

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
- ▶ Importation de modules
- ▶ Les variables
- ▶ Les structures répétitives
- ▶ Les structures conditionnelles
- ▶ Les fonctions
- ▶ TP de validation des acquis 1

Table des matières

— Chapitres

- ▶ Les exceptions
- ▶ Les fichiers
- ▶ Les classes
- ▶ TP de validations des acquis 2
- ▶ Les modules utiles
- ▶ Les modules scientifiques
- ▶ Gestionnaire de paquets
- ▶ Environnements virtuels



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/>

– Installer un éditeur et ses extensions

- ▶ Nous allons utiliser VSCode pour écrire nos codes. Utiliser ce lien pour installer l'application : <https://code.visualstudio.com/download>

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.

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

Installations

– Installation de Python

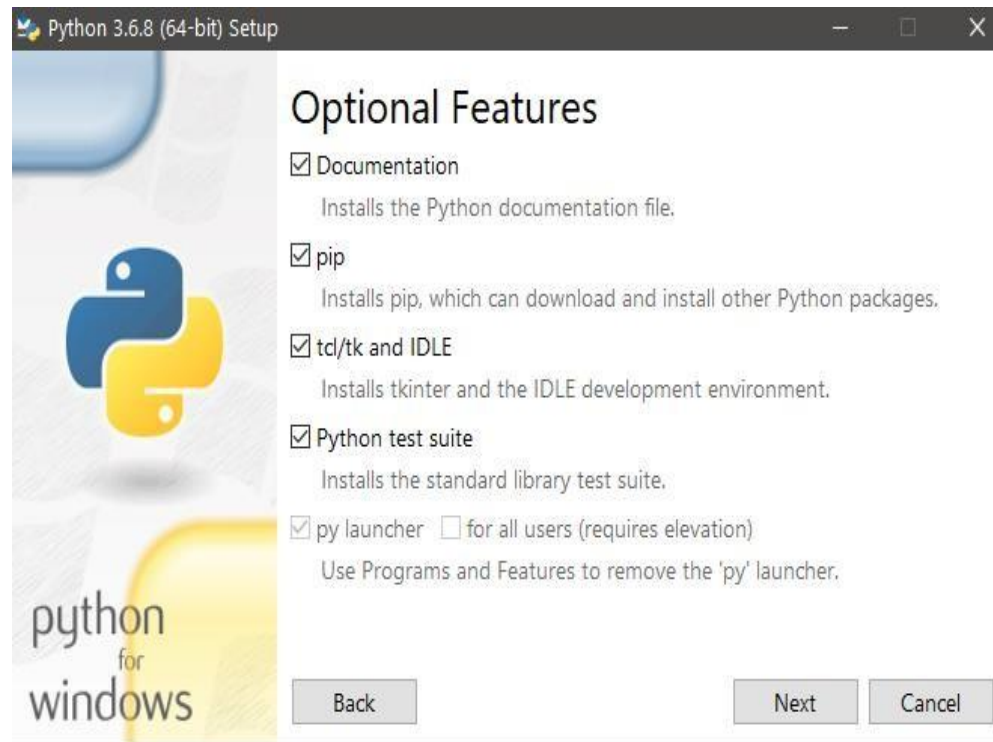
- ▶ A la première fenêtre choisir Customize Installation et cliquez sur suivant.



