Feuille de cours 0 (informatique) : introduction à Python

Il suffit de télécharger n'importe quel article de recherche scientifique pour constater l'omniprésence de l'outil informatique. L'essor de cet outil a permis de résoudre de nombreux problèmes hors de portée du calcul "à la main". On l'utilise entre autres pour tester une modélisation d'une situation concrète ou pour analyser de larges données récoltées lors d'une expérience. Malgré sa place encore relativement marginale dans l'enseignement secondaire, la maîtrise d'un langage de programmation est désormais *indispensable* à tout étudiant scientifique.

1 Architecture des ordinateurs

1.1 Partie matérielle : le "hardware"

Un **ordinateur** est un dispositif électronique capable de traiter l'information. Le mot "hardware" désigne toutes les pièces matérielles dont est composé un ordinateur. Le traitement de l'information peut se décomposer en différentes actions :

- ⋆ L'acquisition et la restitution d'informations se font au moyen de périphériques d'entrée (clavier, souris, microphone, scanner, etc.) et de sortie (écran, imprimante, enceinte, etc.).
- ⋆ Le traitement des données est réalisé par le **processeur** (CPU : *central processing unit*). Le processeur est en quelque sorte le cerveau de l'ordinateur ; c'est lui qui exécute les instructions et réalise les calculs.
- ⋆ Le stockage des données est assuré par la mémoire. L'ordinateur possède deux types de mémoires : la **mémoire vive** (RAM : *random access memory*) et la **mémoire de masse**.

  La mémoire vive est une mémoire de petite capacité mais très rapide d'accès, avec laquelle le processeur communique directement. De plus, c'est une mémoire volatile : elle conserve les données d'usage immédiat mais elle s'efface entièrement dès que l'alimentation de l'ordinateur est coupée.

  La mémoire de masse, qui se trouve essentiellement sur le disque dur, permet le stockage d'informations à long terme.

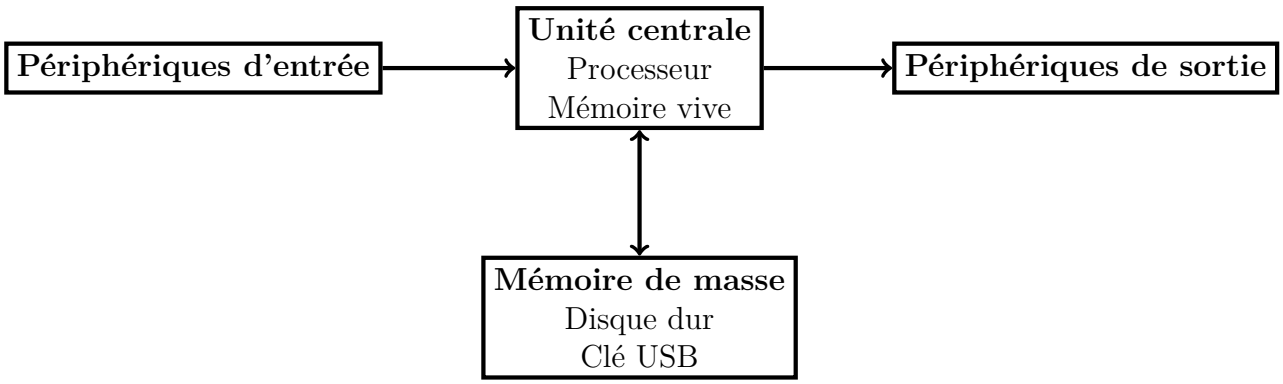


FIGURE 1 – Structure de la partie matérielle d'un ordinateur

1.2 Partie immatérielle : le “software”

Le mot “software” (**logiciel**, en français) désigne par opposition au “hardware” toute la partie immatérielle permettant à un ordinateur de traiter l'information.

1.2.1 Système d'exploitation

Le **système d'exploitation** (OS : *operating system*) — exécuté par le BIOS (*basic input/ouput system*) au démarrage — est un ensemble de programmes qui sert d'interface entre les logiciels (ainsi que tous les programmes que l'on peut écrire) et la partie matérielle de l'ordinateur. Plus précisément, le système d'exploitation gère :

- ⋆ le processeur et l'allocation des ressources aux différentes applications et programmes ;
- ⋆ les accès en mémoire (mémoire vive et mémoire de masse) ;
- ⋆ l'organisation du disque dur et l'arborescence des fichiers ;
- ⋆ les droits de lecture, d'écriture (c'est-à-dire de modification) ou d'exécution des fichiers ;
- ⋆ les périphériques d'entrée et de sortie.

