Chapitre 1

Au commencement, des concepts

Swamp King : One day all this will be yours!
Herbert : What, the curtains?

The Holy Grail, Monty Python

À la fin de ce chapitre, je sais :

- ☞ Distinguer les concepts d'algorithme, de programme et de processus
- ☞ Lister les grandes caractéristiques du langage Python
- ☞ Importer des fonctions d'une bibliothèque (module Python)

Ce premier chapitre n'explique pas dans le détail tous les concepts présentés. Cependant les TP du premier semestre permettent d'approfondir ces concepts et complémentent cette lecture.

Parler d'informatique nécessite de parler de l'information et des langages informatiques qui véhiculent ces informations. Parler des langages, c'est tenter de décrire les mots de ce langage, de la structure qui relie ces mots entre eux et de leur pouvoir. Pour parler de mots, il faut tout d'abord se mettre d'accord sur le sens de ces mots.

A Algorithmes, programmes et processus

■ **Définition 1 — Informatique.** L'informatique est la science du traitement automatique et rationnel de l'information par un système concret ou abstrait.

Vocabulary 1 — Computer Sciences and Computer Engineering ⇜⇝ Les anglo-

saxons font une distinction entre l'informatique comme science abstraite (Computer Sciences) et l'informatique comme science concrète appliquée (Computer Engineering). En français, l'informatique désigne les deux, ce qui est important, l'une n'allant pas sans l'autre. Que ferait-on de la physique théorique sans l'expérience ?

■ **Définition 2 — Algorithme.** Un algorithme est une méthode pour résoudre un problème donné. Cette méthode est constituée d'une suite d'instructions qui permet de trouver une solution au problème.

■ **Exemple 1 — Produit de deux nombres.** L'algotithme 1 calcule le produit de deux nombres.

Algorithme 1 Produit de deux nombres

```
1 : Fonction PRODUIT(a, b)                          ▷ a ∈ ℕ et b ∈ ℕ.
2 :     p ← 0
3 :     c ← 0                                       ▷ c est un entier.
4 :     tant que c < a répéter
5 :         p ← p + b
6 :         c ← c + 1
7 :     renvoyer p
```

On s'efforce généralement :

- de rendre un algorithme non ambigu,
- d'identifier clairement les entrées et les sorties ,
- de décrire un algorithme avec suffisamment d'abstraction afin de rendre de son implémentation indépendante du langage de programmation choisi.

■ **Définition 3 — Programme.** Un programme est un ensemble d'instructions dans un langage de programmation donné.

R Il est important de savoir passer de l'expression d'un algorithme à un programme et, dans le cadre des épreuves des concours, à un programme en Python ! L'expression de l'algorithme dans les épreuves est le plus souvent sous la forme d'un texte, c'est à dire qu'on décrit l'algorithme avec des **phrases** [a]. Le code 1.1 traduit l'algorithme 1 en Python.

a. ce qui ne simplifie pas la tâche du candidat qui doit savoir lire et interpréter…

Code 1.1 – Un exemple de programme en Python - traduction de l'algorithme produit

```
1 def produit(a, b):   # une fonction
2     p = 0       # Un commentaire
3     c = 0
4     while c < a:
5         p = p + b
```

```
6           c = c + 1
7       return p
8
9
10  #  Debut du programme principal
11  print("Le produit de 2 par 21 vaut :", produit(21, 2))
```

R Si un algorithme est abstrait, un programme est concret, c'est à dire qu'il est interprétable soit par un processeur directement, soit par un interpréteur (machine virtuelle). On peut donc demander à une machine d'exécuter un programme, mais on ne peut pas lui demander d'exécuter directement un algorithme.

■ **Définition 4 — Processus.** Un processus est un programme en cours d'exécution sur une machine, c'est à dire un processeur muni d'une mémoire et d'un système d'exploitation.

B Langages, compilateurs et machines --→ HORS PROGRAMME

■ **Définition 5 — Langage de programmation.** Un langage de programmation est une notation normée dont l'objectif est d'implémenter des algorithmes.

■ **Exemple 2 — Langages de programmation.** Parmi les langages de programmation les plus utilisés actuellement on peut citer : C, C++, Java, Python ou Javascript. On choisit un langage plutôt qu'un autre en fonction de l'application visée. Javascript est par exemple un langage plutôt orienté web. C est plutôt dédié aux systèmes embarqués et bas niveau.