Installations

— Installation de Python

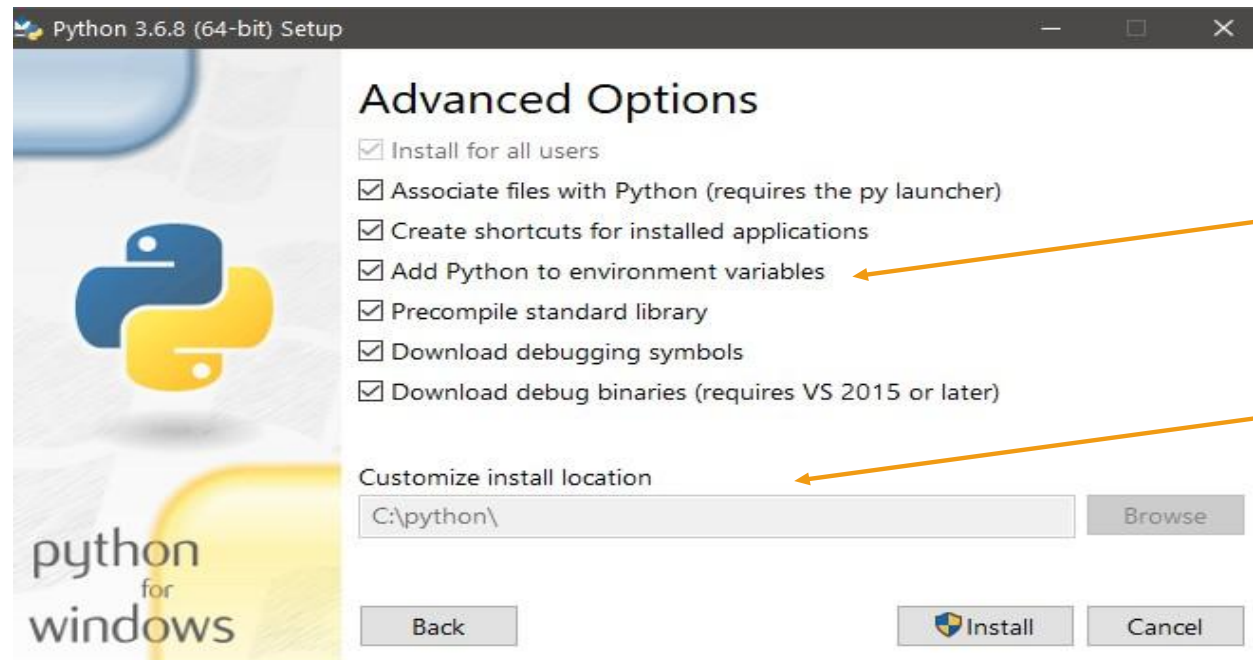
- ▶ A la fenêtre des options cocher toutes les cases puis suivant.



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.



