IFT-1004 - Introduction à la programmation

Module 1 : Présentation et mise en contexte

Honoré Hounwanou, ing.

Département d'informatique et de génie logiciel
Université Laval

Table des matières

Présentation de l’enseignant

Informations administratives

- Courriel: olufemi-honore-etsmoberg.hounwanou.1@ulaval.ca
- Bureau : PLT-3764 (bureau des chargés de cours)
- Disponibilités :
  - Sur rendez-vous
- **Professeur responsable : Simon Hardy Ph.D.**
- Remerciements à Jean-Francis Roy et Simon Hardy pour le matériel

Formation académique et Expérience professionnelle

- Étudiant au doctorat en sécurité informatique à l'Université Laval
- Formateur en ligne (LES TEACHERS DU NET)
- Auteur du livre "Premiers pas avec Python"

Présentation du plan de cours

Présentation du plan de cours

Plan de cours et site Web du cours

- https://sitescours.monportail.ulaval.ca/ena/site/informationsgenerales?idSite=86961

Le cours IFT-1004 est un cours de 3 crédits, avec une formule 3-2-4, ce qui signifie

- 3 heures de cours par semaine
- 2 heures de travaux dirigés par semaine
- 4 heures pour autres travaux et études par semaine
- Total : **9 heures par semaine, en moyenne**

Les travaux dirigés

À chaque semaine, vous aurez des exercices à faire en laboratoire. Ces exercices ne sont pas à remettre, et le corrigé sera publié sur le site Web du cours.

Les séances de laboratoire sont un **dépannage** : des auxiliaires sont là pour **présenter les exercices, répondre à vos questions**, et **présenter/commenter les solutions**.

Nous vous suggérons de faire les exercices des travaux dirigés en binôme. Un étudiant pilote(programme au clavier), l'autre étudiant est le copilote (observe et commente le travail). Les rôles sont échangés après 25-30 minutes. Cette formule offre de nombreux avantages pédagogiques.

Avec Skype ou Google Hangout, vous pouvez faire de même à distance.

Objectifs généraux du cours

- **Comprendre les concepts fondamentaux d'un langage de programmation**
- Pouvoir résoudre un problème par solution informatique (approche algorithmique)
- Savoir programmer en Python (attention, nous utiliserons **Python 3** et non Python 2)

Programme du cours

- Module 1 : Présentation du cours et mise en contexte
- Module 2 : Expressions, séquencement, conditions
- Module 3 : Fonctions, décomposition fonctionnelle en modules
- Module 4 : Types de données complexes
- Module 5 : Initiation à la modélisation
- Module 6 : Paradigme de l’orienté-objet
- Module 7 : Interfaces graphiques
- Module 8 : Modélisation et développement de grande envergure
- Module 9 : Gestion des erreurs et exceptions
- Module 10 : Récursivité

Examens

- Examen intra
  - Le dimanche 29 octobre 2017, de 9h00 à 11h50
  - 30% de la note finale
  - Document autorisé : une feuille **manuscrite, recto**
- Examen final
  - Le dimanche 17 décembre 2017, de 9h00 à 11h50
  - 35% de la note finale
  - Document autorisé : une feuille **manuscrite, recto-verso**

Travaux pratiques

- TP 1
  - Remise le 7 octobre 2017 à 23h50
  - 4% de la note finale
  - Individuel
- TP 2
  - Remise le 27 octobre 2017 à 23h50
  - 6% de la note finale
  - Individuel
- TP 3
  - Remise le 25 novembre 2017 à 23h50
  - 12% de la note finale
  - **En équipes de 2**
- TP 4
  - Remise le 15 décembre 2017 à 23h50
  - 12% de la note finale
  - **En équipes de 2**

Et le 1% manquant?

Le 1% manquant correspond à votre évaluation de l’enseignant !

Vous recevrez un courriel avec les informations pertinentes durant la session, et je ferai un rappel.

Consignes pour les travaux pratiques

- Les TPs 3 et 4 sont à réaliser **obligatoirement en équipe de 2**.
- Tout TP non remis dans les délais prévus se verra pénalisé suivant les modalités qui seront indiquées dans l'énoncé du TP (la date et l'heure d' échéance seront indiquées sur l'énoncé du TP).
- Toute révision de note devra être demandée au professeur dans un délai de 3 jours ouvrables après réception de la note. Note : une révision de note est demandée lorsque **l'on juge que la correction est erronée ou non équitable**. Pas lorsqu'on n'est pas satisfait de la note reçue !
- Les TPs doivent être impérativement soumis via le lien approprié du site Web du cours (aucune remise par courriel n'est acceptée).

Adresses utiles

Centre d'appui à la réussite étudiante (CARÉ)

- Questions relatives aux exercices, travaux dirigés et travaux pratiques
- Lundi : 14h30 à 16h30 au PLT-3966 et à distance sur Adobe Connect
- Mardi : 19h à 20h à distance sur Adobe Connect
- Mercredi : 19h à 20h à distance sur Adobe Connect
- Jeudi : 13h30 à 15h30 au PLT-3966 et à distance sur Adobe Connect
- http://www.ift.ulaval.ca/services/care-centre-dappui-a-la-reussite-etudiante/

Équipe LiberT

Problèmes avec votre ordinateur ? http://www.libert.fsg.ulaval.ca/

Dates importantes

Voir le calendrier universitaire pour les dates d’abandon avec/sans échec et avec/sans remboursement :

https://www.ulaval.ca/les-etudes/programmes-cours-horaire/calendriers-universitaires.html

Politiques importantes

- Politique sur les examens (en cas d’absence, d’un handicap…)
- Politique sur les travaux (en cas de plagiat…)
- Politique sur les cotes
- Politique sur l’utilisation d’appareils électroniques pendant une évaluation
- Politique sur le plagiat et la fraude académique

Important : Il est de votre responsabilité de prendre connaissance de ces politiques disponibles dans le plan de cours.