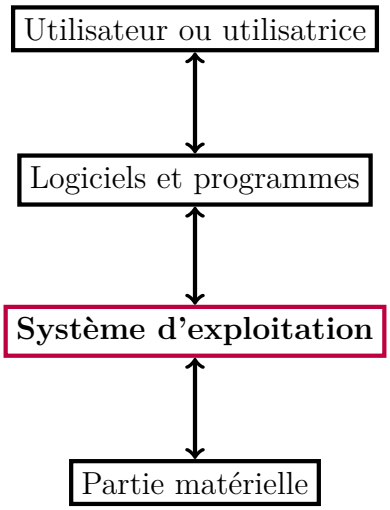


FIGURE 2 – Le système d'exploitation, interface entre hardware et software

Actuellement, les principaux systèmes d'exploitation sont GNU/Linux, MacOS et Windows, mais il y a en d'autres (Solaris, BSD, etc.).

1.2.2 Arborescence de fichiers

Chaque système d'exploitation gère les fichiers contenant l'information à traiter de manière légèrement différente. Toutefois, dans tous les cas, les fichiers sont organisés de manière arborescente, c'est-à-dire qu'ils sont tous “rattachés” par un chemin unique à une position de base qu'on appelle racine. Le chemin donnant la position de chaque fichier dans l'arborescence est appelé **chemin d'accès** ou chemin d'accès absolu. Les noms des différents dossiers (ou répertoires) indiquant ce chemin sont en général séparés par des slashs ou des antislashs. Par exemple, voici un chemin d'accès à ce document de cours :

```
C:/Users/Corentin/BCP1B/Informatique/cours/feuille-0.pdf
```

En pratique, les chemins d'accès absolus peuvent devenir très longs. On utilise donc parfois le chemin d'accès relatif du fichier, c'est-à-dire le chemin d'accès tronqué depuis le répertoire dans lequel on se "situe" actuellement. Par exemple, si je suis en train de travailler dans le dossier `BCP1B` dont le chemin absolu est `C:/Users/Corentin/BCP1B`, je peux accéder à ce fichier de cours en écrivant simplement son chemin relatif

`Informatique/cours/feuille-0.pdf`

2 Algorithmique vs programmation

2.1 Algorithmique

Un **algorithme** est une suite d'instructions précises qu'on peut effectuer sans qu'il y ait d'ambiguité. Il est important de comprendre que la notion d'algorithme est indépendante de l'outil informatique. Les algorithmes que nous étudierons préexistent à leur exécution au sein d'un ordinateur. Pensez pour vous en convaincre que l'algorithme d'Euclide, daté d'environ 300 av. J.C. et permettant de calculer le pgcd de deux entiers, n'a pas attendu l'invention des ordinateurs pour se révéler utile !

L'algorithmique est donc un domaine qu'on peut étudier uniquement à l'aide d'un papier et d'un crayon (et nous utiliserons souvent ces ustensiles en salle de TP !). Pour bien comprendre la différence entre algorithme et programme, prenons un exemple et considérons *l'algorithme* permettant de calculer le centième terme de la suite de Syracuse initialisée à 27 :

$$\begin{cases} \texttt{x} = 27 \\ \\ \text{Répéter 100 fois l'opération : } \texttt{x} = \begin{cases} \dfrac{\texttt{x}}{2} & \text{si x est pair} \\ 3\texttt{x}+1 & \text{sinon.} \end{cases} \end{cases}$$

Cette écriture, dite en **pseudo-code**, est la transcription informelle de ce qu'on aurait écrit en mathématique sous la forme :

$$\begin{cases} u_0 = 27 \\ \\ \forall n \in [\![0, 99]\!],\ u_{n+1} = \begin{cases} \dfrac{u_n}{2} & \text{si } u_n \text{ est pair} \\ 3u_n + 1 & \text{sinon.} \end{cases} \end{cases}$$

Il est bien souvent inutile de se lancer dans l'écriture d'un programme sur le clavier d'un ordinateur si vous n'avez pas d'ores et déjà défini quel algorithme vous voulez réaliser. Il ne faudra donc pas hésiter à d'abord écrire un pseudo-code - avec un papier et un crayon - avant de se lancer dans la programmation sur l'ordinateur.