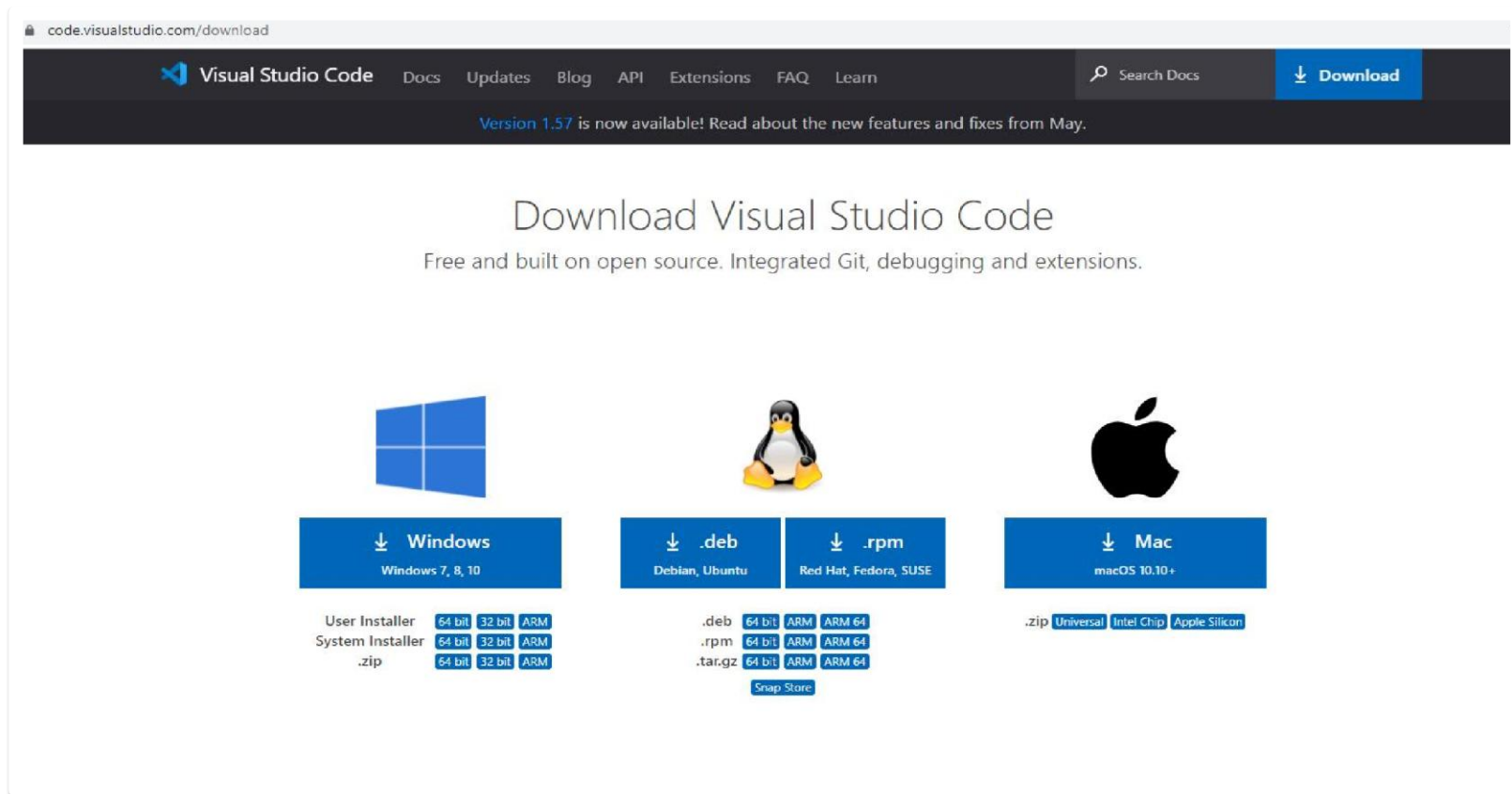
Installations

— 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 states: '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

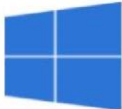
Visual Studio Code Docs Updates Blog API Extensions FAQ Learn

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



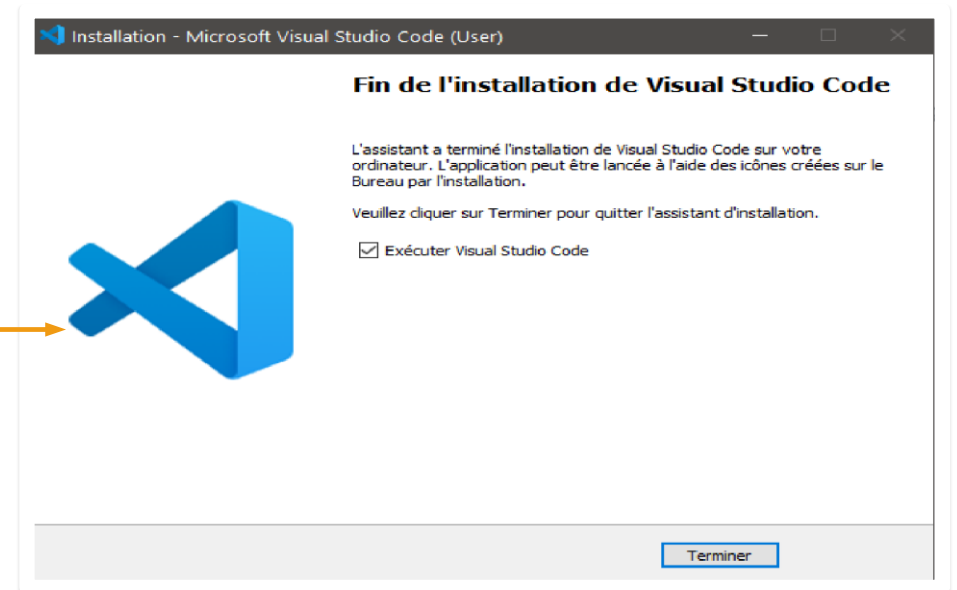
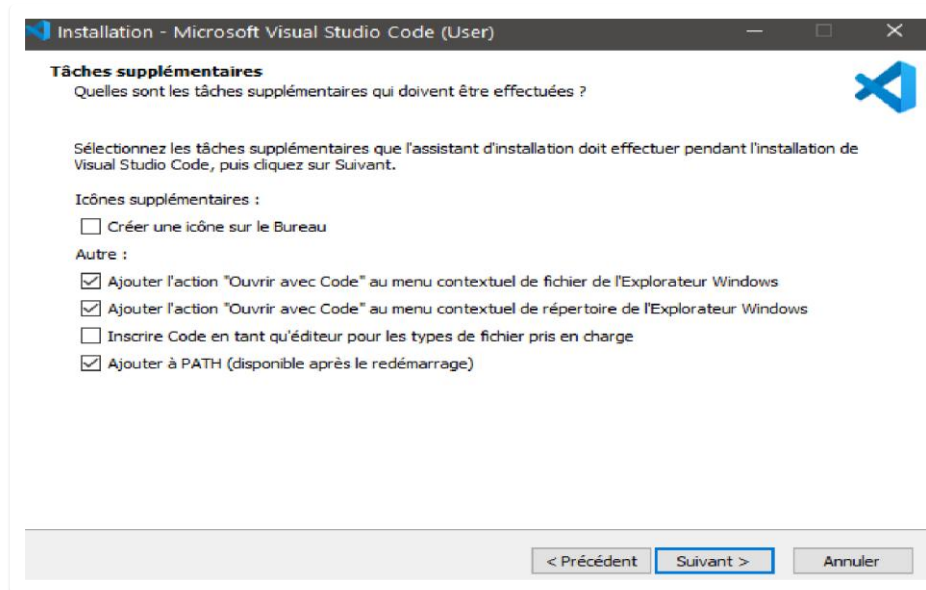
↓ Mac
macOS 10.10+