■ **Définition 6 — Compilateur.** Un compilateur est un logiciel qui transforme un langage en un autre. Le langage des données d'entrée est dit *langage source* et celui des données de sortie *langage cible*.

■ **Définition 7 — Langage machine.** Le langage machine est le langage d'un processeur. Il est composé d'instructions très élémentaires de type :

- mettre une donnée dans une case mémoire,
- modifier le contenu d'une case mémoire (opérations arithmétiques et logiques),
- tester la valeur d'une case mémoire,
- effectuer des branchements et des sauts dans le code,
- appeler une routine de calcul.

Ces instructions élémentaires sont exécutées par le processeur à une fréquence très élevée [a].

a. de l'ordre du GHz dans les processeurs actuels.

■ **Définition 8 — Langage compilé.** Un langage compilé [a] est un langage muni d'un compilateur capable de générer du langage machine à partir du code source.

a. On devrait dire compilable, mais les anglicismes et les traductions littérales sont légions dans le domaine informatique.

■ **Exemple 3 — Quelques compilateurs.** Les compilateurs sont ubiquitaires dans tous les domaines de l'ingénierie et la plupart du temps invisibles. On trouve :

- **gcc** est un compilateur capable de compiler de nombreux langages dont le langage C. Il permet de traduire des instructions en langage C en langage machine, c'est à dire le langage d'un processeur.
- **javac** est un compilateur capable de traduire du code java en bytecode java.
- **pdflatex** est un compilateur capable de traduire du code latex en PDF et de créer les pages que vous lisez.

■ **Définition 9 — Langage interprété.** Un langage interprété est un langage muni d'un interpréteur capable de l'exécuter sur une machine concrète.

■ **Définition 10 — Interpréteur ou machine virtuelle.** Un interpréteur [a] ou machine virtuelle est un **logiciel** qui permet d'interpréter des instructions dans un langage généralement désigné par le terme Bytecode. Ce logiciel est exécuté par un processeur, ce qui explique le qualiticatif virtuel : on ne programme pas un processeur mais un interpréteur.

a. On devrait dire interprète mais il s'agit encore une fois d'un anglicisme.

■ **Exemple 4 — Machines virtuelles.** Quelques exemples d'interpréteurs :

- Java Virtual Machine (JVM) est un interpréteur de code Java.
- Python Virtual Machine (PVM) est un interpréteur de code Python.
- Ocaml dispose d'un interpréteur et d'un compilateur en langage machine.

■ **Définition 11 — Bytecode.** Un bytecode est un langage intermédiaire entre un langage de programmation (de haut niveau) et les langages de plus bas niveau (proche du processeur). Il est exécuté par un interpréteur.

Ⓡ Si un bytecode est indépendant de la machine sur laquelle il sera exécuté, un interpréteur dépend au contraire de la machine concrète sur laquelle il s'exécute c'est à dire du processeur et du système d'exploitation.

Le code Python, avant d'être exécuté par une machine virtuelle, c'est à dire un logiciel, est compilé dans un langage nommé Bytecode par la PVM.