Plagiats

Tolérance zéro ! Attention à certains comportements :

- Vouloir aider un ami : ne vous faites pas prendre par les sentiments
- Trop expliquer : il y a des ressources pour cela
- Être trop enthousiaste de sa solution : éviter de trop l'exposer
- Être imprudent avec son ordinateur, clé USB, etc
- Il est indiqué dans le plan de cours que les communications entre équipes est interdite… Vous pouvez discuter de vos idées et de votre manière de résoudre un problème, mais **ne partagez jamais de code source**
- Un logiciel de détection de plagiat sera utilisé pour nous mettre la puce à l'oreille

Pour réussir le cours...

3 mots pour réussir le cours : **Organisation**, **assiduité** et **implication**

- Faites tout ce que vous avez à faire, le plus tôt possible !
- Assurez-vous de comprendre la matière importante pour les TPs, dès que possible !
- Donc débutez la conception de vos TPs le plus tôt possible pour identifier les éléments dont vous aurez besoin pour le réaliser.
- Surtout prenez le temps de faire les TDs, c’est une bonne préparation pour les TPs.

Ressources (1/2)

Lorsqu'il vous vient une question dans le cadre de ce cours, nous vous demandons de faire preuve de débrouillardise : la réponse est souvent disponible :

1. Dans les notes de cours (présentations) ;
2. Dans le livre de G. Swinnen ;
3. À l'aide de votre meilleur ami : Google ;
4. Sur le forum du cours.

Ressources (2/2)

Nous préconisons la communication entre vous (sans pour autant partager trop d'informations sur votre code, bien sûr), alors nous avons mis un forum disponible sur le site Web du cours.

Utilisez-le ! D'autres étudiants (ou moi-même) pourront répondre à vos questions.

Si vous ne trouvez toujours pas de réponse à votre question, en dernier recours seulement écrivez-moi un courriel.

Note: Les étudiants qui partageront du code entre eux seront accusés de plagiat.

Les courriels…

Il y a **400+** étudiants inscrits à ce cours (2 sections). Si chacun de vous m’écrit un tout petit courriel à chaque 2 ou 3 jours, imaginez !

Je vous demande donc de limiter vos courriels aux questions sur la matière, sur les aspects administratifs du cours, ou pour des éclaircissements sur les énoncés de travaux, **si vous n’avez pas trouvé de réponse sur le forum**.

Le courriel devrait être votre dernier recours.

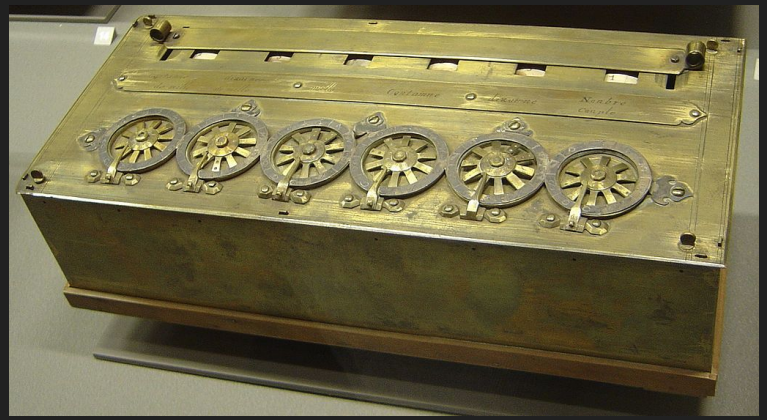
Si je reçois une question qui aurait pu être posée sur le forum ou dont la réponse est dans les notes de cours, il est fort probable que je vous redirige vers la bonne ressource.

Introduction à l’informatique

Objectifs

- Présenter certaines transformations importantes, en cours et à venir, de notre façon de vivre, résultant de l’informatique ;
- Montrer le rôle central que joue la programmation de ces transformations ;

Origines



Ordinateur = **grosse calculatrice**!

Les premières machines à calculer datent des années 1600. Blaise Pascal a réalisé en 1642 à l'âge de 19 ans la **Pascaline**, qui effectue les quatre opérations arithmétiques sans utiliser l’intelligence humaine.

Origines

Applications militaires : **La machine Enigma**, développée au début des années 1920, était une machine de chiffrement électromécanique à cylindres utilisée pour **coder des messages**, notamment pendant la 2e guerre mondiale par les Allemands. **Alan Turing** a participé à la création d'une machine qui, étant donné un message crypté, était capable de retrouver les réglages de la machine Enigma, pour ensuite décrypter les messages.