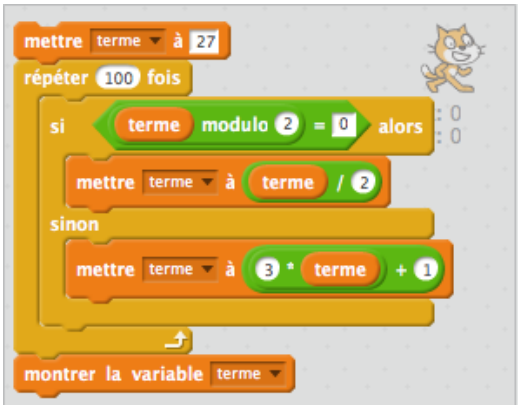
Pour finir notons que cette écriture est suffisante pour mener des preuves sur l'algorithme : démontrer qu'il permet d'obtenir le résultat souhaité en un temps fini, calculer sa complexité (i.e. le nombre d'étapes qu'il doit faire pour obtenir le résultat), etc.

2.2 Programmation

Une fois l'algorithme clairement identifié on peut passer à sa traduction sous une forme comprise par l'ordinateur. On dit qu'on **implémente** l'algorithme. Pour cela on utilise un

langage de programmation : c'est une langue, compréhensible à la fois par les humains et par la machine. Un même algorithme peut être écrit dans différents langages de programmation, de même qu'une recette de cuisine peut être rédigée dans des langues différentes. Par exemple, l'algorithme calculant le centième terme de la suite de Syracuse prend les formes suivantes lorsqu'il est implémenté dans la langages de programmation *Scratch, Python, Algobox, C++* et *OCaml* :

⋆ *En Scratch :*



⋆ *En Python :*

```python
terme = 27

for indice in range(100):
    if terme % 2 == 0:
        terme = terme / 2
    else:
        terme = 3 * terme + 1

print(terme)
```

⋆ *En Algobox :*

Code de l'algorithme

```
1   FONCTIONS_UTILISEES
2   VARIABLES
3     terme EST_DU_TYPE NOMBRE
4     indice EST_DU_TYPE NOMBRE
5   DEBUT_ALGORITHME
6     terme PREND_LA_VALEUR 27
7     POUR indice ALLANT_DE 1 A 100
8       DEBUT_POUR
9       SI (terme % 2 == 0) ALORS
10        DEBUT_SI
11        terme PREND_LA_VALEUR terme/2
12        FIN_SI
13        SINON
14          DEBUT_SINON
15          terme PREND_LA_VALEUR 3 * terme + 1
16          FIN_SINON
17      FIN_POUR
18    AFFICHER terme
19  FIN_ALGORITHME
```

⋆ *En C++ :*

```cpp
#include <iostream>
int main(){
  int terme = 27 ;
  for (int indice = 1 ; indice <= 100 ; indice++){
   if (terme % 2 == 0) terme = terme / 2 ;
   else terme = 3 * terme + 1 ;
  }
  std::cout << terme ;
}
```

⋆ *En OCaml :*

```
let terme = ref 27;;
for indice = 1 to 100 do
   if !terme mod 2 == 0 then terme := (!terme)/2 else terme := 3 * (!terme) + 1;
done;;
print_int !terme;;
```

Comme toute langue, un langage de programmation possède un vocabulaire et une syntaxe qui lui sont propres. Mais contrairement aux langues vivantes, aucune erreur (ou presque) n'est permise dans le message que vous proposez à l'ordinateur. Celui-ci en effet, n'a aucun moyen de corriger vos légères erreurs et de comprendre vos intentions. On écrira donc nos programmes informatiques avec la plus grande rigueur, et on n'hésitera pas à appliquer ensuite cette même rigueur en mathématiques !

Il existe des milliers de langages de programmation qu'on classe selon différents critères. Tout d'abord, il faut être conscient que les processeurs effectuant les calculs au sein de l'ordinateur traitent in fine (dans le "hardware") une information binaire, c'est-à-dire constituée uniquement de 0 et de 1. Comme il serait très pénible (et très long !) de raisonner uniquement sur des 0 et des 1, les langages de programmation utilisent des instructions plus complexes qui sont ensuite traduites en binaire. Cette distance entre le langage et l'exécution réelle dans la machine constitue un premier critère de classement : on parle de langage **bas niveau** quand on reste relativement proche de l'exécution machine, et de langage **haut niveau** quand on s'autorise des instructions plus complexes qui nous en éloignent.