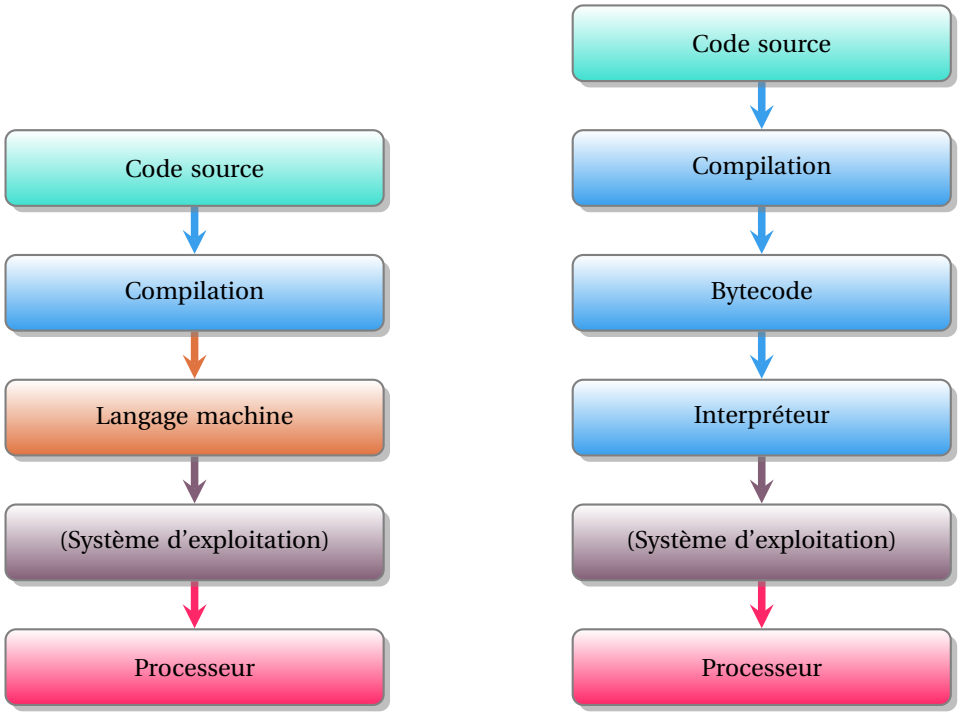

FIGURE 1.1 – Comparaison des chaînes d'exécution des langages compilés (à gauche) et des langages interprétés (à droite)

```
0 LOAD_CONST               1 (0)
2 STORE_FAST               2 (p)
4 LOAD_CONST               1 (0)
6 STORE_FAST               3 (c)
8 LOAD_FAST                3 (c)
10 LOAD_FAST                0 (a)
12 COMPARE_OP               0 (<)
14 POP_JUMP_IF_FALSE       20 (to 40)
16 LOAD_FAST                2 (p)
18 LOAD_FAST                1 (b)
20 BINARY_ADD
22 STORE_FAST               2 (p)
24 LOAD_FAST                3 (c)
26 LOAD_CONST               2 (1)
28 BINARY_ADD
30 STORE_FAST               3 (c)
32 LOAD_FAST                3 (c)
34 LOAD_FAST                0 (a)
36 COMPARE_OP               0 (<)
38 POP_JUMP_IF_TRUE         8 (to 16)
40 LOAD_FAST                2 (p)
42 RETURN_VALUE
```

FIGURE 1.2 – Bytecode Python du code 1.1 de la fonction produit.

P Python est donc un langage interprété.

Le figure 1.2 donne, à titre d'illustration, le bytecode du programme 1.1. On perçoit bien la nature élémentaires des instructions impératives qui ressemble beaucoup à du langage machine. Sauriez-vous l'interpréter à votre tour ?

C Des paradigmes différents --→ HORS PROGRAMME

Tout comme il existe une multitude de langages humains, il existe une grande diversité de langages informatiques. Pour mieux cerner leurs différences, on utilise la notion de paradigme de programmation.

■ **Définition 12 — Paradigme de programmation.** Un paradime de programmation est un ensemble de formes et de figures qui constitue un modèle propre à un langage.

■ **Définition 13 — Paradigme impératif.** Le paradigme impératif s'attache à décrire des séquences d'instructions (ordres) qui agissent sur un état interne de la machine (contexte).

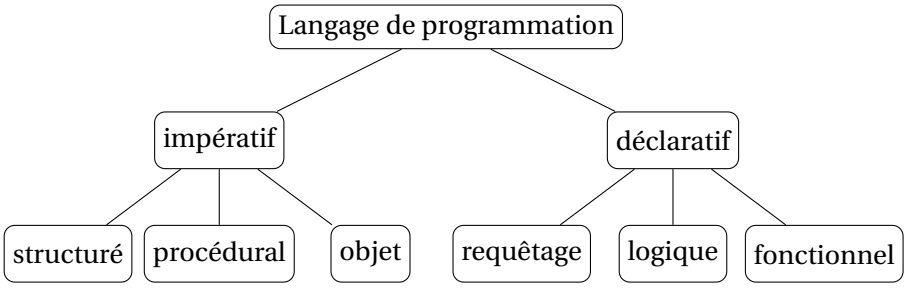

FIGURE 1.3 – Paradigmes des langages de programmation

L'impératif explicite le *comment procéder* pour exécuter un programme. Cette programmation se rapproche de la logique électronique des processeurs.