Applications en analyse d’informations

Traitement de grands volumes de données

- Les expérimentations du Large Hadron Collider représentent la lecture de **150 millions de senseurs**, **40 millions de fois par seconde**. Il y a 600 millions de collisions par seconde
- Décoder le génome humain prenait anciennement **10 ans de calculs** à réaliser. On le fait aujourd’hui en **quelques heures**
- Le NASA Center for Climate Simulation stocke **32 petaoctets** (10^{15}) d’observations climatiques
- Facebook gère **50 milliard** de photos d’utilisateurs
- Google traite **100 heures** de nouvelles vidéos à **chaque minute**

Optimisation

- Horaires (de travail, de cours, de ligues professionnelles, ...)
- Parcours (aérien, de livraison, ...)
- Réseau électrique

Informatique et robotique

Ordinateur + capacité motrices (+ perception)

- Historique
  - Robotique militaire
  - Industrialisation
- Quincaillerie additionnelle
  - Appendices (bras, jambes)
  - Senseurs (yeux, oreilles, capteurs divers)

Informatique et robotique

Ordinateur + capacité motrices (+ perception)

- Systèmes embarqués
  - Domotique
  - GPS
  - Téléphones intelligents (senseurs, capteurs, GPS, etc.)
- Robots
  - Domotique
  - Drones
  - Démineurs

Intelligence artificielle

- Reconnaissance visuelle (objets, personnes, scènes)
- Reconnaissance auditive (son, musique, langage)
- Prédiction de structures biologiques (conception de médicaments)
- Raisonnement de haut niveau (Deep Blue, Watson)
- Systèmes multi-agents

Le Web

- Partage de données et d’information
- Distribution de calculs sur des milliers d’ordinateurs distants
- Réseaux sociaux
- E-commerce

Liens intéressants

- https://www.youtube.com/watch?v=in-2VHDv44Q
- https://www.youtube.com/watch?v=XeEYaX82jSE
- https://www.youtube.com/watch?v=vt79JcPfZQA
- https://www.youtube.com/watch?v=OXgoueCSDpI
- https://www.jitbit.com/alexblog/249-now-thats-what-i-call-a-hacker/
- https://www.youtube.com/watch?v=-WooXoaSiuk
- https://www.ted.com/talks/bettina_warburg_how_the_blockchain_will_radically_transform_the_economy

Pensez-y…

Sortez de votre poche votre téléphone intelligent, et ouvrez Facebook. Que se passe-t-il ?
À partir d'une **interface tactile conviviale** sur un **système d'exploitation embarqué** où vous et moi pouvons développer et partager des **applications mobiles**, votre téléphone se connecte via un **réseau d'envergure planétaire** sur un **serveur distant** partageant la tâche d'**agréger et combiner des données** (texte, photo, vidéo) partagées par des millions d'utilisateurs à travers le monde, tout en **apprenant vos habitudes** à partir de vos actions pour vous afficher des **publicités ciblées**.

Introduction à la programmation

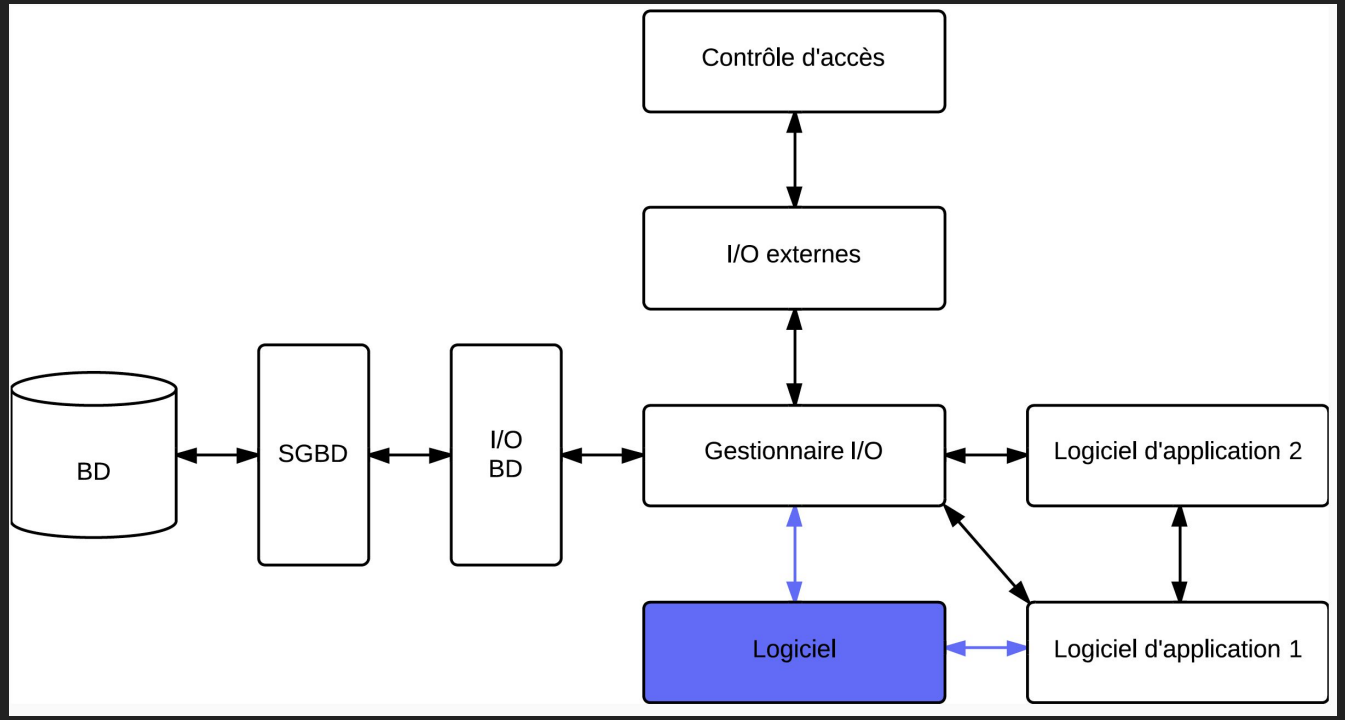
Objectifs

- Situer l’étape de programmation dans le développement logiciel ;
- Initier l’étudiant aux concepts d’entrée-sortie, de variable, d’affectation et d’opérations.
- Présenter les principales composantes logiques d’un ordinateur ;

