettra d'enregistrer en clé le nom et en valeur l'age et la taille de l'utilisateur.
  - Créer une fonction inscription pour saisir les données utilisateurs, les inscrire dans le dictionnaire et poser la question si on veut continuer à saisir un utilisateur.
  - Créer une fonction consultationTotale qui permet de voir les données des utilisateurs enregistrés dans le dictionnaire.
  - Créer une fonction consultation qui permettra de consulter les données d'un utilisateur.
  - Créer un menu pour choisir entre quitter, inscription ou consultation.

# Exercice « ex\_fichier\_mini\_bdd.py »

## — Enoncé :

### ► Dans un deuxième temps :

- Créer une fonction enregistrer qui enregistrera les infos utilisateurs dans le fichier nommé utilisateurs.txt
- On utilisera le caractère séparateur @ pour séparer la clé des valeurs du dictionnaire, et le caractère # pour séparer les données constituant ces valeurs. Exemple Juliette@18#1.68
- Créer une fonction lecture qui permettra de lire le fichier utilisateurs.txt et d'inscrire les données lues dans le dictionnaire.
- Modifier le menu pour ajouter les fonctions enregistrement et lecture.

# Exercice « ex\_fichier\_mini\_bdd.py »

## — Enoncé :

### ► Exemple de menu :

- (L) Lecture
- (I) Inscription
- (C) Consultation par nom
- (T) Consultation totale
- (E) Enregistrement
- (Q) Quitter



**ib  
cegos**

# Les expressions régulières



# Les expressions régulières

- Pour faire des tests, utilisez le site <https://regex101.com/>
- Les expressions régulières représentent les motifs qu'on recherche dans une chaîne de caractères. Par exemple, on peut chercher dans un fichier :
  - ▶ Les lignes qui contiennent un numéro de téléphone
  - ▶ Les lignes qui débutent par une adresse IP

# Les expressions régulières

- On peut chercher dans une chaîne de caractère :
  - ▶ Une séquence qui commence par « ex » et se termine par « le »
  - ▶ Les mots de quatre caractères de long
  - ▶ Les mots sans majuscules
- On utilise les symboles suivants qui ont une signification spécifique chacune !
  - ▶ . ^ \$ \* + ? { } [ ] \ ( )

# Les expressions régulières

## — Les quantificateurs

- ▶+ : Le résultat peut contenir une ou plusieurs occurrences du caractère qui le précède.
- ▶\* : Le résultat peut contenir zéro ou plusieurs occurrences du caractère qui le précède.
- ▶? : Le résultat peut inclure zéro ou une occurrence du caractère qui le précède.
- ▶{x} : Précise le nombre x de répétitions des chaînes de caractères qui le précède.
- ▶[ ] : Obtient le résultat sur plusieurs occurrences.
- ▶() : Créer des groupes.
- ▶| : Correspond à l'expression OU. On peut s'en servir pour trouver une correspondance entre plusieurs expressions.
- ▶- : Sert à regrouper des caractères dans un intervalle donné.



# Les expressions régulières

## — Les métacaractères

- ▶ `^` : Précise que le résultat doit commencer par la chaîne de caractères qui le suit.
- ▶ `$` : Précise que le résultat doit se terminer par la chaîne de caractère qui le précède.
- ▶ `.` : Correspond à n'importe quel caractère sauf le saut de ligne.

## — Les tokens

- ▶ `\d` : `[0-9]`
- ▶ `\w` : `[a-zA-Z0-9_]`
- ▶ `\s` : correspond à tous les espaces dans la chaîne de caractères
- ▶ `\.` : juste les points
- ▶ `\b` : frontière des mots

# Les expressions régulières

## — Les fonctions usuelles

- ▶ `re.compile(motif)` : Compile un motif (expression régulière) utilisable par les autres fonctions telle que `search`, `findall`, etc ...
- ▶ `re.search(motif, string)` : Renvoie la première correspondance du motif trouvée dans la `string`.
- ▶ `re.findall(motif, string)` : Renvoie une liste de toutes les correspondances du motif trouvées dans la `string`.
- ▶ `re.match(motif, string)` : Renvoie la correspondance trouvée si zéro ou plus caractères en début de `string` correspondent au motif.
- ▶ `re.sub(motif, chaîne, string)` : Remplace par `chaîne` les occurrences trouvées dans la `string` (sans chevauchement) et renvoie le résultat. Si le motif n'est pas trouvé, `string` est renvoyée inchangée.

## Exercice « ex\_regex.py »

- Dans une chaîne de caractères donnée :
  - ▶ 1 - Retrouvez tous les mots dans une chaîne de caractères.
  - ▶ 2 - Retrouver tous les mots qui se terminent par un caractère donné.
  - ▶ 3 - Retrouver tous les mots qui commencent par un caractère donné.
  - ▶ 4 - Retrouver tous les mots qui contiennent au moins trois caractères.
  - ▶ 5 - Retrouver tous les mots qui possèdent exactement n caractères.



**ib**  
**cegos**

**Les classes**



# Les classes

## – Orienté objet : ça veut dire quoi ?

- Globalement, les langages de programmation objet implémentent le paradigme de programmation orientée objet (POO). Ce paradigme consiste en la réunion des données et des traitements associées à ces données au sein d'entités cohérentes appelées objets. Python est un langage objet composé de classes. Une classe représente un « moule » permettant de créer des objets (instances), et regroupe les attributs et méthodes communes à ces objets.

# Les classes

## – Les classes

- ▶ L'orienté objet facilite beaucoup dans la conception, la maintenance, la réutilisabilité des éléments (objets). Le paradigme de POO permet de tirer profit de classes parents et de classes enfants (phénomène d'héritage), etc.
- ▶ Tout objet donné possède deux caractéristiques :
  - Son état courant (attributs)
  - Son comportement (méthodes)
- ▶ En approche orienté objet on utilise le concept de classe, celle-ci permet de regrouper des objets de même nature.
- ▶ Une classe est un moule (prototype) qui permet de définir les attributs (variables) et les méthodes (fonctions) de tous les objets de cette classe.
- ▶ Les classes sont les principaux outils de la POO. Ce type de programmation permet de structurer les logiciels complexes en les organisant comme des ensembles d'objets qui interagissent entre eux et avec le monde extérieur.

# Les classes

## — Attributs de classe

- Une classe peut également avoir des attributs. Pour cela, il suffit de les déclarer dans le corps de la classe. Les attributs de classe sont accessibles depuis la classe elle-même et sont partagés par tous les objets. Si un objet modifie un attribut de classe, cette modification est visible de tous les autres objets. Les attributs de classe sont le plus souvent utilisés pour représenter des constantes.

## — Méthodes de classe

- Tout comme il est possible de déclarer des attributs de classe, il est également possible de déclarer des méthodes de classe. Pour cela, on utilise le décorateur `@classmethod`. Comme une méthode de classe appartient à une classe, le premier paramètre correspond à la classe. Par convention, on appelle ce paramètre `cls` pour préciser qu'il s'agit de la classe et pour le distinguer de `self`.

# Les classes

## – Méthode statique

- Une méthode statique est une méthode qui appartient à la classe mais qui n'a pas besoin de s'exécuter dans le contexte d'une classe. Autrement dit, c'est une méthode qui ne doit pas prendre le paramètre `cls` comme premier paramètre. Pour déclarer une méthode statique, on utilise le décorateur `@staticmethod`. Les méthodes statiques sont des méthodes utilitaires très proches des fonctions mais que l'on souhaite déclarer dans le corps d'une classe.



# Les classes

- Exemple de classe

```
#Class avec COstructor
class Velo:
    roues = 2

    def __init__(self, marque, prix, poids):
        self.marque = marque
        self.prix = prix
        self.poids = poids

    def rouler(self):
        print("Wouh, ça roule mieux avec un vélo {} !".format(self.marque))
```

# Les classes

## – Quelques remarques importantes

- ▶ Tous les attributs et méthodes des classes Python sont « publics » au sens de C++, parce que « nous sommes tous des adultes ! » (citation de Guido von Rossum, créateur de Python).
- ▶ Le constructeur d'une classe est une méthode spéciale qui s'appelle `__init__()`.
- ▶ En Python, on n'est pas tenu de déclarer tous les attributs de la classe comme d'autres langages : on peut se contenter de les initialiser dans le constructeur !
- ▶ Toutes les méthodes prennent une variable `self` comme premier argument. Cette variable est une référence à l'objet manipulé.
- ▶ Python supporte l'héritage simple et l'héritage multiple. La création d'une classe fille est relativement simple, il suffit de préciser entre parenthèses le nom de la classe mère lors de la déclaration de la classe fille.

# Exercice « ex\_cercle\_cylindre.py »

## — Énoncé :

- ▶ Définissez une classe Cercle(). Les objets construits à partir de cette classe seront des cercles de tailles variées. En plus de la méthode constructeur (qui utilisera donc un paramètre rayon), vous définirez une méthode surface(), qui devra renvoyer la surface du cercle.
- ▶ Définissez ensuite une classe Cylindre() dérivée de la précédente. Le constructeur de cette nouvelle classe comportera les deux paramètres rayon et hauteur.
- ▶ Vous y ajouterez une méthode volume() qui devra renvoyer le volume du cylindre (rappel :  $\text{volume d'un cylindre} = \text{surface de section} \times \text{hauteur}$ ).

# Exercice « ex\_compte\_bancaire.py »

## — Enoncé :

- ▶ Définissez une classe `CompteBancaire()`, qui permette d'instancier des objets tels que `compte1`, `compte2`, etc.
- ▶ Le constructeur de cette classe initialisera deux attributs d'instance `nom` et `solde`, avec les valeurs par défaut 'Dupont' et 1000.
- ▶ Trois autres méthodes seront définies :
  - `depot(somme)` permettra d'ajouter une certaine somme au solde.
  - `retrait(somme)` permettra de retirer une certaine somme du solde.
  - `affiche()` permettra d'afficher le nom du titulaire et le solde de son compte.

# Exercice « ex\_compte\_epargne.py »

## – Enoncé :

- ▶ Ecrivez un nouveau script qui récupère le code de (compte bancaire) en l'important comme un module.
- ▶ Définissez-y une nouvelle classe `CompteEpargne()`, dérivant de la classe `CompteBancaire()` importée, qui permette de créer des comptes d'épargne rapportant un certain intérêt au cours du temps.
- ▶ Pour simplifier, nous admettrons que ces intérêts sont calculés tous les mois.
- ▶ Le constructeur de votre nouvelle classe devra initialiser un taux d'intérêt mensuel par défaut égal à 0,3 %. Une méthode `changeTaux(valeur)` devra permettre de modifier ce taux à volonté.
- ▶ Une méthode `capitalisation(nombreMois)` devra :
  - Afficher le nombre de mois et le taux d'intérêt pris en compte.
  - Calculer le solde atteint en capitalisant les intérêts composés, pour le taux et le nombre de mois qui auront été choisis.
  - Redéfinir la fonction d'affichage héritée pour ajouter le taux mensuel du compte épargne.

# Exercice « ex\_jeu\_de\_cartes.py »

## – Enoncé :

- ▶ Définissez une classe `JeuDeCartes()` permettant d'instancier des objets dont le comportement soit similaire à celui d'un vrai jeu de cartes. La classe devra comporter au moins les quatre méthodes suivantes :
- ▶ Méthode constructeur : Création et remplissage d'une liste de 52 éléments. Ces éléments sont des tuples contenant la couleur (Coeur, Trèfle, Pique, Carreau) et la valeur (2, 3, 4, 5, 6, 7, 8, 9, 10, Valet, Dame, Roi, As) de chacune des cartes. Dans une telle liste, l'élément (Valet , Pique) désigne donc le Valet de Pique, et la liste terminée doit être sous la forme : [(2, Coeur), (3, Coeur), ....., (As, Carreau)].