■ **Définition 14 — Paradigme procédural.** Ce paradigme est une déclinaison de l'impératif et propose de regrouper des éléments réutilisables de code dans des routines. Ces routines sont appelées procédures (si elles ne renvoient rien) ou fonctions (si elles renvoient un résultat).

■ **Définition 15 — Paradigme objet.** Ce paradigme est une déclinaison de l'impératif et propose de décrire un programme comme l'interaction entre des objets à définir. Une classe est un type d'objet qui possède des attributs et des comportements. Ces caractéristiques sont encapsulées et peuvent être masquées à l'utilisateur d'un objet : cela permet de protéger l'intégrité de l'objet et de garantir une cohérence dans la manipulation des données.

■ **Définition 16 — Paradigme déclaratif.** Le paradigme déclaratif est une syntaxe qui s'attache à décrire le *quoi*, c'est à dire *ce que le programme doit faire*, non pas comment il doit le faire. Un langage déclaratif ne dépend pas de l'état interne d'une machine (contexte). Cette programmation se rapproche de la logique mathématique et délègue au compilateur la délicate question du *comment procéder*.

■ **Définition 17 — Paradigme fonctionnel.** Le paradigme fonctionnel est une déclinaison du déclaratif qui considère qu'un programme n'est qu'un calcul et qu'un calcul est le résultat d'une fonction. Le mot fonction est ici à prendre au sens mathématique du terme (lambda calcul) : une fonction appelée avec les mêmes paramètres produit le même résultat en toute circonstance.

Comme on peut le constater sur la figure 1.2, les langages de types Bytecode sont impératifs. L'objectif est de se rapprocher des langages machines qui sont également impératifs à cause de l'architecture des processeurs de type Von Neumann.

P Python est un langage multiparadigme : impératif, objet et fonctionnel.

■ **Exemple 5 — Langages et paradigmes.** La plupart des langages contemporains sont multiparadigmes, c'est à dire qu'ils permettent de programmer de différentes manières.

- C : impératif, structuré et procédural,
- C++ : impératif, structuré, procédural, objet et fonctionnel,
- Java : impératif, objet, fonctionnel,
- Javascript : fonctionnel, objet (prototype), script,
- Smalltalk : objet,
- Prolog : logique,
- SQL : déclaratif,requêtage,
- Ocaml : fonctionnel, impératif et objet,
- Haskell : fonctionnel pur.

D Systèmes d'exploitations --→ HORS PROGRAMME

■ **Définition 18 — Système d'exploitation.** Un système d'exploitation est un ensemble de logiciels qui forme une interface abstraite entre les applications et la machine. Un système d'exploitation accède directement au matériel de l'ordinateur, c'est à dire les systèmes électroniques constitutifs comme le processeur, les mémoires et les périphériques (écran, clavier, trackpad...).

Les autres applications de l'ordinateur s'appuient sur le système d'exploitation pour accéder au matériel.

Vocabulary 2 — Operating System (OS) ⇜⇝ Système d'exploitation.

Un système d'exploitation :

- garantit ainsi l'intégrité et la cohérence de la machine,
- masque la complexité du matériel aux utilisateurs, aux développeurs et aux applications en présentant une abstraction du matériel,
- gère le matériel en appliquant certaines stratégies prédéfinies (allocation mémoire, ordonnancement des processus),
- isole les applications les unes des autres.

Vocabulary 3 — Software, hardware ⇜⇝ En anglais, on distingue les logiciels (software) du matériel électronique qui lui permet de fonctionner (hardware).

Un système d'exploitation est composé (cf. figure 1.4) :