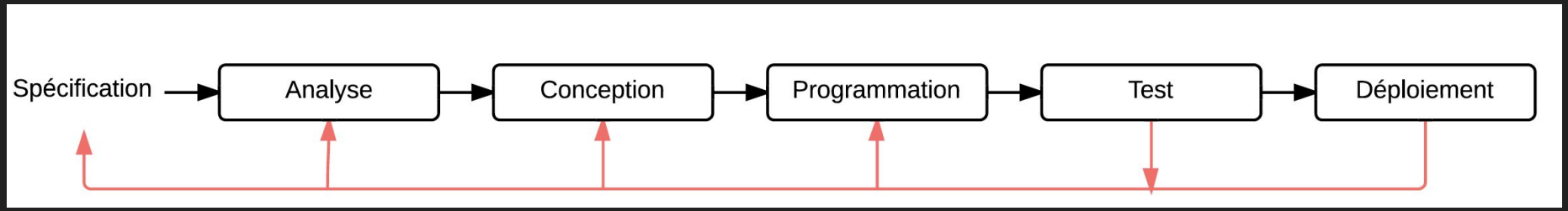
L’architecture logicielle

- Un système n’est jamais « stand alone » ; il fait partie d’un tout.
- L’architecture définit :
  - Les logiciels utilisés par l’organisation
  - Leurs interconnexions
  - Les procédures de sécurité et de recouvrement
  - Les standards et normes de représentation, de documentation et de développement
- Tout nouveau logiciel à développer doit s’intégrer à l’ensemble.

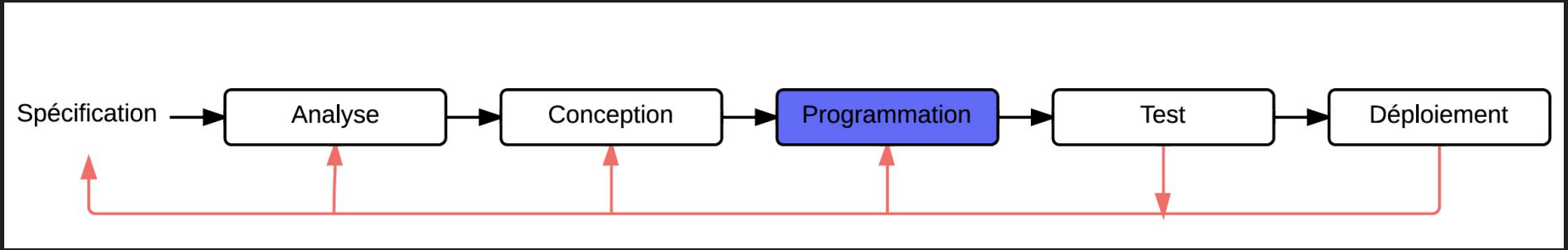
Un exemple


Contrôle d'accès
I/O externes
BD
SGBD
I/O
BD
Gestionnaire I/O
Logiciel d'application 2
Logiciel
Logiciel d'application 1

Cycle de développement d'un logiciel



Cycle de développement d’un logiciel



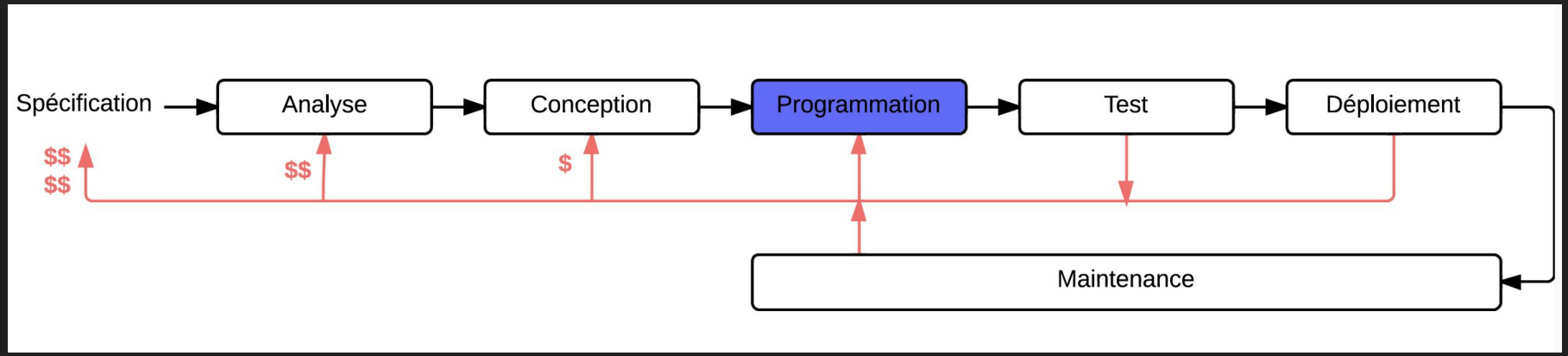
Pour les informaticiens, d'autres cours du bac vont cibler les autres étapes. On commence par la programmation car c'est plus fun! :-)

Objectifs secondaires de votre formation

Lors du développement d'un logiciel, il faut se soucier :

- **Du temps de développement**;
- **Des coûts de développement**;
- De son autonomie et son efficacité ;
- De sa facilité d'interconnexion avec son environnement et de sa facilité d'utilisation (convivialité) ;
- De sa portabilité et de sa compatibilité (ascendante et descendante) ;
- **De sa validité, de sa fiabilité et de sa vérifiabilité**;
- **De sa facilité d'entretien (maintenabilité) et de sa longévité**;

Cycle de développement d’un logiciel



Connaissance déclarative et impérative

Dans la vie réelle, il existe principalement deux types de connaissances.