# Exercice « ex\_jeu\_de\_cartes.py »

## — Énoncé :

- ▶ Méthode `nom_carte()` : cette méthode doit renvoyer, sous la forme d'une chaîne, l'identité d'une carte quelconque dont on lui a fourni le tuple descripteur en argument. Par exemple, l'instruction : `print(jeu.nom_carte((valeur, couleur)))` doit provoquer l'affichage de : 2 de Carreau
- ▶ Méthode `melanger()` : Cette méthode sert à mélanger les éléments de la liste contenant les cartes, quel qu'en soit le nombre.
- ▶ Méthode `tirer()` : lorsque cette méthode est invoquée, la première carte de la liste est retirée du jeu. Le tuple contenant sa valeur et sa couleur est renvoyé au programme appelant. Si cette méthode est invoquée alors qu'il ne reste plus aucune carte dans la liste, il faut alors renvoyer `None` au programme appelant.

# Exercice « ex\_jeu\_a\_et\_b.py »

## — Enoncé :

- ▶ Complément de l'exercice précédent : définir deux joueurs A et B.
- ▶ Instancier deux jeux de cartes (un pour chaque joueur) et les mélanger.
- ▶ Ensuite, à l'aide d'une boucle, tirer 52 fois une carte de chacun des deux jeux et comparer leurs valeurs. Si c'est la première des deux qui a la valeur la plus élevée, on ajoute un point au joueur A. Si la situation contraire se présente, on ajoute un point au joueur B. Si les deux valeurs sont égales, on passe au tirage suivant.
- ▶ Au terme de la boucle, comparer les comptes de A et B pour déterminer le gagnant.





**ib  
cegos**

# **Les classes avancées**



# Les classes avancées

## – Polymorphisme

- ▶ Le polymorphisme est un mécanisme important dans la programmation objet. Il permet de modifier le comportement d'une classe fille par rapport à sa classe mère. Le polymorphisme permet d'utiliser l'héritage comme un mécanisme d'extension en adaptant le comportement des objets.

# Les classes avancées

## – Propriétés

- ▶ Les propriétés permettent de définir des comportements de 'getter' et 'setter' sur les méthodes d'une classe. Cela nous permet également d'appeler une méthode sans avoir besoin d'utiliser les parenthèses.
- ▶ Pour créer une propriété de type getter, on utilise le décorateur `@property` sur une méthode.
- ▶ Pour ajouter un 'setter', il faut décorer la méthode du même nom avec un décorateur ayant la syntaxe `@nom_method.setter`. Si aucun setter n'est défini alors elle n'est pas modifiable, par conséquent c'est une constante.
- ▶ Il existe aussi un 'deleter' qui permet de supprimer la propriété. La syntaxe est la même que le setter en remplaçant « setter » par « deleter ».

# Les classes avancées

## – Encapsulation

- ▶ En Python, il n'existe pas de mécanisme dans le langage qui nous permettrait de gérer la visibilité. Par contre, il existe une convention dans le nommage. Une méthode ou un attribut dont le nom commence par un underscore « \_ » (sunder) est considéré comme privé. Il est donc déconseillé d'accéder à un tel attribut ou d'appeler une telle méthode depuis l'extérieur de l'objet.

# Les classes avancées

## — Masquer des attributs / méthodes à une classe fille

► Parfois, on ne souhaite pas qu'une méthode puisse être redéfinie ou qu'un attribut puisse être modifié dans une classe fille. Pour cela, il suffit que le nom de la méthode ou de l'attribut commence par deux underscore « `_` » (dunder). Nous avons vu précédemment que Python n'a pas de mécanisme pour contrôler la visibilité des éléments d'une classe. Par convention, les développeurs signalent par un `sunder` le statut privé d'un attribut ou d'une méthode. Par contre le recours à un `dunder` a un impact sur l'interpréteur qui renomme la méthode ou l'attribut sous la forme :

- `__<nom de la classe>__<nom>`

# Les classes avancées

## – Destructeur

- ▶ Les destructeurs sont appelés lorsqu'un objet est détruit. En Python, les destructeurs ne sont pas aussi nécessaires que en C++, car Python dispose d'un ramasse-miettes qui gère automatiquement la gestion de la mémoire.
- ▶ La méthode `__del__()` est une méthode appelée destructeur en Python. Elle est appelée lorsque toutes les références à l'objet ont été supprimées, c'est-à-dire lorsqu'un objet est nettoyé via le mot clé « `del` » ou à la fin du programme par exemple.

# Les classes avancées

## – Fermer la liste des attributs

- ▶ Il est possible déclarer la liste finie des attributs. Pour cela, on déclare un attribut de classe appelé `__slots__`.
- ▶ Lorsque `__slots__` n'est pas indiqué, on peut ajouter des attributs inexistants à la création d'une classe. Si on l'indique, cela n'est plus possible et génère une erreur de type `AttributeError`.
- ▶ L'attribut `__slots__` a également une utilité pour l'optimisation du code. Comme on indique à l'interpréteur le nombre exact d'attributs, il peut optimiser l'allocation mémoire pour un objet de ce type.

# Les classes avancées

## – Interface

- ▶ En Python, le concept d'interface n'existe pas de manière explicite comme en Java ou C#. Mais il est possible de créer des interfaces en utilisant des classes abstraites grâce au module « abc » (Abstract Base Classes).



# Exercice « ex\_propriete\_encapsulation.py »

## — Créez une classe Animal :

- ▶ Attributs privés : `_nom`, `_age`
- ▶ Propriétés pour accéder et modifier nom et age avec vérification (âge  $\geq 0$ ).
- ▶ Une méthode `parler()` qui affiche "L'animal fait un bruit."
- ▶ Un destructeur qui affiche : "{nom} a été retiré du zoo."

## — Créez deux classes filles :

- ▶ Chien, redéfinit `parler()` → affiche "{nom} aboie : Woof!"
- ▶ Chat, redéfinit `parler()` → affiche "{nom} miaule : Miaou!"

## — Créez une fonction `faire_parler(animal)` qui prend un objet Animal et appelle sa méthode `parler()`.

# Les classes avancées

## — Classe abstraite

- ▶ Les classes abstraites sont des classes qui ne peuvent pas être instanciées, elles contiennent une ou plusieurs méthodes abstraites. C'est un modèle pour d'autres classes qui héritent d'un ensemble de méthodes et de propriétés. Une méthode abstraite est une méthode déclarée, mais qui n'a pas d'implémentation. Toutefois, une classe abstraite nécessite des sous-classes qui fournissent des implémentations pour les méthodes abstraites. L'abstraction est très utile lors de la conception de systèmes complexes pour limiter la répétition et assurer la cohérence.
- ▶ En Python, le module `abc` permet de simuler ce type d'approche. Le nom de ce module est la contraction de `abstract base classes`. Ce module fournit une méta-classe appelée « `ABC` » qui permet de transformer une classe Python en classe abstraite.

# Les classes avancées

- Méthode abstraite

- Pour créer une méthode abstraite il suffit d'utiliser le décorateur `@abstractmethod` et ne pas mettre de code à l'intérieur de celle-ci (seulement `pass`).

- Il est possible de combiner les autres décorateurs (`@property`, `@classmethod`, `@staticmethod`) avec `@abstractmethod` pour créer des propriétés, méthodes de classe ou méthodes statiques abstraites.

## Exercice « ex\_abstraction.py »

- On souhaite créer une application de gestion de paiements pour différents moyens de paiement (comme Carte Bancaire, PayPal, ou Cryptomonnaie).
- Chaque moyen de paiement doit :
  - ▶ pouvoir effectuer un paiement d'un montant donné,
  - ▶ fournir une méthode concrète pour vérifier la connexion au service.
- On va utiliser une classe abstraite `MoyenPaiement` avec une méthode abstraite `payer(montant)` et une méthode concrète `verifier_connexion()`.

# Les classes avancées

## – Définition de la surcharge

- ▶ La surcharge (overloading) en programmation désigne la capacité d'une fonction ou d'un opérateur à fonctionner différemment selon le type ou le nombre d'arguments.
- ▶ En Python, contrairement à d'autres langages comme Java ou C++, la surcharge des fonctions n'est pas directement prise en charge, car Python ne permet pas d'avoir plusieurs fonctions portant le même nom avec des signatures différentes. Cependant, il existe des moyens d'implémenter un comportement similaire.

## – La surcharge des fonctions

- ▶ Python ne permet pas plusieurs définitions d'une fonction avec le même nom. Si on définit plusieurs fois une fonction avec le même nom, seule la dernière définition est conservée. Pour contourner cette limitation, on peut utiliser des valeurs par défaut ou `*args` pour gérer un nombre variable d'arguments.

# Les classes avancées

## – La surcharge d'opérateur

- ▶ Python permet la surcharge des opérateurs en redéfinissant des méthodes spéciales (dunder methods ou méthodes magiques).

## – Méthodes de comparaison :

- ▶ `__eq__(self, other)` : `==` (égalité)
- ▶ `__ne__(self, other)` : `!=` (différent)
- ▶ `__lt__(self, other)` : `<` (inférieur)
- ▶ `__le__(self, other)` : `<=` (inférieur ou égal)
- ▶ `__gt__(self, other)` : `>` (supérieur)
- ▶ `__ge__(self, other)` : `>=` (supérieur ou égal)

# Les classes avancées

## – Méthodes mathématiques :

- ▶ `__add__(self, other)` : `+` (addition)
- ▶ `__sub__(self, other)` : `-` (soustraction)
- ▶ `__mul__(self, other)` : `*` (multiplication)
- ▶ `__truediv__(self, other)` : `/` (division)
- ▶ `__floordiv__(self, other)` : `//` (division entière)
- ▶ `__mod__(self, other)` : `%` (modulo)
- ▶ `__pow__(self, other)` : `**` (puissance)
- ▶ `__matmul__(self, other)` : `@` (multiplication matricielle)
- ▶ `__and__(self, other)` : `&` (Et logique)
- ▶ `__or__(self, other)` : `|` (Ou logique)

# Les classes avancées

## – Méthodes in-place :

- ▶ `__iadd__(self, other)` : `+=` (addition)
- ▶ `__isub__(self, other)` : `-=` (soustraction)
- ▶ `__imul__(self, other)` : `*=` (multiplication)
- ▶ `__itruediv__(self, other)` : `/=` (division)
- ▶ `__ifloordiv__(self, other)` : `//=` (division entière)
- ▶ `__imod__(self, other)` : `%=` (modulo)
- ▶ `__ipow__(self, other)` : `**=` (puissance)
- ▶ `__imatmul__(self, other)` : `@=` (multiplication matricielle)
- ▶ `__iand__(self, other)` : `&=` (Et logique)
- ▶ `__ior__(self, other)` : `|=` (Ou logique)



# Les classes avancées

## — Méthodes d'affichage :

- ▶ `__str__(self)` : `str(objet)`, `print(objet)` (affichage)
- ▶ `__repr__(self)` : `repr(objet)` (autre affichage)
- ▶ `__format__(self, format_spec)` : `format(objet, format_spec)` (format)
- ▶ `__bytes__(self)` : `bytes(objet)` (bytes code)

# Les classes avancées

## — Fonctions sur les attributs d'une classe :

- ▶ `__getattr__(self, attr)` : `objet.attr`
- ▶ `__getattribute__(self, attr)` : `objet.attr`
- ▶ `__setattr__(self, attr, val)` : `objet.attr = val`
- ▶ `__delattr__(self, attr)` : `del objet.attr`
- ▶ `__dir__(self)` : `dir(objet)`

# Les classes avancées

## — Fonctions sur les attributs d'une classe :

- ▶ `__getattr__(self, attr)` : `objet.attr`
- ▶ `__getattribute__(self, attr)` : `objet.attr`
- ▶ `__setattr__(self, attr, val)` : `objet.attr = val`
- ▶ `__delattr__(self, attr)` : `del objet.attr`
- ▶ `__dir__(self)` : `dir(objet)`

## Exercice « ex\_surcharge.py »

- On souhaite modéliser une classe Rectangle représentant un rectangle dans un plan cartésien. Un rectangle est défini par sa largeur (largeur) et sa hauteur (hauteur).
- Vous devez :
  - ▶ Implémenter la classe Rectangle avec un constructeur prenant en paramètre largeur et hauteur.
  - ▶ Surcharger les operateurs suivants :
    - `+` : pour additionner deux rectangles (somme des largeurs et hauteurs respectives).
    - `*` : pour multiplier un rectangle par un entier (multiplier la largeur et la hauteur).
    - `==` : pour comparer si deux rectangles ont la même surface.
    - `str` : pour afficher un rectangle sous la forme "Rectangle(largeur, hauteur)".
- Tester la classe avec différents cas.