- d'un système de fichiers (file system),

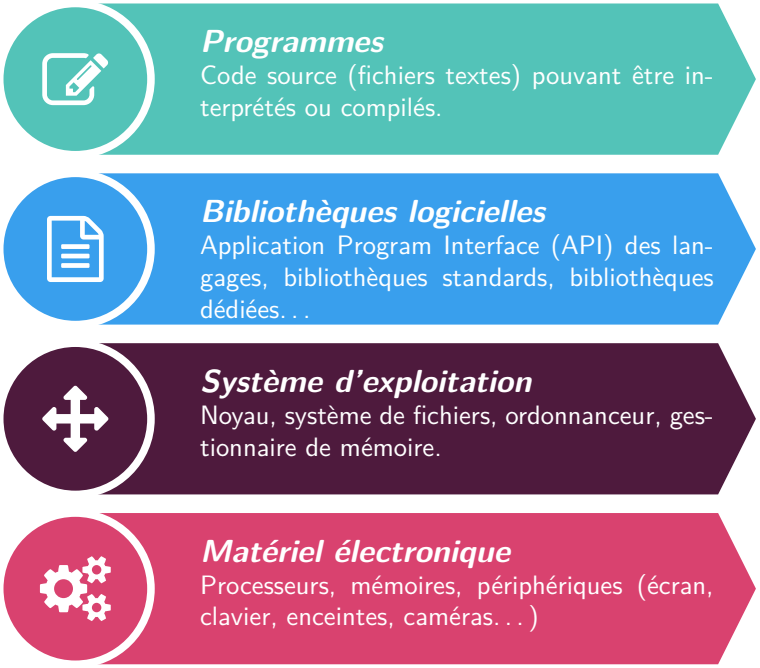

FIGURE 1.4 – Positionnement du système d'exploitation et des bibliothèques logicielles entre les éléments logiciels et le matériel électronique

- de pilotes (drivers) spécifiques aux matériels électroniques,
- d'un ordonnanceur (scheduler) de processus,
- d'un gestionnaire de mémoire.

L'ordonnanceur et le gestionnaire de mémoire appliquent des stratégies sophistiquées afin de garantir une certaine équité entre les processus qui s'exécutent, l'accès aux ressources de la machine et l'allocation de l'espace mémoire.

Un système d'exploitation offre la plupart du temps deux modes de fonctionnement différents :

- Le mode utilisateur, non privilégié : dans cet espace, l'utilisateur peut utiliser les ressources du système et l'interface du système d'exploitation. Cependant, il n'accède à aucun matériel de manière privilégiée. C'est l'espace de développement et de fonctionnement des applications et des services. Cet espace garantit que tout accès au système logiciel/matériel est légal, car contrôlé par le système d'exploitation.
- Le mode administrateur, super-utilisateur, privilégié (KERNEL SPACE) : dans cet espace, l'administrateur peut programmer le système d'exploitation, le modifier et accéder au matériel, sans restrictions.

■ **Exemple 6 — Quelques systèmes d'exploitation.** Parmi les plus courants, on peut citer :

- Windows,
- Mac OS X,
- Linux,
- la famille de Unix BSD.

Dans des domaines plus spécifiques, comme l'informatique temps réel ou les systèmes embarqués, on peut trouver d'autres systèmes d'exploitation comme Windows CE, FreeRTOS ou Xenomai.

■ **Définition 19 — Système de fichier.** Un système de fichier est une organisation hiérarchique des fichiers mis en œuvre par un système d'exploitation. Cette organisation se présente sous la forme d'un arbre (systèmes Unix) ou d'une forêt (systèmes Windows).

Lorsqu'on travaille sur un ordinateur, il est toujours important de savoir où l'on se trouve et où l'on peut aller dans un système de fichiers.

■ **Définition 20 — Chemin.** Un chemin est une chaîne de caractères qui permet de décrire la position d'un fichier ou d'un répertoire dans un système de fichier.

On peut lire directement un chemin sur l'arborescence de la figure 1.5. La racine de l'arbre est dénotée `/`. Ainsi, le chemin vers le répertoire personnel d'Olivier est `/home/olivier/`. Le sous-répertoire `python/tp1/` est désigné par le chemin `/home/olivier/Desktop/python/tp1`.

Lorsque l'on ouvre un environnement de développement comme Pyzo, celui-ci se place dans un répertoire de travail. C'est dans ce répertoire que sont lancés les commandes et les scripts Python. Il dépend de la configuration du système sur lequel on travaille. On peut éventuellement en changer si besoin, mais tout d'abord quel est-il ?

Le module `os` est une bibliothèque qui permet d'interagir avec le système d'exploitation d'un ordinateur. La commande `import os` permet d'utiliser les commandes de ce module.