Nous avons la connaissance dite déclarative et celle dite impérative.

En fonction du problème que nous souhaitons résoudre, il va généralement falloir nous assurer d’avoir ces deux connaissances.

Connaissance déclarative

La connaissance déclarative est composée d'énonciations à priori vraies. Par exemple:

- Il faut bien manger pour être en bonne santé
- Quand le feu est rouge, les voitures doivent s'arrêter et laisser les piétons passer
- **y est une racine carrée de x si et seulement si y*y = x**

Connaissance impérative

La connaissance impérative quant à elle fait référence à un procédé permettant d'accomplir une tâche. C'est un peu comme une recette.

Approximation de la racine carrée d'un nombre **x** par la méthode babylonienne:

1. **Faire un premier choix**: Choisir n'importe quel nombre positif n_0
2. Si $n_0 * n_0$ est assez proche de x alors n_0 est une bonne approximation de la racine carrée de x
3. **Amélioration du choix**: Dans le cas contraire, améliorer le choix en faisant la moyenne de n_0 et x/n_0. C'est-à-dire: $\mathbf{n_{nouveau} = (n_{ancien} + x/n_{ancien}) / 2}$
4. **Itération jusqu'à une meilleure approximation**: Utiliser ce nouveau choix $n_{nouveau}$ et retourner à l'étape 2

Impl. en Python de la méthode babylonienne

```
from random import randint

def babylonian_algorithm(n):
    assert n >= 0

    if n == 0: return 0

    epsilon = 0.001 # Précision de 3 décimales

    guess = randint(1, n)
    while(abs(n - guess * guess) > epsilon):
        guess = (guess + n/guess) / 2

    return guess
```

Petite amélioration

```
from random import randint

def babylonian_algorithm(n):
    assert n >= 0

    if n == 0: return 0

    epsilon = 0.001 # Précision de 3 décimales

    guess = randint(1, n//2)
    while(abs(n - guess * guess) > epsilon):
        guess = (guess + n/guess) / 2

    return guess
```

Note: D’autres améliorations restent toutefois possibles.

Une seconde méthode pour obtenir la racine carrée

Implémentation en Python

```
def square_root(n):
    return n ** 0.5
```

Explication:

1. $n^a * n^b = n^{a+b}$
2. y est une racine carrée de x si et seulement si y*y = x
3. $n^{1/2} * n^{1/2} = n^{1/2 + 1/2} = n^{2/2} = n^1 = n$
4. donc $n^{1/2}$ est une racine carrée de n.

En programmation, comme vous le verrez, il existe très souvent **plusieurs solutions** à un problème donné.

Définitions

- Un algorithme est une recette (procédé) permettant de résoudre un problème donné
- Un programme est un ensemble d'instructions exécutées par l'ordinateur. C'est en effet la traduction de votre algorithme dans un langage de programmation
- Une instruction est une tâche que l'ordinateur doit exécuter

Notion de langage de programmation

Rappelons qu'un programme est juste une séquence d'instructions indiquant à l'ordinateur ce qu'il doit faire. Évidemment, nous avons besoin de fournir ces instructions dans un langage que l'ordinateur peut comprendre. Il aurait été très intéressant si nous pouvions juste dire à un ordinateur ce qu'il doit faire dans notre langue maternelle, comme dans les films de science-fiction. Malheureusement, en dépit de plusieurs efforts des informaticiens, créer des ordinateurs capables de comprendre le langage humain est un problème encore non résolu.

Même si les ordinateurs pouvaient nous comprendre, le langage humain n'est vraiment pas commode pour décrire des algorithmes assez complexes car il est rempli d'ambiguïté et d'imperfection.

Notion d’ambiguïté (1/2)

“J’ai vu un homme dans le parc avec des jumelles.”

- Est-ce que cela veut dire que j’avais des jumelles dans les mains?
- Ou plutôt c’était l’homme que j’ai vu dans le parc qui avait des jumelles?
- Et de plus qui est-ce qui était dans le parc?

Notion d’ambiguïté (2/2)

“Le salaire de Nathalie devra lui être envoyé de façon bihebdomadaire.”

- Aujourd’hui, l’adjectif bihebdomadaire signifie « qui a lieu ou parait deux fois par semaine »
- Autrefois, il avait plutôt le sens de « toutes les deux semaines »

Définition

Les informaticiens ont donc essayé de résoudre ce problème en créant des notations spéciales pour exprimer nos intentions de manière précise et non ambiguë. L'ensemble de ces notations spéciales forme ce qu'on appelle **un langage de programmation**.

Qu’est-ce que Python?

Python est un langage de programmation et c’est ce dernier que nous allons découvrir tout au long de ce cours.

Vous avez peut être déjà entendu parler d’autres langages de programmation comme C#, Ruby, PHP, JavaScript, Java... Bien que ces langages diffèrent de beaucoup de détails, ils partagent la propriété d’avoir une syntaxe bien définie et non ambiguë.