.zip Universal Intel Chip Apple Silicon

Installations

— Installation de VS Code

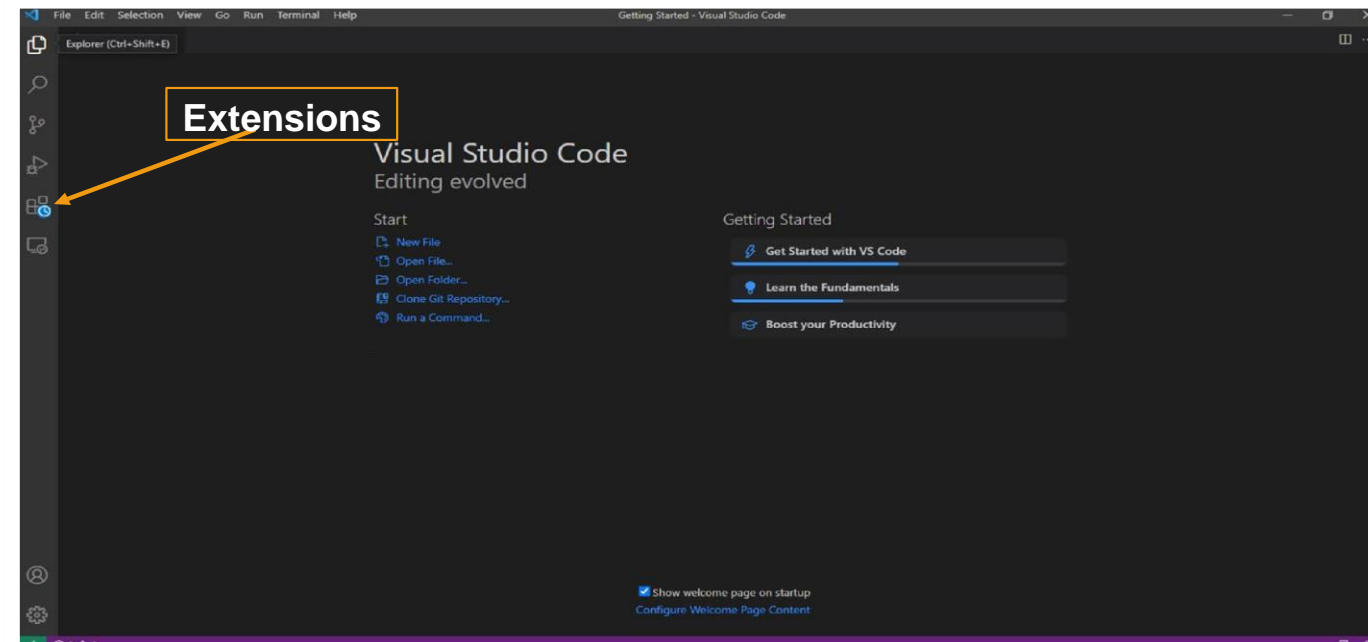
- ▶ Le processus d'installation de VS Code est très simple et n'exigera pas trop de configurations.
- ▶ Une fois la tâche finie, vous pouvez déjà lancer l'éditeur.



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 ».