Par exemple pour connaître le répertoire dans lequel on se trouve on utilise l'instruction `os.getcwd()`. Il faut remarquer la notation pointée qui permet d'accéder à une fonction en particulier du module `os`, le nom de la fonction qui signifie *get current working directory* ainsi que les parenthèses qui permettent d'exécuter la fonction. Dans un script Python, pour afficher le résultat, il est nécessaire d'appeler la commande `print` de la manière suivante `print(os.getcwd())`. L'affichage est automatique dans le shell interactif.

E Bibliothèques logicielles

Tous les langages modernes possèdent des bibliothèques logicielles qui permettent d'accélérer le développement. Ce sont des collections de fonctions ou de classes réutilisables directement par le développeur. Il s'agit de gagner du temps et de ne pas réinventer la poudre. Elles permettent souvent d'adresser un domaine particulier de l'informatique (calcul, traitement des images, création de graphiques) ou de simplifier les interfaces (interagir avec un système d'ex-

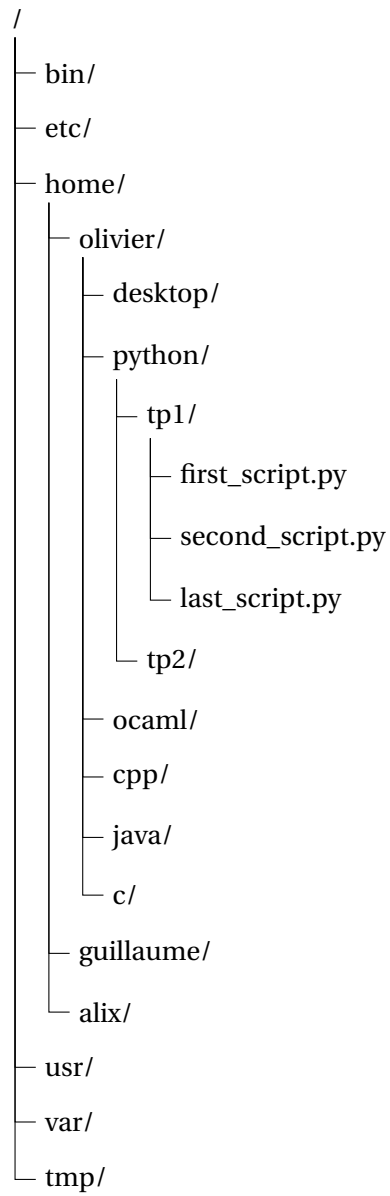

FIGURE 1.5 – Arborescence typique d'un système de fichier Unix/Linux/Mac OS X

ploitation ou un périphérique (image, son, réseaux)).

Comme de nombreux langages, Python possède un grand nombre de bibliothèques [1]. Les épreuves des concours se basent sur le cœur du langage et quelques bibliothèques. Parmi elles on peut citer :

- math,

1. On parle de l'Application Program Interface (API) Python

- random,
- numpy.

P Il faut bien faire à attention à **respecter les consignes des épreuves** de concours car ces épreuves peuvent être de nature légèrement différente selon les concours. Si le sujet précise qu'on ne doit pas utiliser numpy, alors il ne faut pas l'utiliser. Si le sujet précise qu'on doit utiliser une fonction d'une bibliothèque, alors il faut l'utiliser !

STOP Le paragraphe précédent est important pour l'épreuve d'informatique !

P En python, on peut choisir d'utiliser tout ou partie d'une bibliothèque et plusieurs syntaxes sont possibles. Il faut les connaître (cf. codes 1.2, 1.3 et 1.4) et **ne pas les mélanger** : au concours, ces points sont faciles à obtenir, alors faites attention !

STOP Le paragraphe précédent est important pour l'épreuve d'informatique !

Code 1.2 – Importer un module et l'utiliser

```
1 import math
2
3 a = math.sqrt(math.log(7))
```

Code 1.3 – Importer toutes les fonctions d'un module et en utiliser certaines

```
1 from math import *
2
3 a = sqrt(log(7))
```

Code 1.4 – Importer quelques fonctions d'un module et les utiliser

```
1 from math import sqrt, log
2
3 a = sqrt(log(7))
```

R D'un point de vue mémoire, la dernière syntaxe est plus parcimonieuse puisqu'elle n'importe en mémoire que les fonctions dont le code a besoin.