# Les classes avancées

- Les générateurs sont des itérables (comme les collections) mais aussi des itérateurs, ce qui implique qu'ils se consomment quand on les parcourt et que nous pouvons donc les parcourir qu'une fois (comme map ou filter). Ils sont généralement créés par des fonctions construites à l'aide du mot clef « yield ». Par abus de langage ces fonctions génératrices sont elles-mêmes appelées générateurs.

# Les classes avancées

- Pour créer une fonction génératrice, elle doit contenir un ou plusieurs mot clé `yield`. Lors de l'appel, la fonction retournera un générateur, et appel à la fonction builtin « `next()` » sur la fonction génératrice, le code jusqu'au prochain `yield` sera exécuté. Puis la fonction reprendra à partir du `yield` lorsqu'un nouvel appel à ce générateur sera effectué. Comme pour tout itérateur, la fonction `next` appelle la méthode spéciale `__next__` du générateur.
- `Yield` peut-être ou non suivi d'une expression. La valeur retournée par `next` sera celle du `yield`, ou `None` si aucune valeur ou expression n'est spécifiée après `yield`.

# Les classes avancées

- Après la création d'un générateur, il est possible de le modifier via la méthode `send(paramètre)`. L'utilisation de `send` provoque la reprise de l'exécution de la fonction génératrice à partir du `yield` où elle s'était arrêtée. Ce `yield` retourne à la fonction, le paramètre transmis via le `send`.
- Il est possible d'utiliser un sous-générateur (un générateur dans un autre) via le mot clé `yield from` « `sous_generateur()` ».
- Il est également possible créer des générateurs en compréhension, comme les listes, en remplaçant les `[]` par des `()`.

# Les classes avancées

- La méthode `throw()` permet de lever une exception.
- La méthode `close()` d'un générateur permet de stopper celui ci.
- Remarque : liste ou générateur ?
  - ▶ Les listes prennent plus de place en mémoire car tous les éléments existent en même temps alors que le générateur n'a que le dernier élément généré en mémoire.
  - ▶ Les générateurs ont une durée de vie plus courte car une fois itérés, ils disparaissent. Par conséquent on peut créer des générateurs avec des boucles infinies.
  - ▶ Les générateurs ne sont pas indexables.



# Exercice « ex\_generateurs.py »

## — Énoncé :

- ▶ Écrire un générateur en Python qui génère tous les nombres pairs jusqu'à un nombre donné  $n$ .
- ▶ Le générateur doit s'arrêter lorsque le nombre généré est supérieur ou égal à  $n$ .

ib  
cegos

# Patrons de conception





# Patrons de conception

## – Qu'est-ce qu'un patron de conception

- ▶ En Python, comme dans d'autres langages orientés objet, les design patterns (ou patrons de conception) sont des solutions réutilisables à des problèmes courants de conception logicielle. Python est très flexible et certains patrons sont déjà implémentés naturellement grâce à ses fonctionnalités (décorateurs, introspection, etc.). On peut souvent remplacer des patterns classiques Java/C++ par des structures Python plus élégantes.

# Patrons de conception

## – Singleton

- ▶ Assure qu'une classe n'ait qu'une seule instance dans tout le programme.
- ▶ Exemple d'intérêt :
  - Gérer une seule connexion à une base de données.

# Patrons de conception

## — Factory

- ▶ Délègue la création d'objets à une fonction ou une classe.
- ▶ Exemple d'Intérêt :
  - Simplifie la création d'objets complexes.
  - Permet d'instancier dynamiquement des classes selon les besoins.
  - Favorise l'extensibilité (on peut ajouter de nouveaux types sans changer le code client).

# Patrons de conception

## — Observer

- ▶ Permet à des objets d'être notifiés automatiquement quand un autre change d'état.
- ▶ Exemple d'Intérêt :
  - Découpler l'émetteur (sujet) des récepteurs (observateurs) dans un système de messagerie où chaque utilisateur est notifié quand il reçoit un message.

# Patrons de conception

## – Strategy

- ▶ Permet à un objet de changer dynamiquement son comportement sans changer son code. Cela fonctionne en définissant une interface de stratégie que des classes concrètes peuvent implémenter, puis en permettant à un "contexte" (l'objet qui utilise ces stratégies) de changer la stratégie pendant l'exécution.
- ▶ Exemple d'intérêt :
  - Permet de changer d'algorithme à la volée sans toucher au reste du code.

## Exercice « ex\_patron.py »

- Un centre météorologique souhaite développer un système de notifications.
- Lorsqu'une mise à jour de la température est enregistrée, plusieurs dispositifs doivent être informés automatiquement :
  - ▶ Une application mobile
  - ▶ Un panneau d'affichage
  - ▶ Un site web



# Exercice « ex\_patron.py »

- Implémente le design pattern Observer pour :
  - ▶ Créer une classe StationMeteo (le sujet/observable) qui contient la température actuelle.
  - ▶ Permettre aux dispositifs (observateurs) d'être ajouté ou retiré.
  - ▶ Informer automatiquement tous les observateurs ajoutés à chaque mise à jour de température.
- Contraintes :
  - ▶ Chaque observateur doit implémenter une méthode mise\_a\_jour(temp) qui affiche un message personnalisé.
  - ▶ Tu dois utiliser des classes Python pour structurer le tout.

ib  
cegos

## TP validation des acquis 2



# TP validation des acquis 2

## — Énoncé :

- ▶ Créer un programme en Python pour enregistrer des commandes clients, en utilisant :
  - Une classe simple,
  - La gestion des erreurs,
  - Un fichier texte pour sauvegarder.

## — Consignes :

- ▶ Créer une classe Commande avec 3 attributs :
  - client (nom du client),
  - produit (nom du produit),
  - quantite (nombre entier strictement positif).
- ▶ Si la quantité est invalide (pas un entier positif), le programme doit lever une exception.

# TP validation des acquis 2

## — Enoncé :

- ▶ Ajouter une méthode `afficher()` qui retourne la commande sous forme de texte :
  - Exemple : "Client: Alice - Produit: Livre - Quantite: 2"
- ▶ Ajouter une méthode `sauvegarder()` qui ajoute la commande dans un fichier texte `commandes.txt`.
- ▶ Créer une fonction `charger_commandes()` qui affiche toutes les commandes enregistrées dans le fichier.
- ▶ Dans le programme principal :
  - Créer deux commandes valides,
  - Tenter de créer une commande avec une quantité invalide,
  - Sauvegarder les commandes valides,
  - Afficher le contenu du fichier.

# TP validation des acquis 2

## – Résultat attendu dans le terminal :

Creation de la commande 1 (valide).

Commande 1 creee avec succes.

Creation de la commande 2 (valide).

Commande 2 creee avec succes.

Creation de la commande 3 (invalide).

Erreur lors de la creation de la commande: Quantite invalide. Elle doit etre un entier positif.

Commandes enregistrees :

Client: Alice - Produit: Livre - Quantite: 2

Client: Bob - Produit: Stylo - Quantite: 5



ib  
cegos

UML



# UML

- UML (Unified Modeling Language) est utilisé pour modéliser des projets informatiques et particulièrement pour la Programmation Orienté Objet (POO).
- Il permet de représenter graphiquement (sous forme de différent diagramme) la structure et le comportement d'un système, indépendamment du langage de programmation.
- Il existe 14 diagrammes possible en UML.



# UML

- Liste des diagrammes disponibles par catégorie :
  - ▶ Diagrammes structurels (montrent la structure statique du système)
    - Diagramme de classes
    - Diagramme d'objets
    - Diagramme de composants
    - Diagramme de déploiement
    - Diagramme de packages
    - Diagramme de structure composite
    - Diagramme de profils



# UML

## — Liste des diagrammes disponibles par catégorie :

### ► Diagrammes comportementaux (montrent la dynamique, les interactions)

- Interaction (communication entre éléments) :

- Diagramme de cas d'utilisation
- Diagramme de séquence
- Diagramme de communication
- Diagramme de minutage (timing)
- Diagramme d'interaction globale

- Activité et états internes :

- Diagramme d'activités
- Diagramme d'états (ou d'états-transitions)