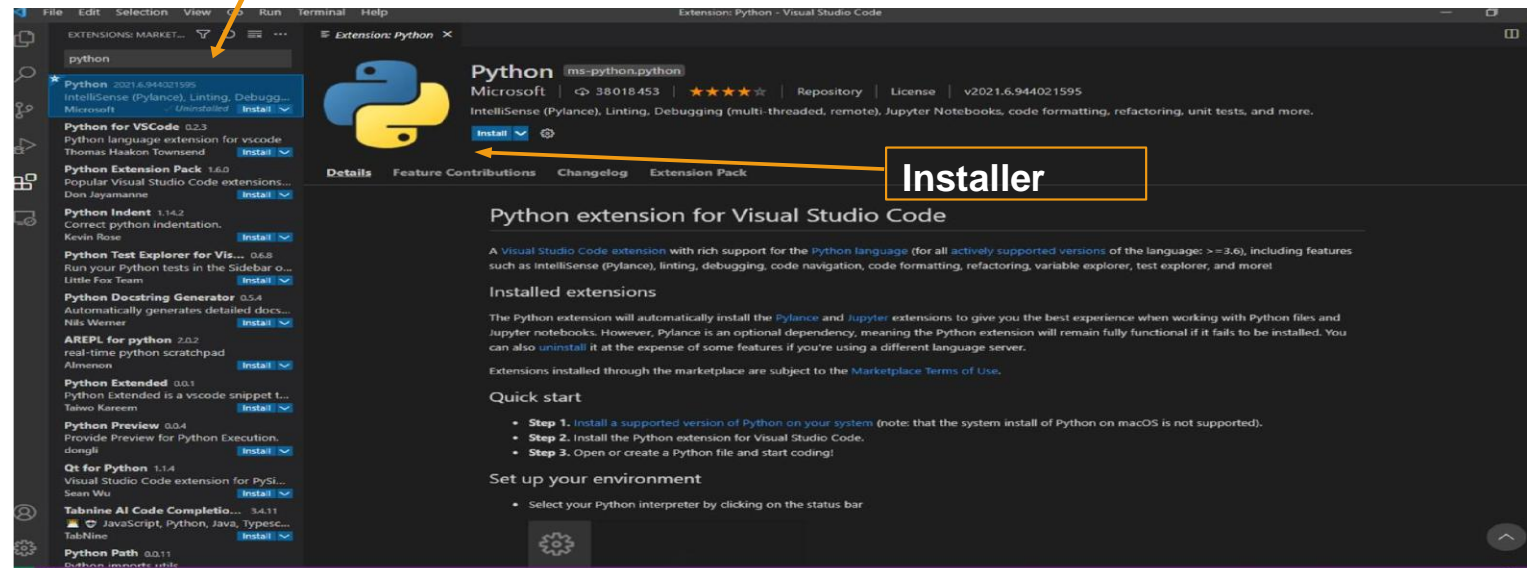


Installations

– 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.