Par ailleurs, on distingue deux types de langages de programmation : les langages compilés et les langages interprétés. Pour un **langage compilé**, le code est d'abord compilé, c'est-à-dire traduit en langage machine, puis ce nouveau code en langage machine est exécuté. Pour un **langage interprété**, l'exécution se fait au fur et à mesure (par un **interpréteur** ou **interprète de commandes**), sans phase de compilation. Les langages interprétés sont en général plus lents, mais sont moins sujets aux problèmes de portabilité, c'est-à-dire de changement de système d'exploitation.

Enfin, même si la différence entre algorithmqiue et programmation est profonde, cela ne veut pas dire que ces deux branches de l'activité informatique ne s'influencent pas. Vous constaterez en pratique que l'étape d'implémentation d'un algorithme pourra vous amener par exemple à revoir les variables utilisées dans votre pseudo-code, à inverser l'ordre de deux étapes de l'algorithme, etc. Inversement, les possibilités offertes par le langage de programmation peuvent aussi être une source d'inspiration algorithmique.

2.3 Le langage Python

Le langage que nous allons apprendre cette année est **le langage Python**. Il a été créé en 1989 par Guido van Rossum et doit son nom à la troupe d'humoristes anglais Monty Python. C'est un langage interprété, de haut niveau, dont la syntaxe est très simple et aérée. Cette grande simplicité fait de Python un très bon langage pour débuter en programmation, mais a également encouragé le développement de nombreuses bibliothèques, qui enrichissent le langage de fonctions d'usage spécifique. Ainsi, le langage Python est très largement utilisé à la fois dans le milieu académique et dans le milieu professionnel.

3 Logiciel pour Python

Évoluant du plus abstrait au plus concret, nous avons distingué l'algorithmique de la programmation. Sur cette même voie, il faut encore distinguer le langage de programmation - Python - du logiciel qui permet de l'utiliser, qu'on appellera **IDE** (pour *Integrated Development Environment*). Cette année, nous utiliserons **Pyzo** ou **Spyder**.

L'IDE est le logiciel qui permet d'écrire le code Python des algorithmes que nous considérerons. Pour fonctionner, il utilise un **interpréteur** de Python. Plusieurs choix d'interpréteurs sont possibles. Personnellement, j'utilise "Anaconda" qui contient (sauf exception) tout ce dont nous avons besoin en BCPST.

3.1 Installation

Vous devez impérativement installer de quoi travailler Python chez vous sur votre ordinateur personnel. Si vous le souhaitez, vous pourrez également apporter votre ordinateur portable pendant les séances de TP. Vous trouverez ci-dessous des liens et quelques rapides explications vous permettant d'installer Pyzo (ou Spyder sur Mac). N'hésitez pas à consulter de l'aide sur Internet avant de revenir vers moi pour des problèmes d'installation (je ne suis pas expert en ce domaine !). Une remarque importante : **tous les logiciels à installer sont entièrement gratuits**. Enfin, si vous avez déjà installé Python sous une autre version sur votre ordinateur, n'hésitez pas à me demander si celle-ci convient à la BCPST avant de vous lancer dans une autre installation.

Pour ceux d'entre vous travaillant sous Windows, rendez-vous sur le site `https://pyzo.org` pour télécharger et installer la dernière version de Pyzo sur votre ordinateur personnel. Vous trouverez sur la page "Quickstart" `https://pyzo.org/start.html` des explications (en anglais) vous guidant à travers l'installation. Comme mentionné précédemment, il y a en fait deux choses à installer : "l'IDE" Pyzo lui-même ("step 1" sur la page "Quickstart"), et un "interpréteur" Python (appelé aussi "Python environment", "step 2" de cette page). Pour l'interpréteur, je vous recommande d'installer "Anaconda" sauf si vous manquez vraiment d'espace libre sur votre ordinateur (auquel cas, privilégiez "Miniconda").

Pour ceux d'entre vous travaillant sur Mac, je vous recommande d'installer plutôt **Spyder** : il s'agit d'un autre IDE proposant les mêmes fonctionnalités que Pyzo. Rendez-vous sur le site `https://www.spyder-ide.org/`, puis plus précisément sur la page d'installation `https://docs.spyder-ide.org/current/installation.html`. Spyder propose automatiquement d'installer l'interpréteur "Anaconda" : le plus simple est de suivre cette recommandation.