Tous les langages cités plus haut sont des exemples de langages de haut niveau. En fait, plus le langage de programmation se rapproche du langage humain, plus on dit qu’il est de haut niveau (mais évidemment la traduction en binaire prendra un peu plus de temps :))

https://en.wikipedia.org/wiki/List_of_programming_languages_by_type

Binaire ?

L’ordinateur ne comprend en réalité qu’un seul langage qu’on appelle communément **langage machine** ou **langage binaire**. Ce langage n’est composé que de deux chiffres 0 et 1 appelés aussi bits (élément binaire).

En d’autres termes, l’ordinateur ne comprend que des 0 et 1.

Comme vous pouvez l’imaginer, écrire un programme avec des 0 et 1 est un vrai parcours de combattant. Ainsi grâce aux langages de programmation qui se rapprochent un peu plus du langage naturel (anglais, français ...), nous pourrons écrire nos programmes (on dit aussi coder) plus aisément. Ce code sera ensuite traduit en langage machine afin que l’ordinateur puisse le comprendre.

Alors comment est-ce que notre code est traduit en langage machine ?

Compilation et Interprétation

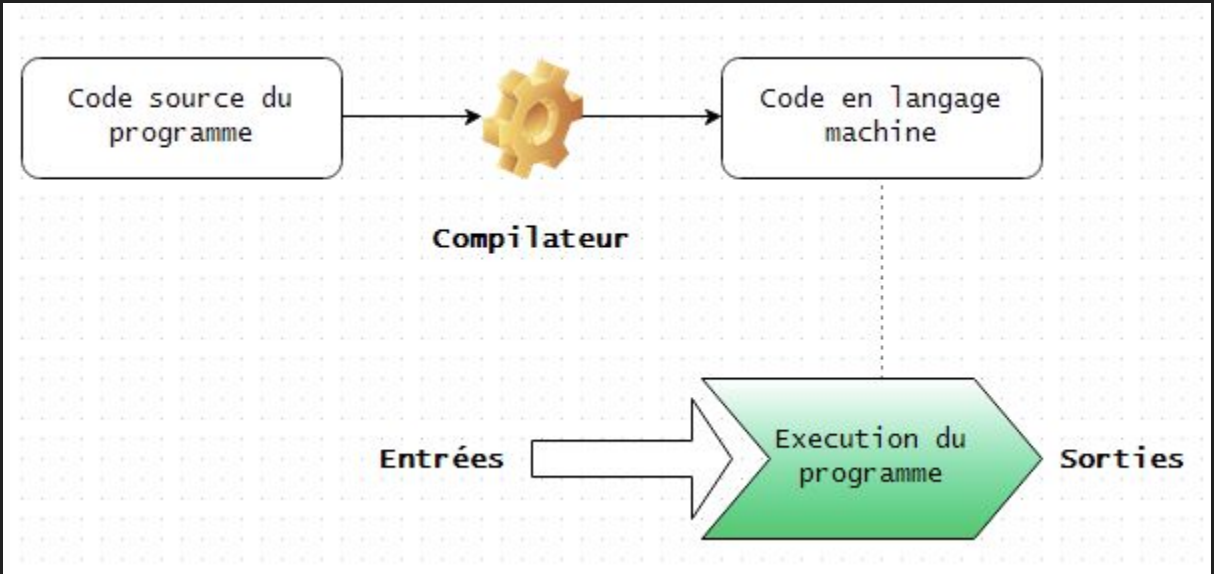
Il existe de manière générale deux moyens principaux pour traduire un code écrit dans un langage de programmation en langage machine: la compilation et l’interprétation.

Compilation

Un compilateur est un programme complexe qui prend en entrée un programme écrit dans un langage de programmation (en C par exemple) et le traduit en un programme équivalent en langage machine.

Le programme écrit dans un langage de programmation est appelé **code source** et le résultat obtenu après traduction en code machine est un programme que l'ordinateur peut directement exécuter.

Processus de compilation d'un programme



https://leanpub.com/premiers-pas-avec-python/read

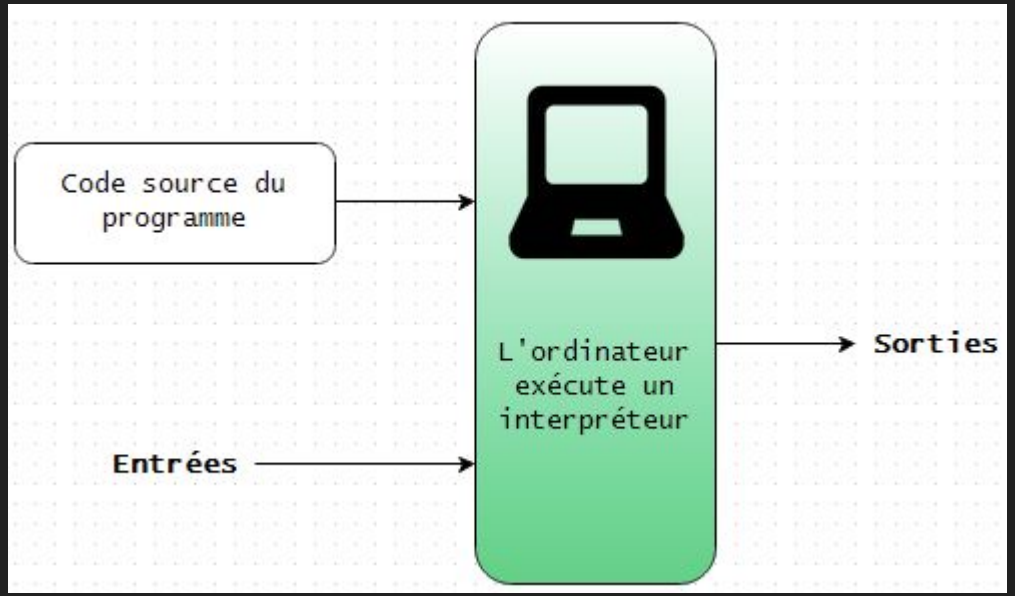
Interpréteur

Un interpréteur est un programme qui simule un ordinateur qui comprend un langage de haut-niveau. Plutôt que de traduire le code source du programme en langage machine, l'interpréteur analyse et exécute le code source instruction après instruction.