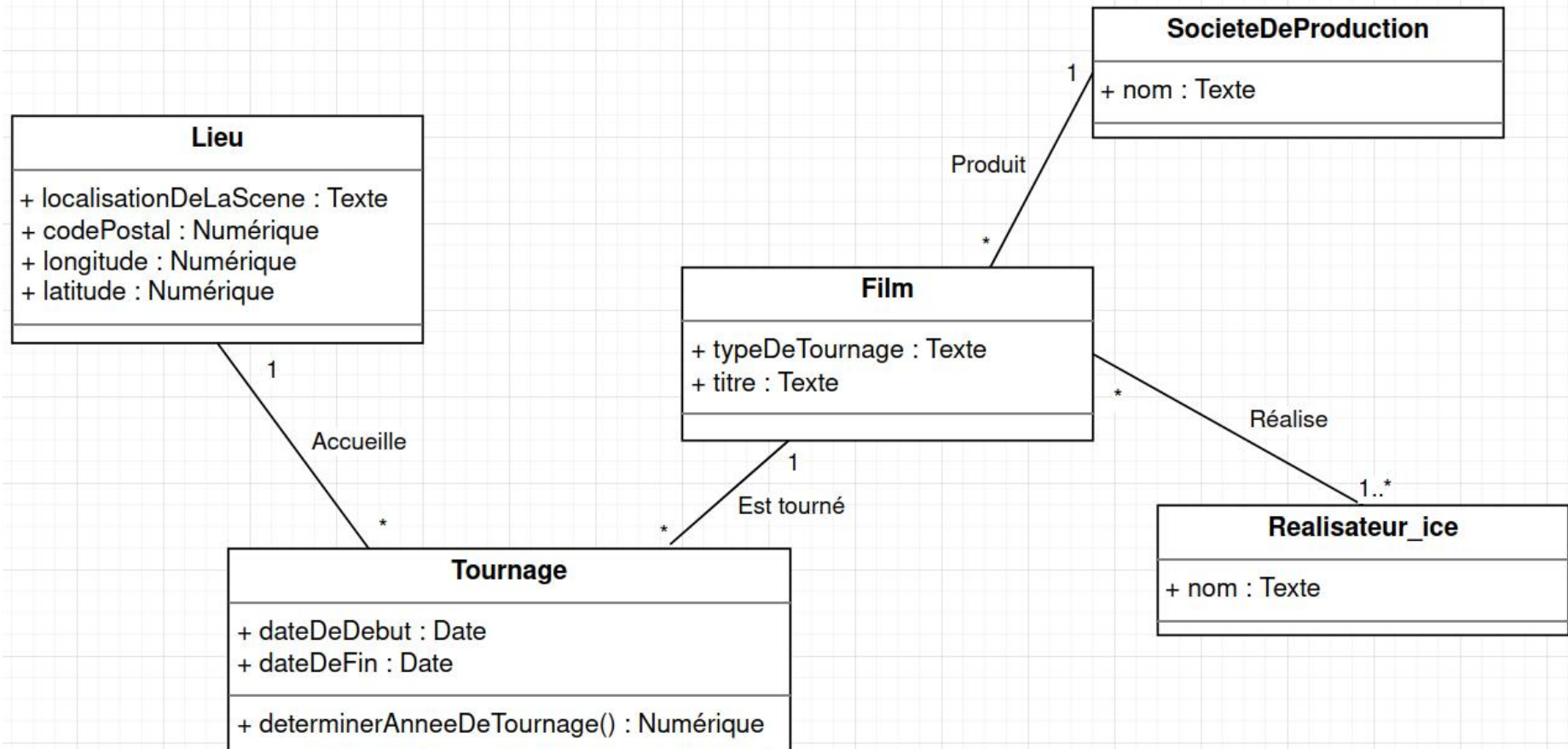
# UML

## – Diagramme de classes :

- ▶ Représente la structure statique du système : les classes, leurs attributs, leurs méthodes et les relations entre elles.
- ▶ Éléments clés :
  - Classe : boîte avec 3 parties (Nom / Attributs / Méthodes)
  - Relation d'association : lien entre classes (ex : un étudiant suit un cours)
  - Héritage : flèche avec un triangle vide
  - Agrégation/Composition : losange vide/plein
  - Multiplicité : 1, 0..1, 0..\*, etc.

# UML

## – Diagramme de classes :



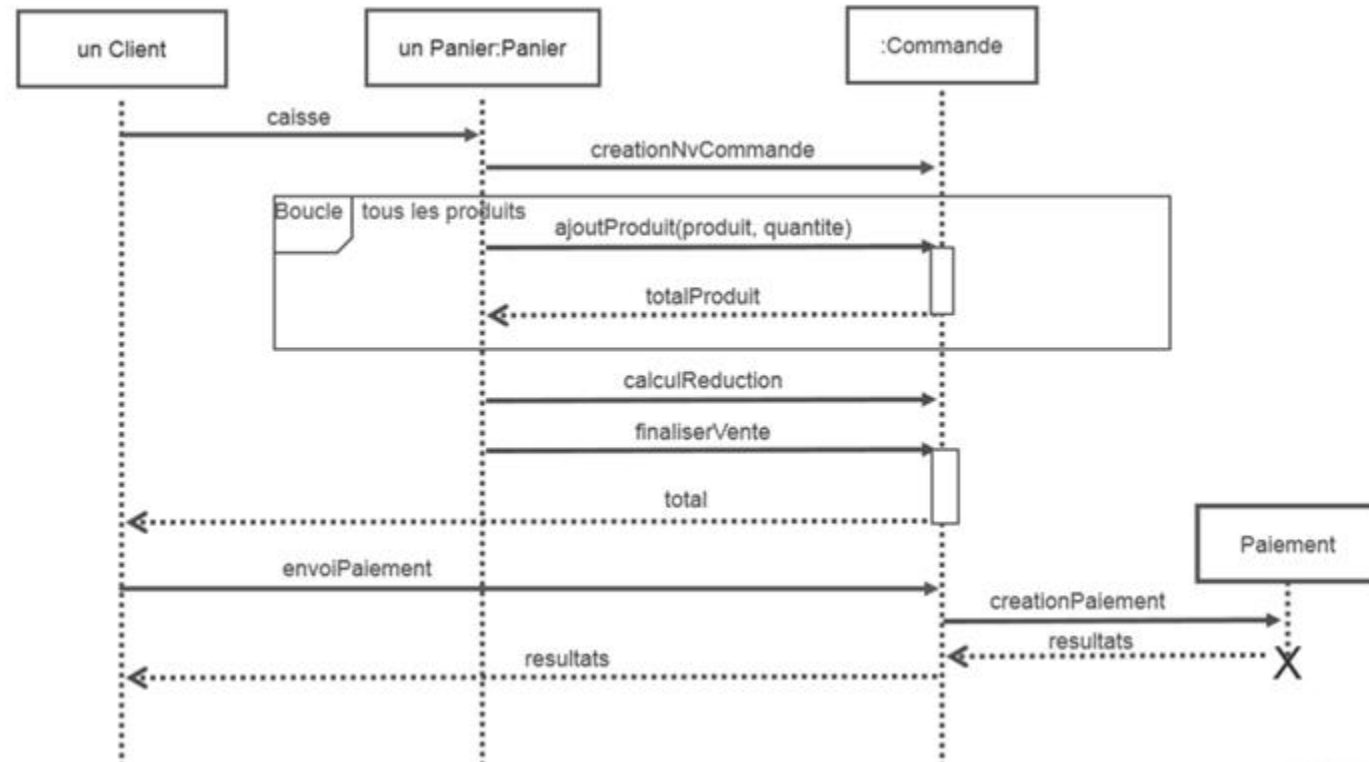
# UML

## – Diagramme de séquence :

- ▶ Modélise les interactions dynamiques entre objets dans le temps (appel de méthodes, messages échangés).
- ▶ Éléments clés :
  - Acteurs/Objets : représentés horizontalement
  - Temps : vertical, du haut vers le bas
  - Messages : flèches avec nom de méthode + paramètres
  - Activation : barre verticale pendant l'exécution

# UML

## – Diagramme de séquence :



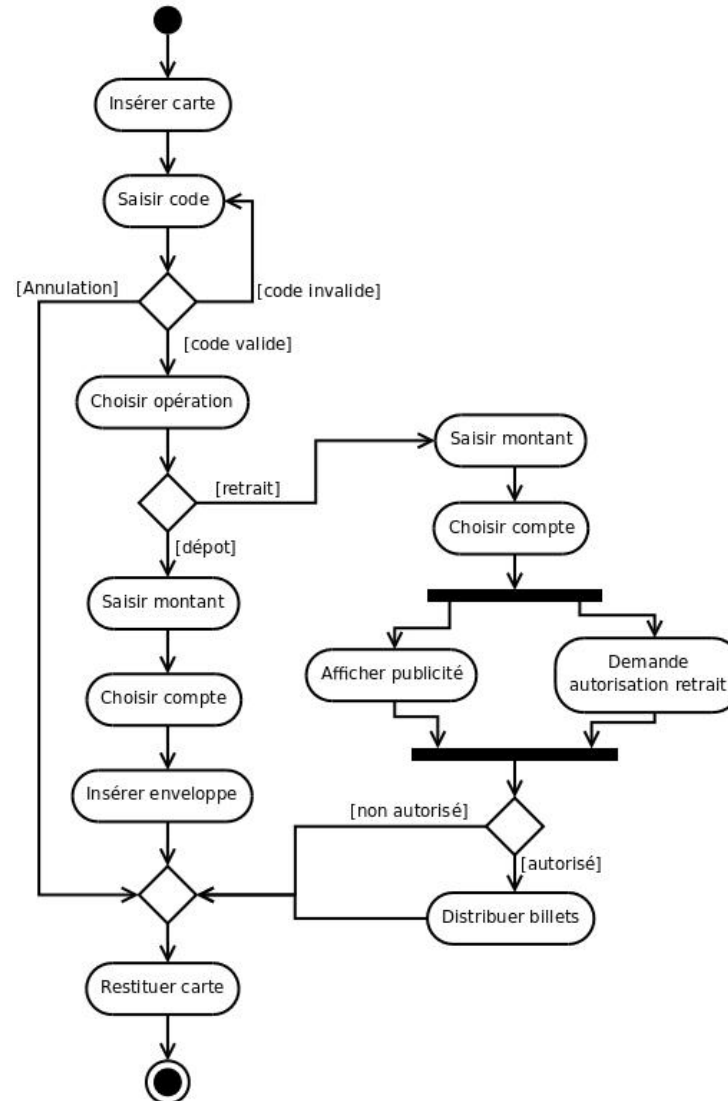
# UML

## – Diagramme d'activités :

- ▶ Représente le flux de contrôle ou de données dans un processus métier ou une opération.
- ▶ Éléments clés :
  - Activités : rectangles arrondis
  - Transitions : flèches
  - Décision/Condition : losanges
  - Début/Fin : ronds pleins et cibles
  - Barre Horizontale ou verticale : plusieurs activités en parallèles

# UML

## — Diagramme d'activité :





**ib  
cegos**

# Les modules utiles





# Les modules utiles

## — Modules

- Il existe un grand nombre de modules préprogrammés qui sont fournis d'office avec Python. Vous pouvez en trouver d'autres chez divers fournisseurs. Souvent on essaie de regrouper dans un même module des ensembles de fonctions apparentées, que l'on appelle des bibliothèques.

### Module Math

```
from math import *  
sqrt()  
sin()  
cos()  
pi  
...
```

### Module DateTime

```
from datetime import date  
from datetime import time  
from datetime import datetime
```

# Les modules utiles

## – Subprocess

- ▶ Le module subprocess en Python permet de lancer des processus externes, de communiquer avec eux et de récupérer leur sortie ou leur code de retour. Il est très utile pour exécuter des commandes système depuis un script Python.

## – Les fonctions principales :

- ▶ `run()` : Exécute une commande, attend qu'elle se termine, et retourne un objet `CompletedProcess`.
- ▶ `Popen()` : Exécute une commande, et continue le code python même si la commande n'est pas terminée.
- ▶ `check_output()` : Exécute une commande et retourne la sortie (stdout).

# Les modules utiles

## — Subprocess

### ► Options courantes :

- `stdin`, `stdout`, `stderr` : pour gérer les flux d'entrée/sortie.
- `shell=True` : exécute la commande dans un shell (attention aux failles de sécurité).
- `text=True` (ou `universal_newlines=True`) : pour avoir des chaînes de caractères au lieu de bytes.
- `timeout` : pour définir un délai d'expiration.

# Les modules utiles

## – Sys

► Ce module est utile pour interagir avec l'environnement d'exécution et manipuler les entrées/sorties ou les modules dynamiquement. Le module sys en Python fournit des fonctions et variables permettant d'interagir avec l'interpréteur Python. Voici ses principales fonctions et variables :

- Gestion des Arguments de la Ligne de Commande
  - `sys.argv` : Liste des arguments passés au script (le premier élément est le nom du script).
  - `sys.exit([code])` : Termine le programme avec un code de sortie optionnel.
- Flux d'Entrée et de Sortie
  - `sys.stdin` : Gère l'entrée standard.
  - `sys.stdout` : Gère la sortie standard.
  - `sys.stderr` : Gère les erreurs et les messages d'alerte.

# Les modules utiles

## — Sys

- Informations sur l'Interpréteur Python

- `sys.version` : Retourne la version de Python sous forme de chaîne.
- `sys.platform` : Indique le système d'exploitation.
- `sys.executable` : Chemin de l'exécutable Python utilisé.

- Gestion des Modules

- `sys.modules` : Dictionnaire des modules chargés.
- `sys.path` : Liste des chemins où Python recherche les modules.
- `sys.getrecursionlimit()` : Obtient la profondeur maximale de récursion.
- `sys.setrecursionlimit(n)` : Définit la profondeur maximale de récursion.