3.2 Découverte des IDE Pyzo et Spyder

Dans la configuration par défaut, lorsque Pyzo ou Spyder s'ouvrent on voit apparaître trois fenêtres (voir illustrations ci-dessous) :

- ⋆ **L'éditeur** ou **script** dans lequel on écrit le code, et qui permet de les sauvegarder.
- ⋆ **La console** ou **shell** dans lequel le code est executé. On peut y rentrer directement une opération ou y appeler ce qu'on a écrit dans l'éditeur.
- ⋆ Une zone permettant de voir les variables en jeu, les graphiques, etc.

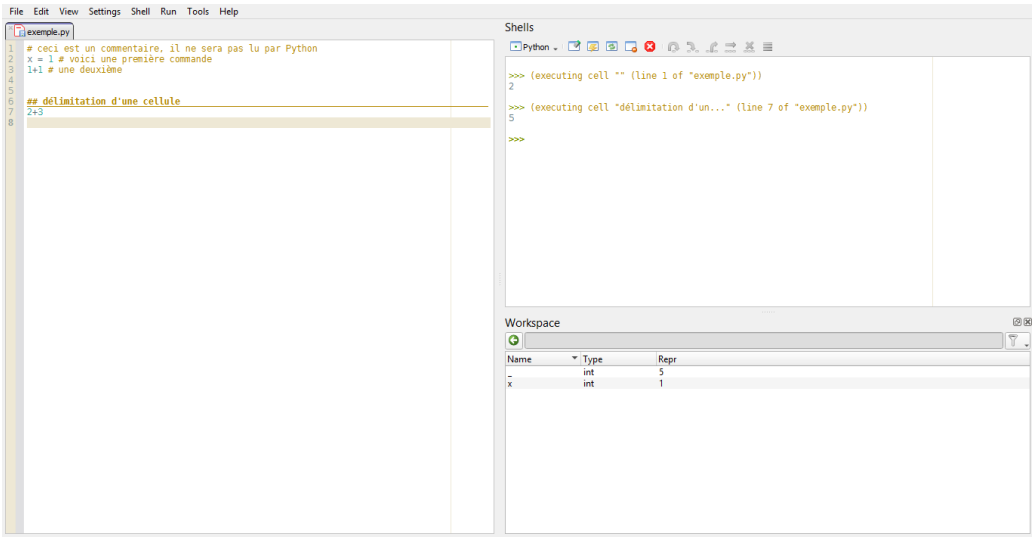


FIGURE 3 – L'IDE Pyzo

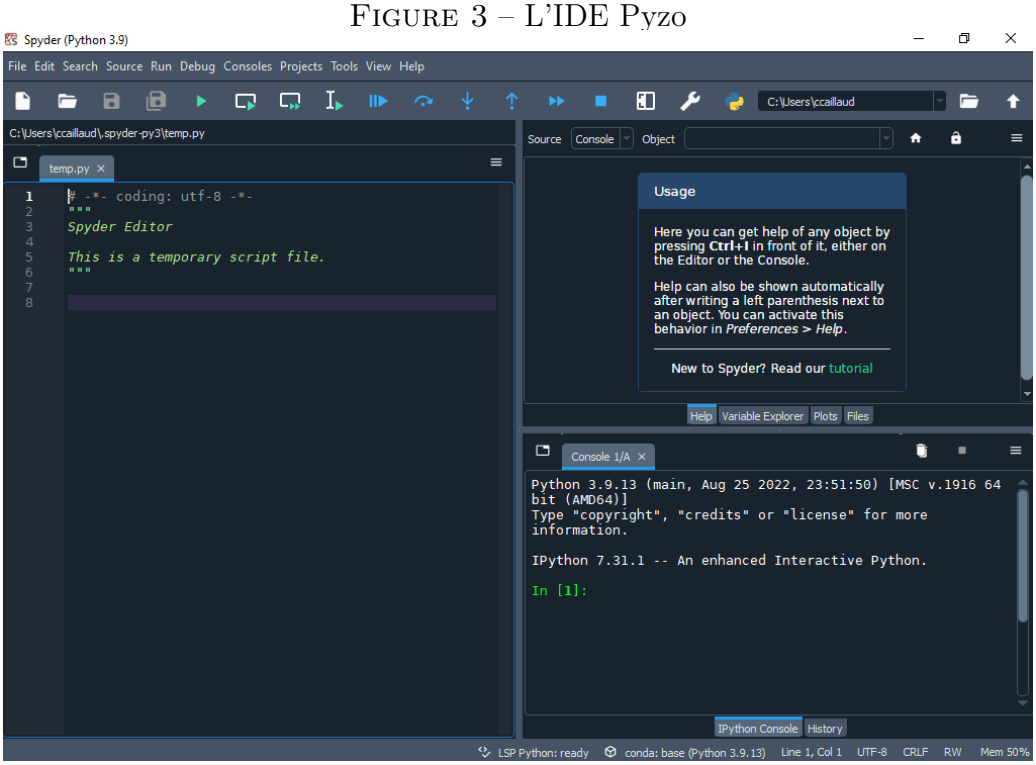


FIGURE 4 – L'IDE Spyder

1. Puisque c'est la console qui exécute le code, pourquoi y a-t-il un éditeur ?

2. Comment transmet-on le code de l'éditeur à la console ? (Cette étape est appelée **compilation**).

3. Comment faire pour qu'une partie du code de l'éditeur ne soit pas transmise à la console ? (C'est les **commentaires**).

4. Comment ne compiler qu'une partie du code de l'éditeur à la fois (sans devoir commenter tout le reste) ?

5. Je me suis trompé et j'ai lancé un code très long à s'executer. Comment l'arrêter ?

6. Je veux redemander à la console le même calcul que précédemment. Dois-je tout retaper ?

4 Savoir-faire et bonnes pratiques en informatique

Vous trouverez ci-dessous des consignes et conseils qui ne sont pas propres à Python mais valables pour tout usage de l'informatique.

4.1 Enregistrer son fichier

Il est obligatoire d'enregistrer son fichier au début de chaque séance, et régulièrement au cours de la séance. Cela permet de ne pas perdre son travail en cas de mauvaise manipulation ou de coupure de courant, ainsi que d'en garder une trace pour l'avenir. Lors de certaines séances de TP, les fichiers seront d'ailleurs "ramassés".