Python est un exemple de langage interprété. Ce qui revient à dire que nous aurons besoin d'un interpréteur Python afin de pouvoir exécuter nos futurs programmes.

Note: Lors du premier laboratoire, vous découvrirez comment installer un interpréteur Python au travers de l'outil tout-en-un Anaconda.

Processus d’interprétation d’un programme



https://leanpub.com/premiers-pas-avec-python/read

Avantages et Inconvénients (1/2)

La différence entre l'interprétation et la compilation est que la compilation est un peu plus courte lors de la traduction en langage machine. Une fois le programme compilé et que tout fonctionne, il peut être exécuté plusieurs fois sans avoir besoin de compiler à nouveau le code source. Alors que dans le cas de l'interpréteur, il a besoin d'être lancé à chaque fois qu'on exécute le programme vu qu'il devra de nouveau analyser et exécuter le code source instruction après instruction.

Les programmes compilés ont donc tendance à être plus rapide à l'exécution, vu que la traduction est faite une fois pour toute, mais les langages interprétés permettent une plus grande flexibilité pour l'environnement de programmation car les programmes peuvent être développés et exécutés de manière interactive.

Avantages et Inconvénients (2/2)

Le processus d'interprétation fait ressortir un autre avantage, c'est que les programmes écrits pourront être généralement exécutés sur plusieurs plateformes (plusieurs systèmes d'exploitation différents): on parle de **portabilité**.

Pour résumer, sachez qu'un programme écrit dans un langage de programmation pourra être exécuté sur plusieurs sortes d'ordinateurs tant qu'il y a un compilateur et/ou un interpréteur approprié.

Bon à savoir!

Bien que l'interprétation et la compilation soient les deux principaux moyens par lesquels la traduction du code est faite, ils ne s'excluent pas mutuellement.

Les désignations "langage interprété" ou "langage compilé" sont en pratique uniquement fondées sur l’implémentation la plus commune du langage, plutôt que de représenter une propriété essentielle d'un langage.

Exécution d’un programme Python

“Code source” en Python

```
print(‘hello, world’)
```

1. Compilation

“Bytecode” équivalent

```
1           0 LOAD_NAME                0 (print)
            2 LOAD_CONST               0 ('hello, world')
            4 CALL_FUNCTION            1
            6 RETURN_VALUE
```


2. Interprétation/Exécution

L’interpréteur Python traduit le code source en une représentation intermédiaire efficiente (bytecode) et l'exécute immédiatement.

Naissance du langage Python

Python est un langage de programmation créé en 1991 par le développeur néerlandais **Guido van Rossum**.

Guido a décidé de baptiser son projet Python en référence à la série télévisée des Monty Python dont il en est un grand fan.

https://fr.wikipedia.org/wiki/Guido_van_Rossum

Caractéristiques du langage Python

- Langage interprété
- Facile d’apprentissage
- Portable
- Typage dynamique
- Interactif
- Librairie standard très complète, grande quantité de librairies third-party
- Langage orienté objet
- Excellente documentation
- **Syntaxe élégante et concise**

D’accord, mais est-ce utilisé dans l’industrie ?

Oui ! Mais même s’il ne l’était pas, ce ne serait pas un problème : vous apprenez à **programmer**, le langage est un outil pour y arriver.

Quelles entreprises utilisent Python ?

- Google
- NASA
- Mozilla
- Instagram
- Eventbrite
- etc.

Et en éducation?

Quelles universités enseignent Python comme premier langage de programmation?

- L’université Laval
- Le MIT
- UC Berkeley
- Georgia Tech
- etc.

https://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-u-s-universities/fulltext

Et on peut faire quoi avec ?

- Analyser des données (pandas, matplotlib)
- Faire du calcul scientifique (numpy, scipy)
- Faire de l’apprentissage automatique (scikit-learn)
- Construire des sites Web (flask, cherrypy)
- Maintenir des serveurs
- Programmer des jeux vidéos (pygame)
- Faire de la robotique avec un Raspberry Pi
- Et beaucoup plus !

https://www.python.org/about/success/

Environnement de développement intégré (EDI)

Pour écrire votre code source, vous aurez besoin d'un éditeur de texte.

En théorie, un logiciel comme Bloc-notes sous Windows, ou encore TextEdit sous macOs fera parfaitement l'affaire, mais vous verrez que dans la pratique, l'on a tendance à utiliser ce qu'on appelle des environnements de développement intégrés (EDI) qui ont l'avantage d'avoir en plus de l'éditeur de texte (généralement avec une excellente coloration syntaxique), des outils comme un interpréteur/compilateur, un débogueur (pour faciliter la correction des erreurs), etc, qui nous faciliteront la tâche lors du développement de nos programmes.

Environnement de développement intégré (EDI)

Lors du premier laboratoire, vous découvrirez comment installer un environnement de développement intégré Python appelé PyCharm.