# Les modules utiles

## — Sys

- Gestion de la Mémoire

- `sys.getsizeof(objet)` : Retourne la taille d'un objet en mémoire.
- `sys.maxsize` : Taille maximale d'un entier en Python.
- `sys.getrefcount(objet)` : Retourne le nombre de références à un objet.

# Les modules utiles

## — OS

► Le module `os` en Python permet d'interagir avec le système d'exploitation. Ce module est souvent utilisé avec `shutil` pour des manipulations avancées de fichiers et répertoires. Voici ses principales fonctions :

- Gestion des processus
  - `system(command)` : Exécute une commande système.
  - `popen(command)` : Exécute une commande et retourne son résultat.
  - `getpid()` : Retourne l'ID du processus actuel.
  - `getppid()` : Retourne l'ID du processus parent.

# Les modules utiles

## — OS

- Informations système

- `name` : Donne le nom du système d'exploitation ('posix', 'nt', etc.).
- `uname()` : Retourne des informations sur le système (uniquement sous Unix).
- `environ` : Dictionnaire contenant les variables d'environnement.
- `getlogin()` : Retourne le nom de l'utilisateur connecté.



# Les modules utiles

## — OS

- Gestion des fichiers et répertoires
  - `getcwd()` : Retourne le répertoire de travail actuel.
  - `chdir(path)` : Change le répertoire de travail.
  - `listdir(path)` : Liste les fichiers et dossiers d'un répertoire.
  - `mkdir(path)` : Crée un dossier.
  - `makedirs(path)` : Crée un dossier et ses parents si nécessaires.
  - `remove(path)` : Supprime un fichier.
  - `rmdir(path)` : Supprime un dossier vide.
  - `removedirs(path)` : Supprime un dossier et ses parents s'ils sont vides.

# Les modules utiles

## — OS

- Manipulation des chemins

- `path.join(path1, path2)` : Construit un chemin valide.
- `path.exists(path)` : Vérifie si un chemin existe.
- `path.isfile(path)` : Vérifie si un chemin est un fichier.
- `path.isdir(path)` : Vérifie si un chemin est un répertoire.
- `path.abspath(path)` : Donne le chemin absolu d'un fichier.
- `path.basename(path)` : Retourne le nom du fichier d'un chemin.
- `path.dirname(path)` : Retourne le dossier d'un chemin.

# Les modules utiles

## — pathlib

► Le module `pathlib` de Python v3.4 ou supérieur fournit une interface orientée objet pour manipuler des chemins de fichiers et de répertoires. Ce module est particulièrement utile pour remplacer `os.path` avec une approche plus intuitive et plus puissante. Voici ses principales fonctions et classes :

- Création et manipulation de chemins

- `Path("chemin/vers/fichier")` : Crée un objet `Path` représentant un chemin (relatif ou absolu).
- `Path.cwd()` : Retourne le chemin du répertoire de travail actuel.
- `Path.home()` : Retourne le chemin du répertoire utilisateur.

- Navigation et vérifications

- `exists()` : Vérifie si le chemin existe.
- `is_file()` : Vérifie si le chemin est un fichier.
- `is_dir()` : Vérifie si le chemin est un répertoire.

# Les modules utiles

## — pathlib

- Opérations sur les chemins
  - `name` : Nom du fichier avec extension.
  - `stem` : Nom du fichier sans extension.
  - `suffix` : Extension du fichier.
  - `parent` : Répertoire parent du fichier/dossier.
  - `parts` : Renvoie les différentes parties du chemin sous forme de tuple.
- Lecture et écriture de fichiers
  - `read_text()` : Lit le contenu du fichier sous forme de texte.
  - `read_bytes()` : Lit le contenu du fichier en bytes.
  - `write_text("contenu")` : Écrit du texte dans le fichier.
  - `write_bytes(b"contenu")` : Écrit des bytes dans le fichier.

# Les modules utiles

## — pathlib

- Création et suppression
  - `mkdir(parents=True, exist_ok=True)` : Crée un répertoire (y compris les parents s'ils n'existent pas).
  - `touch()` : Crée un fichier vide.
  - `unlink()` : Supprime un fichier.
  - `rmdir()` : Supprime un répertoire vide.
- Gestion des fichiers et répertoires
  - `iterdir()` : Itère sur les éléments d'un répertoire.
  - `glob("*.txt")` : Recherche des fichiers correspondant à un motif.
  - `rename("nouveau_nom")` : Renomme le fichier ou le répertoire.
  - `replace("nouveau_chemin")` : Déplace le fichier en écrasant s'il existe déjà.

ib  
cegos

**Base de données**





# Base de données

- Une base de données est une collection organisée de données. Les informations ou les données doivent être organisées de manière à pouvoir être facilement accessibles, gérées et mises à jour. Il existe les bases des données relationnelles et non-relationnelles.

# Base de données

- Une base de données relationnelle consiste en une collection de données avec des relations prédéfinies entre son contenu. Les données d'une base de données relationnelle ont une relation avec au moins une autre donnée de la base. Souvent, la collection de données est organisée en tableaux avec des colonnes et des lignes. Chaque colonne d'un tableau contient un certain type de données et chaque ligne comporte un ensemble de valeurs liées à une entité.



# Base de données

- Pour créer et mettre à jour une base de données relationnelle, nous devons utiliser un Système de Gestion de Base de Données Relationnelle, un SGBDR. Il s'agit d'un type particulier de programme qui nous permet de travailler avec des bases de données relationnelles (SQLite, MySQL, SQL Server ou PostgreSQL). Ils utilisent le langage standard ISO d'SQL (Structured Query Language) pour accéder à la base de données. Mais les fonctions supplémentaires ne sont pas standard et varient d'un SGBDR à l'autre.

# Base de données

- Une base de données non relationnelle n'utilise pas de tables et de rangées. Le modèle de stockage est optimisé pour le type de données stockées. Et le format de stockage peut varier en fonction des données. Elle est pour traiter des données non structurées et diverses qui ne correspondent pas exactement aux lignes et aux colonnes d'une base de données relationnelle : MongoDB, Cassandra.

# Base de données

- Michael Widenius : finlandais créateur de 3 BDD.
- MaxDB : c'est la première BDD qu'il a créée. Max est pour le prénom de son fils.
- MySQL : c'est sa deuxième BDD pour corrigé les erreurs faites dans MaxDB. My est le prénom de sa première fille. C'est un prénom finlandais courant.

# Base de données

- En 2009, Oracle rachète Sun Microsystems pour 7,4 milliards de dollars. Mais Michael Widenius et les dirigeants d'Oracle ne s'entendent pas du tout. Michael Widenius a quitté Oracle et puisque MySQL était open source, il a créé un fork de MySQL.
- Ce fork, c'est sa troisième BDD. Elle se nomme MariaDB du prénom de sa deuxième fille qui était juste née.

# Base de données

- Python DB API 2.0 (PEP 249) Encourage la conformité entre les modules Python utilisés pour accéder aux bases de données. Cela permet au code d'être plus facilement transférable d'une base de données à l'autre et d'élargir les systèmes de bases de données qui peuvent être utilisés avec Python. Presque tous les modules de base de données Python se conforment à cette interface. Ainsi, une fois que vous avez appris à utiliser les bases de données avec un module, il est facile de reprendre et de comprendre le code des autres modules.

# Base de données

- L'accès à la base de données doit être fourni par le biais d'un objet de connexion qui assure l'interface entre votre programme et la base de données.

`Connection = connect(param...)`

- Avec cet objet de connexion, plusieurs opérations peuvent être effectuées :

`Connection.commit()`

`Connection.rollback()`

`Connection.close()`

# Base de données

- Avec cet objet connexion, nous pouvons créer un curseur. Avec l'objet curseur, nous pouvons interagir avec la base de données :

`Connection = connect(param...)`

`Cursor = connection.cursor()`

`Cursor.execute(param)`

`Cursor.executeMany(param)`

`Cursor.fetchOne(param)`

`Cursor.fetchmany([numOfRows])`

`Cursor.fetchall()`

# Base de données

## – SQLite

- ▶ La bibliothèque standard de Python inclut un moteur de base de données relationnelles SQLite, qui a été développé en C, et implémente le standard SQL.
- ▶ Cela signifie donc que vous pouvez écrire en Python une application contenant son propre SGBDR intégré, sans qu'il soit nécessaire d'installer quoi que ce soit d'autre.
- ▶ Les instructions d'interaction à la BDDR sont standardisées (cf : PEP 249 sur [python.org](https://python.org)).



# Base de données

## – SQLite

```
>>> import sqlite3
>>> fichierDonnees = "E:/python3/essais/bd_test.sq3"
>>> conn = sqlite3.connect(fichierDonnees)
>>> cur = conn.cursor()
>>> cur.execute("CREATE TABLE membres (age INTEGER, nom TEXT, taille REAL)")
>>> cur.execute("INSERT INTO membres(age,nom,taille) VALUES(21,'Dupont',1.83)")
>>> cur.execute("INSERT INTO membres(age,nom,taille) VALUES(15,'Blumâr',1.57)")
>>> cur.execute("INSERT INTO membres(age,nom,taille) VALUES(18,'Özémir',1.69)")
>>> conn.commit()
>>> cur.close()
>>> conn.close()
```

# Base de données

## — Postgres

- ▶ Pour Postgres il faut installer le module via `pip install psycopg2`
- ▶ Puis importer le module `psycopg2`.
- ▶ Ensuite, le code ressemble beaucoup à celui de SQLite.

# Base de données

## — Postgres

```
>>> import psycopg2
>>> psycopg2.connect(dbname = 'template1', user = 'dbuser', password = 'dbpass', host = 'localhost', port =
'5432')
>>> cur = conn.cursor()
>>> cur.execute("CREATE TABLE membres (age INTEGER, nom TEXT, taille REAL)")
>>> cur.execute("INSERT INTO membres(age,nom,taille) VALUES(21,'Dupont',1.83)")
>>> cur.execute("INSERT INTO membres(age,nom,taille) VALUES(15,'Blumâr',1.57)")
>>> cur.execute("INSERT INTO membres(age,nom,taille) VALUES(18,'Özémir',1.69)")
>>> conn.commit()
>>> cur.close()
>>> conn.close()
```

# Base de données

## – Modules pour les BDD Relationnelles

BDD	Module	Remarque
SQLite	-	Pas besoin d'installation
MySQL / MariaDB	my-sql-connector	Officiel, stable, compatible PEP249
	PyMySQL	Pur Python, facile à utiliser
	MySQLdb	Ancien, à remplacer
PostgresSQL	psycopg2	Le plus populaire et performant
	asyncpg	Pour les applications asynchrones
Oracle	cx_Oracle	Officiel, maintenu chez Oracle
SQL Server	pyodbc	Supporte aussi d'autre bases via ODBC
	pymssql	Simple à utiliser

# Base de données

## — Modules pour les BDD NoSQL

BDD	Module	Remarque
MongoDB	pymongo	Officiel et largement utilisé
Redis	redis	Pour cache, sessions et file d'attente
Cassandra	cassandra-driver	Officiel par DataStax
CouchDB	CouchDB	Interface REST, peut aussi utiliser requests
Elasticsearch	elasticsearch	Pour la recherche full-text
Neo4 (graph)	neo4j	Accès via Cypher

ib  
cegos

# Gestionnaire de paquet PIP



# Gestionnaire de paquet PIP

## – Dans l'invite de commande de Windows :

- ▶ `pip install monPackage` # installation d'un paquet dans l'environnement actuel
- ▶ `pip freeze > requirements.txt` # liste les packages de l'environnement actuel dans le fichier requirements.txt
- ▶ `pip install -r requirements.txt` # installation des paquets listés dans le fichier requirements.txt

Remarque : les dépendances ne sont pas automatiquement mises à jour à ce moment là.

pip est le successeur du gestionnaire de paquets easy\_install.