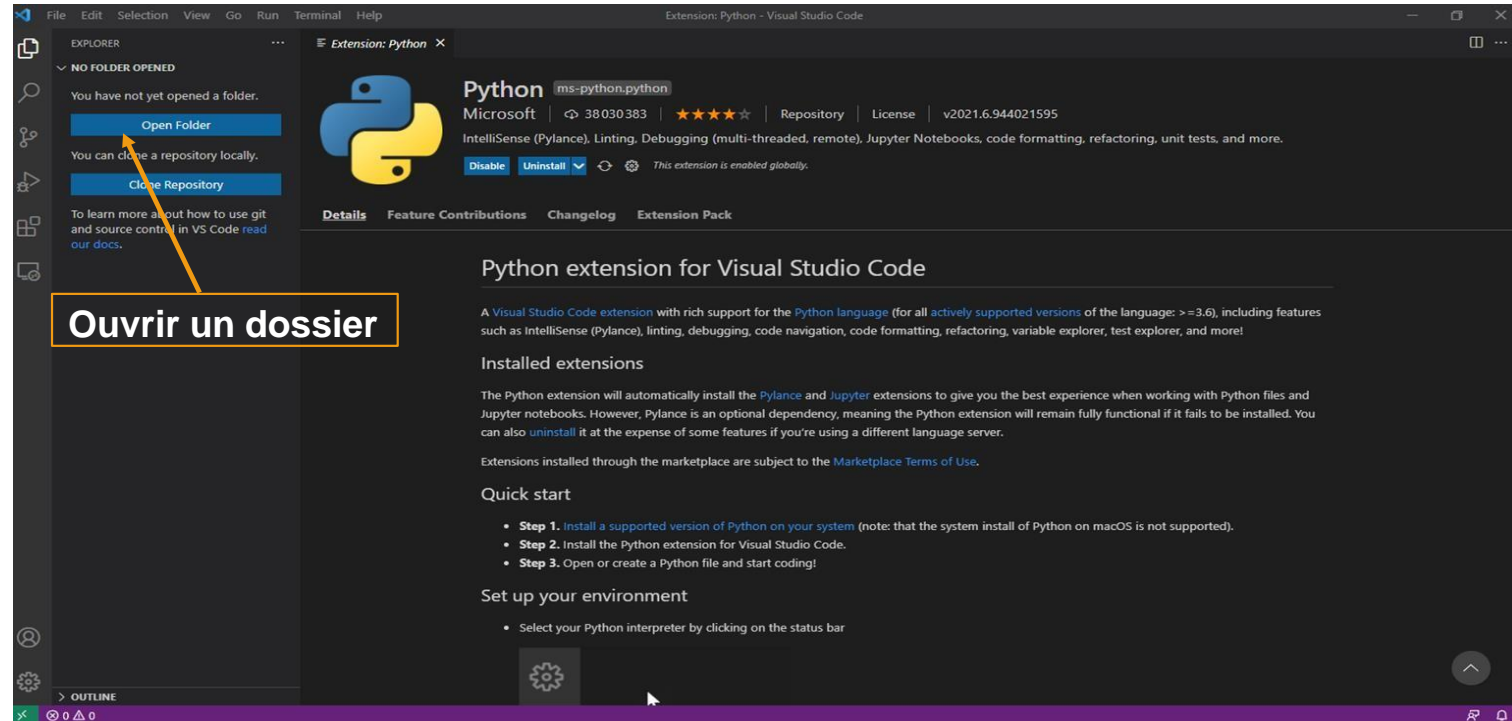
Rechercher et sélectionner



Installations

— Installation de VS Code

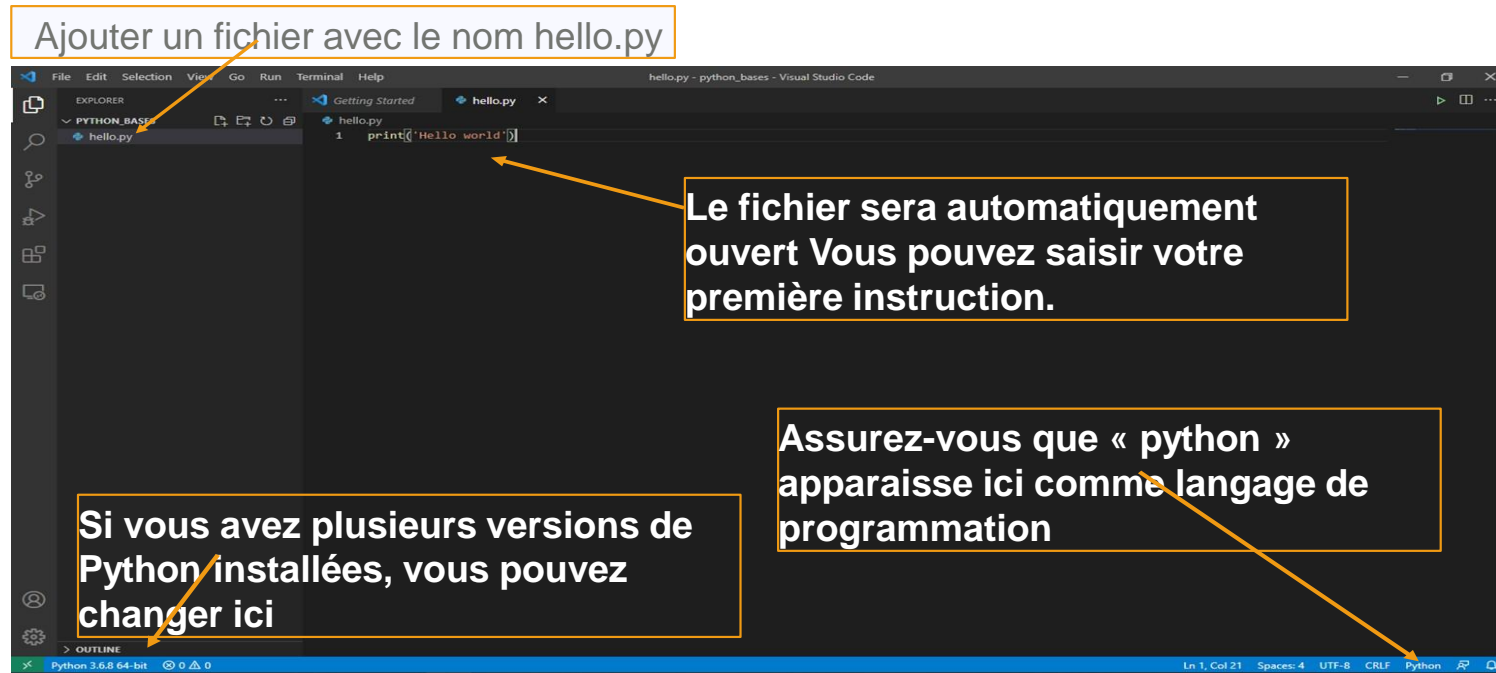
- Nous allons créer notre premier fichier capable d'exécuter du code python. D'abord, il faudra sélectionner ou créer un dossier.