Pour cela, vous devez vous créer un dossier "TP Python" sur votre session personnelle du lycée. Attention, les sessions "élève" sont programmées de sorte que vous n'avez pas le droit d'enregistrer un fichier sur le bureau (sur certains postes, vous avez le droit de le faire, mais le fichier est alors supprimé lorsque vous fermez votre session). Enregistrez ensuite votre fichier (avec un nom explicite "TP1", "TP2",...) dans ce dossier tout au long de l'année.

Lorsque vous voulez ouvrir un fichier déjà enregistré, par exemple pour reprendre le TP de la séance précédente, il ne faut pas double-cliquer dessus dans le dossier. En effet, votre ordinateur ouvre alors la plupart du temps le fichier avec le mauvais logiciel. Pour résoudre ce problème, commencez par ouvrir Pyzo ou Spyder, puis cliquez sur "fichier, ouvrir" et retrouvez votre fichier.

4.2 Dactylographie

Il est indispensable de savoir écrire vite et bien sur un clavier d'ordinateur. Voici consignes à ce sujet :

- ⋆ Tapez avec vos 10 doigts, les deux pouces sur la barre Espace, les mains gauche et droite se répartissant les différentes lettres.
- ⋆ Privilégiez le pavé numérique du côté pour les chiffres. Précisons que ce pavé peut être activé ou désactivé grâce à la touche Verr Num (ou autre nom similaire).
- ⋆ Avez-vous déjà testé la touche Inser du clavier, souvent placé au-dessus de la touche "Suppr" ? À quoi sert-elle ? Ne soyez pas surpris, et sachez la désactiver si elle est active automatiquement à l'allumage de certains ordinateurs du lycée.
- ⋆ Nous aurons besoin de symboles spéciaux pour écrire nos codes Python. Il faut impérativement savoir taper les symboles suivants : `[,],#,<,>,%`. En particulier, pour ceux qui utilisent un Mac, il faut connaître les raccourcis clavier correspondants.
- ⋆ Repérez la touche "Tabulation"  permettant de faire un large espace horizontal sans avoir à appuyer plusieurs fois sur la barre Espace (très utile pour l'indentation en Python).