Ces gestionnaires de paquets se base sur le dépôt de paquets PyPi (Python Package Index).

# Gestionnaire de paquet PIP

## — Créer un exécutable avec PyInstaller

► Dans l'invite de commande de Windows :

- `pip install PyInstaller` # installation de PyInstaller
- `PyInstaller -F chemin_complet\nom_fichier.py` # -F pour faire un exécutable indépendant

► Le fichier exécutable sera dans un dossier « dist » créer à l'emplacement du script.

- `--distpath chemin` permet de changer le dossier où se situe le fichier exécutable.
- `--noconsole` permet de ne pas lancer la console quand on exécute le programme (utile pour les interfaces graphiques)
- `--onefile` permet de créer un exécutable intégrant tout, même python.

► La différence entre -F et --onefile c'est que --onefile, décompresse pour installer les dépendances à l'exécution du fichier. -F exécute sans faire d'installation.





**ib  
cegos**

# Environnements virtuels



# Environnements virtuels

Installer Python c'est simple. Mais installer un environnement homogène et performant devient laborieux (un environnement scientifique par exemple) :

- ▶ Il faut commencer par isoler son environnement de travail du système.
- ▶ Il existe une multitude de librairies et gérer leurs dépendances peut s'avérer difficile.
- ▶ Il faut les compiler spécifiquement pour votre système si vous souhaitez exploiter toute la puissance de calcul des processeurs modernes.

# Environnements virtuels

Les environnements virtuels Python permettent d'avoir des installations de Python isolées du système et séparées les unes des autres. Cela permet de gérer plusieurs projets sur sa machine de développements, certains utilisant des modules de versions différentes, voir même des versions différentes de Python. Nous utiliserons le module `virtualenv` pour la suite du cours.

- Installation du package `virtualenv`

- ▶ `pip install virtualenv` # n'est plus nécessaire car `virtualenv` est fourni avec python

- Création d'un environnement virtuel

- ▶ `Python -m venv monvenv`

# Environnements virtuels

## — Activation d'un environnement virtuel

Sous windows :

```
.\monvenv\Scripts\activate
```

Sous linux :

```
. venv/bin/activate
```

Le « . » sous linux demande à exécuter la commande dans le shell courant au lieu d'un autre shell pour ne pas perdre les variables d'environnements.

### Remarque

Lorsque l'environnement virtuel est actif, nous avons le nom de l'environnement virtuel entre parenthèses en début de ligne de console.

# Environnements virtuels

- Désactivation d'un environnement virtuel

Sous windows :

```
deactivate
```

Sous linux :

```
. venv/bin/deactivate
```

## Remarque

- PyCharm créer directement un environnement virtuel à la création d'un nouveau projet.
- Menu, file, settings, python interpreter : la fenêtre indique les librairies installées et leurs versions. Cela indique aussi si une nouvelle version est disponible.



ib  
cegos

Tkinter



# Tkinter

## – Interface graphique avec Python

- Le domaine des interfaces graphiques (ou GUI, Graphical User Interfaces) est extrêmement complexe. Chaque système d'exploitation peut en effet proposer plusieurs « bibliothèques » de fonctions graphiques de base, auxquelles viennent fréquemment s'ajouter de nombreux compléments, plus ou moins spécifiques de langages de programmation particuliers. Tous ces composants sont généralement présentés comme des classes d'objets, dont il vous faudra étudier les attributs et les méthodes. Avec Python, la bibliothèque graphique la plus utilisée jusqu'à présent est la bibliothèque Tkinter, qui est une adaptation de la bibliothèque Tk développée à l'origine pour le langage Tcl. Tkinter est un environnement graphique de type événementiel fourni par défaut avec python. Cela permet à l'utilisateur de saisir des informations de façon non séquentielle.

# Tkinter

## – Interface graphique avec Python

- ▶ Plusieurs autres bibliothèques graphiques fort intéressantes ont été proposées pour Python : wxPython, pyQT, pyGTK, etc. Il existe également des possibilités d'utiliser les bibliothèques de widgets Java et les MFC de Windows.
- ▶ Il est aisé de construire différents modules Python, qui contiendront des scripts, des définitions de fonctions, des classes d'objets, etc... On peut alors importer tout ou partie de ces modules dans n'importe quel programme, ou même dans l'interpréteur fonctionnant en mode interactif. Pour utiliser Tkinter, il suffit de l'importer dans votre fichier python.



# Tkinter

## — Les 15 classes principales (widget) du module Tkinter

- ▶ **Button** : Un bouton classique, à utiliser pour provoquer l'exécution d'une commande quelconque.
- ▶ **Canvas** : Un espace pour disposer divers éléments graphiques. Ce widget peut être utilisé pour dessiner, créer des éditeurs graphiques, et aussi pour implémenter des widgets personnalisés.
- ▶ **Checkbutton** : Une « case à cocher » qui peut prendre deux états distincts (la case est cochée ou non). Un clic sur ce widget provoque le changement d'état.
- ▶ **Entry** : Un champ d'entrée, dans lequel l'utilisateur du programme pourra insérer un texte quelconque à partir du clavier.
- ▶ **Frame** : Un cadre rectangulaire dans la fenêtre, où l'on peut disposer d'autres widgets. Cette surface peut être colorée. Elle peut aussi être décorée d'une bordure.

# Tkinter

## — Les 15 classes principales (widget) du module Tkinter

- ▶ Label : Un texte (ou libellé) quelconque (éventuellement une image).
- ▶ Listbox : Une liste de choix proposés à l'utilisateur, généralement présentés dans une sorte de boîte. On peut également configurer la Listbox de telle manière qu'elle se comporte comme une série de « boutons radio » ou de cases à cocher.
- ▶ Menu : Un menu. Ce peut être un menu déroulant attaché à la barre de titre, ou bien un menu « pop up » apparaissant n'importe où à la suite d'un clic.
- ▶ Menubutton : Un bouton-menu, à utiliser pour implémenter des menus déroulants.
- ▶ Message : Permet d'afficher un texte. Ce widget est une variante du widget Label, qui permet d'adapter automatiquement le texte affiché à une certaine taille ou à un certain rapport largeur/hauteur.

# Tkinter

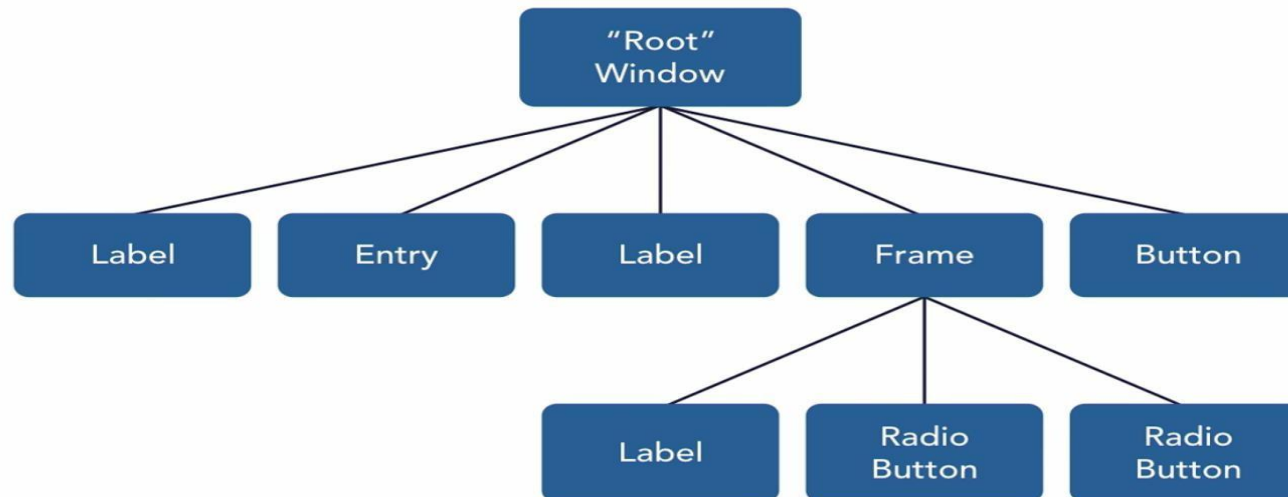
## — Les 15 classes principales (widget) du module Tkinter

- ▶ Radiobutton : Représente (par un point noir dans un petit cercle) une des valeurs d'une variable qui peut en posséder plusieurs. Cliquer sur un « bouton radio » donne la valeur correspondante à la variable, et « vide » tous les autres boutons radio associés à la même variable.
- ▶ Scale : Vous permet de faire varier de manière très visuelle la valeur d'une variable, en déplaçant un curseur le long d'une règle.
- ▶ Scrollbar : « ascenseur » ou « barre de défilement » que vous pouvez utiliser en association avec les autres widgets : Canvas, Entry, Listbox, Text.
- ▶ Text : Affichage de texte formaté. Permet aussi à l'utilisateur d'éditer le texte affiché. Des images peuvent également être insérées. Le texte peut être multiligne.
- ▶ Toplevel : Une fenêtre affichée séparément, « par-dessus ».

# Tkinter

## – Chaque type de widget est une classe

- Les widgets sont les conteneurs visuels utilisés pour organiser ces contrôles, comme la fenêtre de l'application et les cadres qu'elle contient. Les widgets peuvent également être utilisés pour afficher des informations à l'utilisateur, allant d'une simple étiquette de texte à un graphique complexe sur un canevas.



# Tkinter

## – Geometry Manager

- ▶ Le simple fait de créer un widget Tkinter ne le rend pas visible. Pour qu'un widget soit affiché à l'écran, Tkinter doit savoir exactement où dessiner ce widget. C'est là que la gestion de la géométrie entre en jeu.
- ▶ Le gestionnaire de géométrie de Tk prend les instructions du programme et fait de son mieux pour placer les widgets à l'endroit prévu. Pour ce faire, il utilise le concept de widgets maître et esclave.
- ▶ Lorsque vous créez un nouvel objet widget et que vous spécifiez son parent, vous identifiez ce widget parent comme le maître du widget enfant. Ce widget maître utilisera un gestionnaire de géométrie pour contrôler le placement de son widget esclave.
- ▶ Les widgets maîtres sont généralement des conteneurs organisationnels tels que des fenêtres ou des cadres de niveau supérieur. Déterminer l'endroit exact où placer le widget esclave n'est pas toujours un problème simple.

# Tkinter

## – Geometry Manager

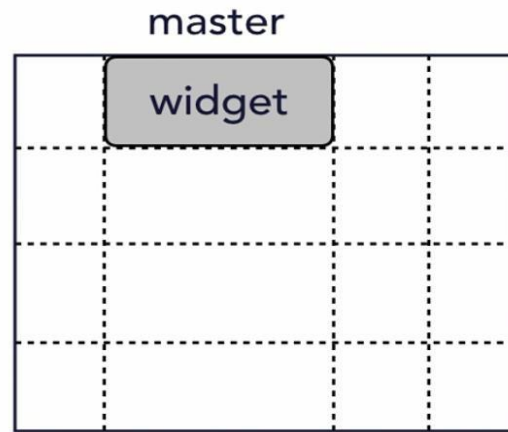
Pack

`Widget.pack(side = RIGHT)`



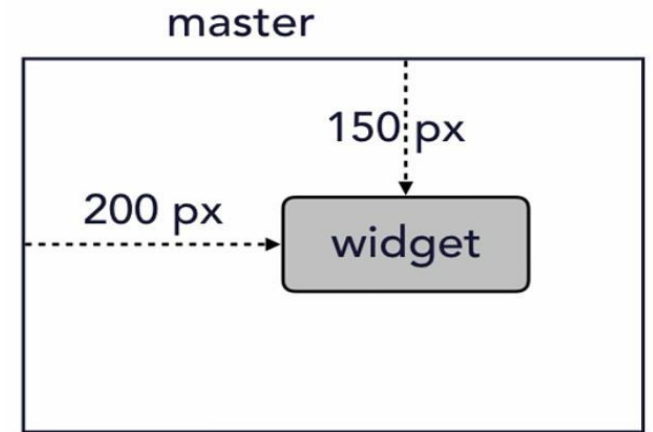
Grid

`Widget.grid(row = 0, column = 1)`



Place

`Widget.place(x = 200, y = 150)`



- Il est possible d'utiliser plusieurs gestionnaires de géométrie au sein d'une même application mais vous devez toujours utiliser le même gestionnaire de géométrie pour organiser les widgets dans le même master.