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

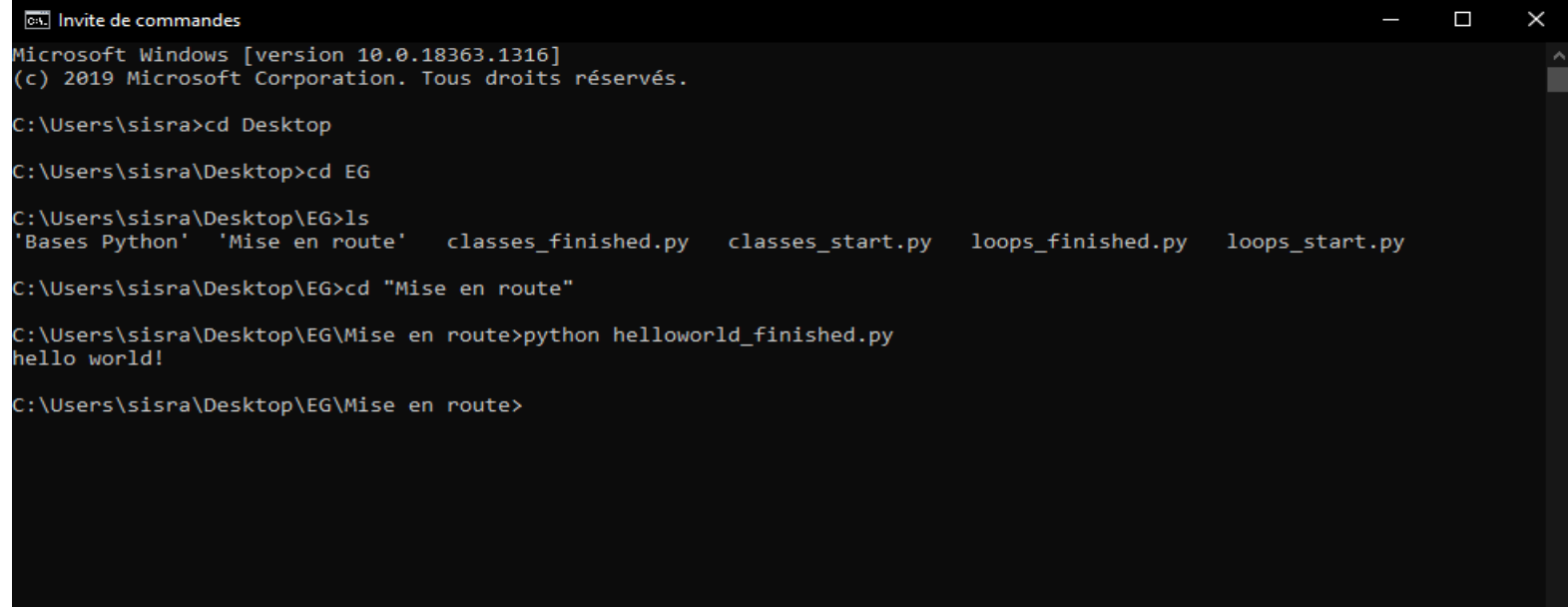
- ▶ Console Python

- ▶ 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

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. Une chaîne contenant uniquement des chiffres ne devient pas par magie un nombre, comme cela peut arriver en Perl. Chaque changement de type nécessite une conversion explicite.
- ▶ Le typage dynamique signifie que les objets d'exécution (valeurs) ont un type qui peut être changer, par opposition au typage statique où les variables ont un type interchangeable.