# Tkinter

## – Liaison d'événements

- ▶ La liaison d'événement est l'affectation d'une fonction à un événement d'un widget. Lorsque l'événement se produit, la fonction affectée est appelée automatiquement. Cela se fait via la méthode `bind()`. Le nom des événements gérés sont toujours des strings commençant par `<` et finissant par `>`. La fonction liée au widget devra obligatoirement avoir un paramètre qui correspondra à l'événement déclenché.

# Tkinter

## — Liaison d'événements

### ► Liste de quelques événements gérables :

- `<Button-1>` : clic avec le bouton de gauche.
- `<Button-2>` : clic avec le bouton du milieu.
- `<Button-3>` : clic avec le bouton de droite.
- `<Double-Button-1>` : double clic avec le bouton gauche
- `<Button>` : un bouton de la souris est enfoncé.
- `<ButtonRelease>` : un bouton de souris est relâché.



# Tkinter

## — Liaison d'événements

### ► Liste de quelques événements gérables :

- `<Alt>` : la touche Alt est enfoncée.
- `<Control>` : la touche Ctrl est enfoncée.
- `<Shift>` : la touche Shift est enfoncée.
- `<KeyPress>` : n'importe quelle touche est enfoncée.
- `<KeyRelease>` : une touche est relâchée.
- `<F1>` : la touche F1 est enfoncée.
- `<Control-C>` : la combinaison de touche Ctrl+C est enfoncée.

## Exercice « ex\_dessin\_lignes.py »

- Créer une fenêtre comportant trois boutons et un canevas. Suivant la terminologie de tkinter, un canevas est une surface rectangulaire délimitée, dans laquelle on peut installer ensuite divers dessins et images à l'aide de méthodes spécifiques. Lorsque l'on clique sur le bouton <Tracer une ligne>, une nouvelle ligne colorée apparaît sur le canevas, avec à chaque fois une inclinaison différente de la précédente.
- Si l'on actionne le bouton <Autre couleur>, une nouvelle couleur est tirée au hasard dans une série limitée et sera affichée dans un Entry. Cette couleur est celle qui s'appliquera aux tracés suivants.
- On affichera les coordonnées de début et fin de la ligne dans un Entry.
- La couleur de la ligne sera aléatoire au commencement du programme.
- Le bouton <Quitter> va terminer l'application en refermant la fenêtre.

## Exercice « ex\_resistance.py »

- Notre application doit faire apparaître une fenêtre comportant un dessin de la résistance, ainsi qu'un champ d'entrée dans lequel l'utilisateur peut encoder une valeur numérique.
- Un bouton <Montrer> déclenche la modification du dessin de la résistance, de telle façon que les trois bandes de couleur se mettent en accord avec la valeur numérique introduite.
- Contrainte : Le programme doit accepter toute entrée numérique dans les limites de 1 à 999 GΩ.

## Exercice « ex\_resistance.py »

- Pour rappel, la fonction des résistances électriques consiste à s'opposer (à résister) plus ou moins bien au passage du courant. Les résistances se présentent concrètement sous la forme de petites pièces tubulaires cerclées de bandes de couleur (en général 3 ou 4).
- Dans cet exercice nous utiliserons 3 bandes pour la valeur et 1 bande pour le coefficient multiplicateur. Nous n'indiquerons pas la bande suivante qui est la tolérance.

## Exercice « ex\_resistance.py »

- Ces bandes de couleur indiquent la valeur numérique de la résistance, en fonction du code suivant :
- Noir = 0, Brun = 1, Rouge = 2, Orange = 3, Jaune = 4, Vert = 5, Bleu = 6, Violet = 7, Gris = 8, Blanc = 9.
- On oriente la résistance de manière telle que les bandes colorées soient placées à gauche.
- La valeur de la résistance – exprimée en ohms ( $\Omega$ ) – s'obtient en lisant ces bandes colorées également à partir de la gauche : les trois premières bandes indiquent les trois premiers chiffres de la valeur numérique ; il faut ensuite accoler à ces trois chiffres un coefficient multiplicateur fournie par la quatrième bande.

## Exercice « ex\_resistance.py »

- Le coef multiplicateur egale 1 pour noir, 10 pour marron, 100 pour rouge, jusqu'à 1 000 000 000 (1 GΩ) pour blanc.
- Pour les valeurs inferieures à 100 on rajoute des 0 pour être sur 3 chiffres et on appliquera un coef de 0,1 (gold) ou 0,01 (silver)
- Exemple 1 : si les bandes colorées sont jaune, violette, noir et jaune.
- La valeur de cette résistance est 4700000 Ω, ou 4700 kΩ, ou encore 4,7 MΩ.
- Exemple 2 : pour 12Ω les couleurs seront marron, rouge, noir et gold.



## Exercice « ex\_calculatrice.py »

- Faire une calculatrice simple avec les opérations de base (+, -, \*, /).



ib  
cegos

Tests





# Tests

## – Test unitaire

- ▶ Nous pouvons faire des tests unitaires avec le module unittest.
- ▶ Les tests sont regroupés dans des classes de test qui doivent obligatoirement héritées de la classe unittest.TestCase.
- ▶ Généralement, on groupe dans une classe les tests ayant la même classe ou le même module comme point d'entrée. Les tests sont représentés par des méthodes dont le nom commence par test.
- ▶ Pour exécuter une classe de test, il faut utiliser la fonction unittest.main().
- ▶ Il est possible d'exécuter des instructions avant et après chaque test pour allouer et désallouer des ressources nécessaires à l'exécution des tests. On redéfinit respectivement pour cela les méthodes setUp() et tearDown().

# Tests

## – Test unitaire

- Pour réaliser les assertions, une classe de test utilise des méthodes d'assertion tel que :

`assertEqual(a, b)` → `a == b`

`assertNotEqual(a, b)` → `a != b`

`assertTrue(x)` → `bool(x)` est vrai

`assertFalse(x)` → `bool(x)` est faux

`assertIs(a, b)` → `a is b`

`assertIsNot(a, b)` → `a is not b`

`assertIsNone(x)` → `x is None`

`assertIsNotNone(x)` → `x is not None`

`assertIn(a, b)` → `a in b`

`assertNotIn(a, b)` → `a not in b`

`assertIsInstance(a, b)` → `isinstance(a, b)`

`assertNotIsInstance(a, b)` → `not isinstance(a, b)`

`assertRaises(x)` → `except(x)` est vrai

- Toutes ces méthodes acceptent le paramètre optionnel `msg` pour passer un message d'erreur à afficher si l'assertion échoue.

# Tests

## – Couverture de code

- ▶ Une couverture de test, nous permet de connaître le pourcentage de notre code qui est testé. C'est une métrique très utile qui peut vous aider à évaluer la qualité de votre suite de tests. Cela nous donne une idée des parts d'ombre qui peuvent subsister dans notre projet !
- ▶ Une bonne couverture de test, supérieure à 80 %, est signe d'un projet bien testé et auquel il est plus facile d'ajouter de nouvelles fonctionnalités.
- ▶ Les outils de couverture de code utilisent un ou plusieurs critères pour déterminer quelle ligne de code a été vérifiée lors de l'exécution de votre suite de tests.

# Tests

## – Couverture de code

► Il existe 5 façons de mesurer la couverture de code :

- La couverture de ligne : C'est sans doute la méthode la plus utilisée dans les outils de couverture de code. Elle permet de compter le nombre de lignes qui ont été testées.
- La couverture des instructions : Cette méthode peut être confondue avec la couverture de ligne, mais c'est un peu différent. En effet, elle permet de différencier les instructions qu'il y a dans une ligne, et de vérifier si elles sont toutes testées.
- La couverture de branche : Cette méthode compte le nombre de structures de contrôle qui ont été exécutées. Par exemple, la condition if constitue une structure de contrôle, ainsi if et else comptent pour deux branches. Cette métrique permet de vérifier qu'on a pris en compte toutes les possibilités du code.
- La couverture des conditions : Contrairement à la couverture de branche, cette métrique prend en compte l'ensemble des sous-expressions booléennes.
- La couverture des fonctions : Ici, on va compter le nombre de fonctions ou méthodes qui ont été appelées dans le code. Cependant, cette métrique ne prend pas en compte le nombre de lignes dans la fonction.

# Tests

## — Couverture de code

- ▶ Pour effectuer ces tests nous allons installer la bibliothèque coverage.py en écrivant :
  - `pip install coverage`
- ▶ Puis le module pytest-cov en tapant :
  - `pip install pytest-cov`
- ▶ Syntaxe pour tester un projet situé dans un dossier :
  - `pytest --cov=dossier_du_projet`

# Tests

## — Couverture de code

- ▶ Pour générer un rapport html dans un dossier nommé htmlcov situé dans le dossier du projet testé :
  - `pytest --cov=dossier_du_projet --cov-report html`
- ▶ Pour ne pas tester certains fichier ou dossier nous devons créer un fichier `.coveragerc` et indiquer à l'intérieur de celui-ci ce qu'on ne veut pas tester :
  - `[run]`
  - `omit = chemin/*`

# Tests

## — Module PyTest

- ▶ pytest est un framework de test open source pour Python. Il permet :
  - d'écrire des tests simples avec peu de code.
  - de gérer des tests complexes avec des fixtures, hooks, plugins, etc.
  - de produire des rapports clairs et lisibles.
- ▶ Pourquoi l'utiliser ?
  - Syntaxe claire, sans classe obligatoire.
  - Découverte automatique des fichiers/tests.
  - Compatibilité avec unittest et nose.
  - Très extensible via des plugins (pytest-cov, pytest-django, etc.).

# Tests

## — Module PyTest

### ► Installation

- Dans une fenêtre invite de commande en mode administrateur :
  - `pip install pytest`

### ► Exécution des tests

- Allez dans le dossier du projet via l'invite de commande et exécuter la commande `pytest`. Cela testera tout les fichiers Python qui commence par « `test_` ». Les classes à tester doivent commencer par « `Test` » et les fonctions par « `test_` ».



# Tests

## – Module PyTest

### ► Couverture de code

- Pour faire un rapport de couverture de code avec pytest, on installe d'abord le module « pytest-cov » via la commande dans une fenêtre d'invite de commande en mode administrateur :
  - `pip install pytest-cov`
- Ensuite il faut se placer dans le dossier racine du projet et exécuter la commande suivante :
  - `pytest --cov=src .\tests`
  - `.\tests` = le dossier des fichiers de tests dans le projet (optionnel selon le nom du dossier).
- Pour obtenir un rapport html (un dossier htmlcov est créé) :
  - `pytest --cov=src --cov-report=html .\tests`

# Tests

## — Module PyTest

### ► Couverture de code

- Pour ouvrir le rapport html à partir de la console :
  - `start htmlcov/index.html`
- Pour obtenir un rapport dans la console et un fichier texte :
  - `pytest --cov=src --cov-report=term --cov-report=term-missing > coverage.txt`

### ► Fichiers optionnels du projet

- Le fichier « requirements.txt » sert à connaître les modules utilisés dans le projet.
- Le fichier « conftest.py » permet de mettre des données réutilisables pour tous les fichiers de tests.