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
```



**ib
cegos**

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)
```

ib
cegos

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

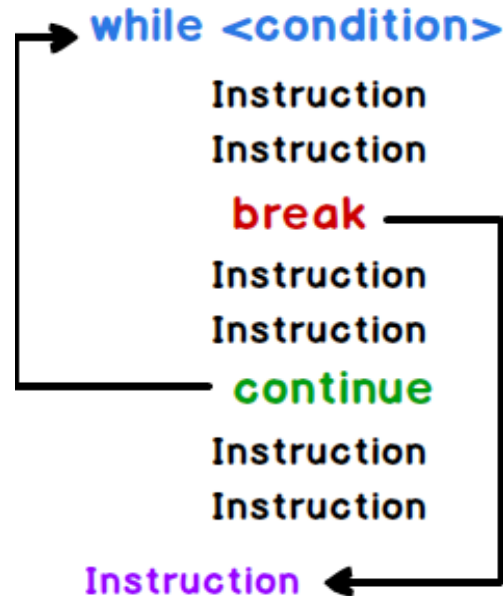
- La structure conditionnelle match / case est apparue depuis seulement la version 3.10.

```
# 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 des 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 »

— Énoncé :

- ▶ chaîne = "Retrouvez tous les mots dans une chaîne de caractères"
- ▶ 1 - Retrouvez tous les mots dans une chaîne 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`



**ib
cegos**

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 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

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' (pour write), utilisé aussi pour l'écriture mais lorsqu'on utilise ce mode, 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 remplacer le curseur à la position `x` voulue.
- ▶ La méthode `close()` referme le fichier dans n'importe qu'elle mode.



Exercice « ex_fichier.py »

— Énoncé :

- 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 »

– Énoncé :

- ▶ 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 »

— Énoncé :

► Exemple de menu :

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



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 »

– Énoncé :

- ▶ Écrivez 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**

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**

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
```



**ib
cegos**

Les modules scientifiques



Les modules scientifiques

– Module NumPy

- ▶ NumPy (Numerical Python) est une bibliothèque Python permettant de manipuler des tableaux et de réaliser des calculs mathématiques rapidement. Elle est largement utilisée en intelligence artificielle, science des données et calcul scientifique. Il remplace les listes Python pour les calculs scientifiques et l'analyse de données. Ses fonctions avancées permettent de faire des opérations vectorielles et matricielles facilement.
- ▶ L'installation de NumPy se fait par la commande : `pip install numpy`

Les modules scientifiques

– Module SciPy

- ▶ SciPy (Scientific Python) est une bibliothèque Python qui étend NumPy en fournissant des outils mathématiques avancés pour le calcul scientifique et l'ingénierie.
- ▶ L'installation de SciPy se fait par la commande : `pip install scipy`

Les modules scientifiques

— Module SciPy

► SciPy est organisé en plusieurs sous-modules spécialisés :

- | | |
|----------------------------------|---|
| ▪ Sous-module | Utilisation |
| ▪ <code>scipy.optimize</code> | Optimisation et résolution d'équations |
| ▪ <code>scipy.integrate</code> | Calcul d'intégrales |
| ▪ <code>scipy.linalg</code> | Algèbre linéaire avancée (déterminant, inverse de matrice, valeurs propres) |
| ▪ <code>scipy.stats</code> | Statistiques et tests probabilistes |
| ▪ <code>scipy.interpolate</code> | Interpolation de données |
| ▪ <code>scipy.fft</code> | Transformée de Fourier (analyse de signaux) |
| ▪ <code>scipy.signal</code> | Traitement du signal (filtrage, convolution) |
| ▪ <code>scipy.sparse</code> | Manipulation de matrices creuses (grands systèmes linéaires) |
| ▪ <code>scipy.ndimage</code> | Traitement d'images |



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

- ▶ `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 :

```
cd 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 :

```
.\Scripts\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**

Remerciements





Remerciements

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

Seriez-vous intéressé par une autre formation ?



ib
cegos

FIN