# Tests

## — Module Pylint

► Pylint est un outil d'analyse statique de code source pour Python. Son principal objectif est d'identifier des problèmes potentiels, des erreurs de programmation, des non-conformités aux conventions de codage et des pratiques non recommandées dans le code Python. Pour utiliser Pylint nous devons installer le module en exécutant la ligne de commande suivante :

- `pip install pylint`

► Puis dans Visual Studio Code, il faut installer l'extension du même nom.

## — Remarque :

► Il existe d'autres modules permettant de faire les mêmes choses, tel que :

- Flake8, MyPy, autopep8, BlackFormatter ou yapf

# Tests

## – Module PyChecker

- ▶ PyChecker est un outil équivalent à Pylint, mais il n'est compatible qu'avec Python v2. Pour installer le package, il faut exécuter la ligne de commande suivante :
  - `pip install PyChecker`
- ▶ Une fois PyChecker installé, vous devez configurer VSCode pour l'utiliser. Pour ce faire, vous pouvez ajouter une configuration dans votre fichier settings.json. Pour accéder à ce fichier, ouvrez le menu "Fichier" dans VSCode, sélectionnez "Préférences", puis "Paramètres". Ajoutez une section pour la configuration de PyChecker dans settings.json :
  - `"python.linting.pycheckerEnabled": true,`
  - `"python.linting.pycheckerPath": "/chemin/vers/pychecker"`
- ▶ Assurez-vous de remplacer `"/chemin/vers/pychecker"` par le chemin réel vers l'exécutable PyChecker.

## Exercice « ex\_unittest.py »

- Importer la fonction add du module "a\_tester.py"
- Les tests suivants doivent être couverts :
- L'addition de deux entiers positifs
- L'addition d'un entier et de zéro
- L'addition de deux entiers négatifs
- L'addition d'un entier négatif et d'un entier positif

ib  
cegos

# Interfaçage Python / C



# Interfaçage Python / C

## – Chargement et appel de librairie C dans Python

- ▶ On ne peut pas exécuter un fichier C directement dans Python. Pour le pouvoir il faut compiler le fichier C en shared library via gcc.
- ▶ Ouvrir VSCode, et placez-vous dans le dossier où se trouve le fichier C à compiler.
- ▶ Taper la ligne de commande pour compiler le fichier C « exemple.c » en shared library :
  - pour Linux : `gcc -shared -o exemple.so -fPIC exemple.c`
  - pour Windows : `gcc -shared -o exemple.dll exemple.c`
- ▶ Sous Windows cela crée un fichier « exemple.dll », sous Linux un fichier « exemple.so ».
- ▶ Puis pour appeler les fonctions du fichier C compilé nous créerons un fichier Python qui appellera les fonctions du fichier C compliées dans le fichier dll via le module standard Ctypes de Python.

# Interfaçage Python / C

## — Réécriture d'une fonction Python en C avec l'API Python / C

- ▶ Si on veut écrire une fonction Python en C il existe une API officielle nommé « Python.h ».
- ▶ Il faut se placer dans le dossier ou le fichier C est situé.
- ▶ Puis on exécute cette commande :
  - `gcc -shared -I"xxxx/include" carre_module.c -L"xxxx/libs" -lpythonvvv -o carre_module.pyd`
  - xxxx = chemin complet de Python.
  - vvv = version de python sans « . ». Exemple : 3.13 devient 313
- ▶ Dans le fichier Python il suffira d'importer le fichier .pyd comme un module classique et d'utiliser les fonctions de la même façon qu'un programme Python.



# Interfaçage Python / C

## – Réécriture d'une fonction Python en C avec l'API Python / C

- ▶ L'import du fichier `.pyd` risque de ne pas être reconnu par l'extension pylance de VSCode (souligné en jaune indiquant un avertissement de module non reconnu), cela fonctionne tout de même. Si on veut éviter le message d'erreur et avoir l'auto complétion il faut créer un fichier « `.pyi` » du même nom que le fichier « `.pyd` » dans le même dossier. C'est un fichier interface de python. Dans ce fichier il faut écrire les déclarations des fonctions compilées dans le fichier « `.pyd` ».

# Interfaçage Python / C

## — Création de modules C pour Python avec Cython

- ▶ Cython permet d'écrire du code proche de Python mais compilé en C. Auparavant nous utilisions pyrex.
- ▶ Pour utiliser Cython nous devons l'installer via le gestionnaire de paquet avec la ligne de commande suivante :
  - `pip install cython`
- ▶ Si un warning concernant le chemin d'installation s'affiche, vous pouvez rajouter le chemin indiquer dans votre variable d'enregistrement PYTHONPATH.

# Interfaçage Python / C

## — Création de modules C pour Python avec Cython

- Pour compiler avec cython nous aurons besoin d'un fichier « setup.py » et d'installer setuptools via la ligne de commande suivante :

- `pip install setuptools`

- Contenu du fichier « setup.py » :

```
from setuptools import setup
from Cython.Build import cythonize
setup(ext_modules = cythonize("exemple_cython.pyx"))
```

# Interfaçage Python / C

## — Création de modules C pour Python avec Cython

► Compilation via le fichier « setup.py » précèdent :

- `python setup.py build_ext --inplace`

► Une fois compiler, un fichier « c » et un fichier « pyd » est généré. Le fichier « pyd » peut être importé dans un fichier python standard. Il faudra créer un fichier d'interface python « .pyi » comme vu dans les slides précédents pour éviter d'avoir des avertissements de l'extension pylance de Python et avoir l'auto complétion.

# Interfaçage Python / C

## — Utilisation du code Python depuis C

- ▶ On crée un fichier C qui embarque l'interpréteur Python et on le compile en exécutable par la ligne de commande suivante :
  - `gcc -I"xxxx/include" python_depuis_c.c -L"xxxx/libs" -lpythonvvv -o python_depuis_c`
  - `xxxx` = chemin complet de Python.
  - `vvv` = version de python sans « . ». Exemple : 3.13 devient 313
- ▶ Un fichier exécutable est créé.

# Interfaçage Python / C

## – Utilisation du profileur de code Python

- ▶ Quand on développe des liaisons Python / C, il est utile de profiler le code pour trouver les goulots d'étranglement.
- ▶ Pour cela nous utilisons le module cProfile. Ce module Python est un outil de profilage très utile pour analyser les performances de votre code. Il vous permet de mesurer le temps d'exécution de chaque fonction et d'obtenir des informations détaillées sur les parties de votre programme qui consomment le plus de ressources. Cela est particulièrement utile pour l'optimisation du code.

# Interfaçage Python / C

## – Utilisation du profileur de code Python

► Le rapport généré par cProfile contient plusieurs colonnes importantes :

- `ncalls` : Le nombre d'appels de la fonction.
- `tottime` : Le temps total passé dans la fonction (sans inclure le temps passé dans des fonctions appelées par celle-ci).
- `percall` : Temps moyen par appel.
- `cumtime` : Temps cumulé, c'est-à-dire le temps passé dans cette fonction, y compris les fonctions appelées.
- `filename:lineno(function)` : Emplacement de la fonction dans le fichier source.



**ib  
cegos**

# **Extraction automatique de documentation**





# Extraction automatique de documentation

L'extraction automatique de documentation en Python consiste à extraire des commentaires, des docstrings ou des métadonnées du code pour générer de la documentation lisible. Voici les approches principales qu'on peut utiliser selon le contexte :

- Module « ast » (Abstract Syntax Tree)
  - ▶ Permet d'analyser statiquement le code Python et d'extraire les docstrings.
- Module « inspect »
  - ▶ Fonctionne à l'exécution. Utile si on peut importer les modules.

# Extraction automatique de documentation

## — Module « sphinx »

- ▶ C'est un module qui n'est pas compris dans la bibliothèque standard de Python. Il faut l'importer via :
  - `pip install sphinx`
- ▶ Cette commande est à effectuer dans l'invite de commande en mode administrateur de Windows.
- ▶ Ce module permet de générer des documents HTML, PDF, etc... Il se base sur des docstrings formatées (reStructuredText, Google style, NumPy style).

# Extraction automatique de documentation

## — Module « sphinx »

### ► Exemple :

- Fichier « doc\_sphinx.py » avec une docstring structurée :

```
def addition(a, b):  
    """  
    Additionne deux nombres.  
    :param a: premier nombre  
    :param b: deuxième nombre  
    :return: somme de a et b  
    """  
    return a + b
```

# Extraction automatique de documentation

## — Module « sphinx »

► Dans l'invite de commande :

- `cd mon_projet`
- `sphinx-quickstart docs`

Répondre aux questions et cela crée la structure de la documentation dans un dossier nommé « docs ».

- Copier le fichier « `doc_sphinx.py` » dans le dossier « docs » nouvellement créé.
- Aller dans le fichier « `conf.py` » situé dans le sous dossier « docs\source »

# Extraction automatique de documentation

## — Module « sphinx »

- Remplacer la liste nommée extensions par la liste suivante :

```
extensions = [  
    'sphinx.ext.autodoc',           # Pour les directives .. automodule:: etc.  
    'sphinx.ext.napoleon',         # Pour les docstrings Google/NumPy (optionnel)  
    'sphinx.ext.viewcode'         # Pour les liens vers le code source (optionnel)  
]
```

# Extraction automatique de documentation

## — Module « sphinx »

► Dans l'invite de commande :

- `sphinx-apidoc -o docs/source docs`

► Cela génère les fichiers reStructuredText « rst » dans le sous dossier « docs\source » à partir du code python situé dans le dossier « docs ».

► Il est possible d'avoir un projet Python dans un autre dossier. Dans ce cas il faudra changer « docs » par celui du projet ou se situe les fichiers Python. Pour pouvoir générer les fichiers « rst » de tous les fichiers situés dans des sous dossiers du projet il faut créer un fichier « `__init__.py` » (vide s'il n'existe pas déjà) dans chaque sous dossier du projet pour que sphinx considère ces sous dossiers comme étant des packages de Python. Si on ne crée pas ce fichier, alors sphinx fera la documentation quand même, mais ces sous dossier seront considérés comme des modules isolés.

# Extraction automatique de documentation

## — Module « sphinx »

- ▶ Aller dans le fichier index.rst et rajouter en fin de fichier la ligne suivante :

```
modules
doc_sphinx
```

Attention à bien avoir 4 espaces avant pour qu'il soit au même niveau que toctree::

- ▶ Dans l'invite de commande :

- cd docs
- make html

- ▶ Cela génère les fichiers de documentation en html ouvrable à partir de « index.html » situé dans le sous dossier « docs\build\html ».

# Extraction automatique de documentation

## — Module « sphinx »

- ▶ Il est possible d'automatisé l'insertion des fichiers « rst » générés. Pour cela il faut écrire dans l'invite de commande :
  - `sphinx-apidoc -o docs/source docs --tocfile index`
- ▶ Si le fichier « index.rst » a déjà été généré par une commande précédente, alors il sera remplacé.



ib  
cegos

**Remerciements**





# Remerciements

Votre formateur Xavier TABUTEAU et ib Cegos vous remercie d'avoir participé à cette formation.

Seriez-vous intéressé par une autre formation ?

Si oui, laquelle ?



ib  
cegos

FIN