Étant donné que vous débutez, nous avons eu à effectuer ce choix afin que vous n'ayez pas à décider de quoi que ce soit. Sachez toutefois que dans la pratique, choisir un EDI, c'est un peu comme choisir votre stylo/cahier; les goûts et les couleurs ne se discutent pas :).

“hello, world” en C et Python

Implémentation en C

```c
#include <stdio.h>

int main(void) {

    printf(“hello, world\n”);

    return 0;

}
```

Implémentation en Python

```python
print(“hello, world”)
```

Formule de Leibniz pour π

La formule de Leibniz est un exemple de série alternée :

$$\sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1} = \frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots = \frac{\pi}{4}$$

Note: La formule de Leibniz converge extrêmement lentement. Par exemple, elle nécessite environ cinq milliards de termes pour calculer une valeur de π à 10 décimales correctes. Cependant, elle peut être utilisée pour calculer π avec une plus grande précision en utilisant diverses techniques d'accélération de convergence. Nous l'utilisons ici pour la simplicité de son implémentation.

https://en.wikipedia.org/wiki/Leibniz_formula_for_%CF%80

Impl. de la formule de Leibniz pour π en C et Python

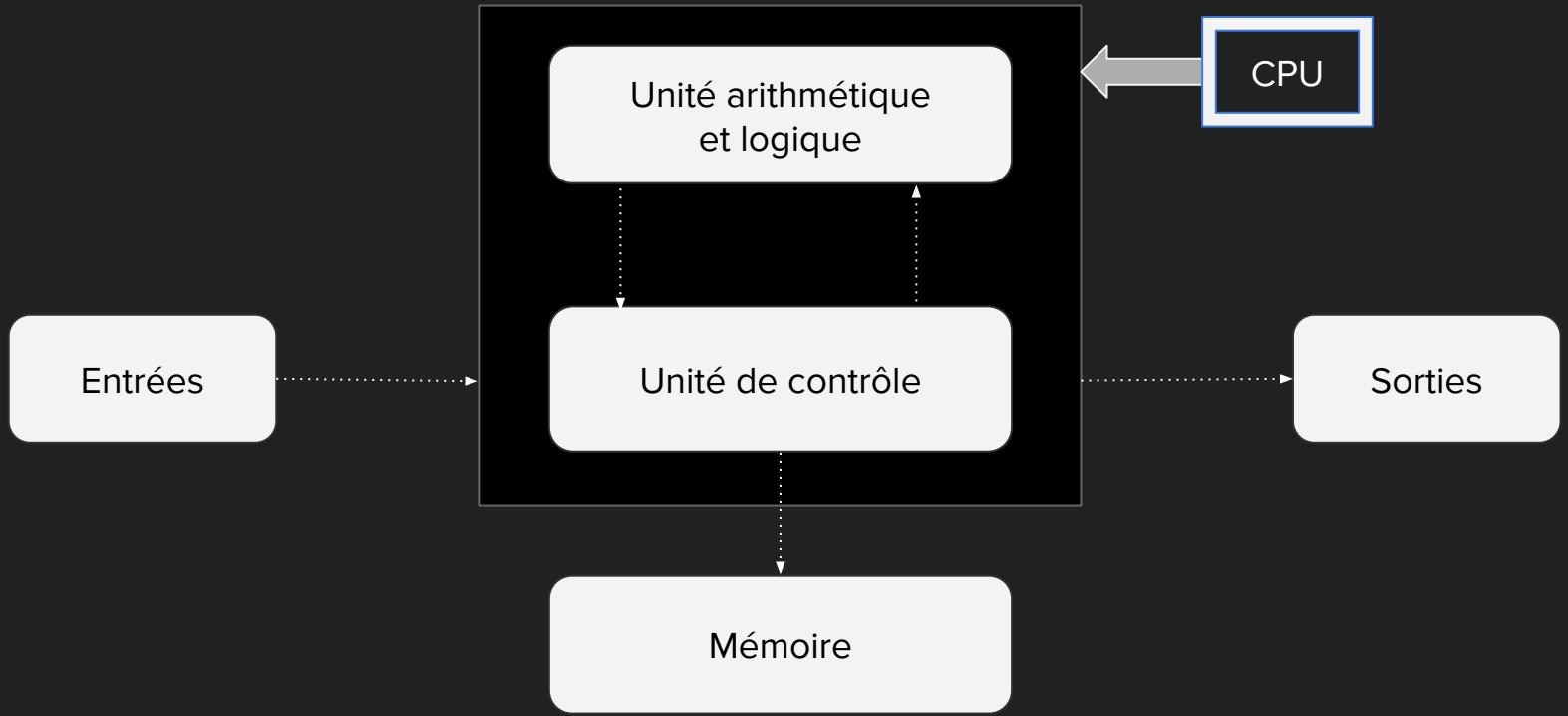
Implémentation en C

```c
#include <stdio.h>
#include <math.h>

int main(void) {
    int k;
    double result = 0.0;

    for(k = 0; k < 1000000; k++) {
        result += pow(-1, k) / (2*k + 1);
    }

    result *= 4;

    printf("pi = %.16f\n", result);

    return 0;
}
```

Implémentation en Python

```python
print("pi =", 4 * sum(pow(-1, k) / (2*k + 1) for k in range(1_000_000)))
```

Unités logiques d’un ordinateur (1/2)



http://tinyurl.com/ycf5h9d8

Unités logiques d’un ordinateur (2/2)

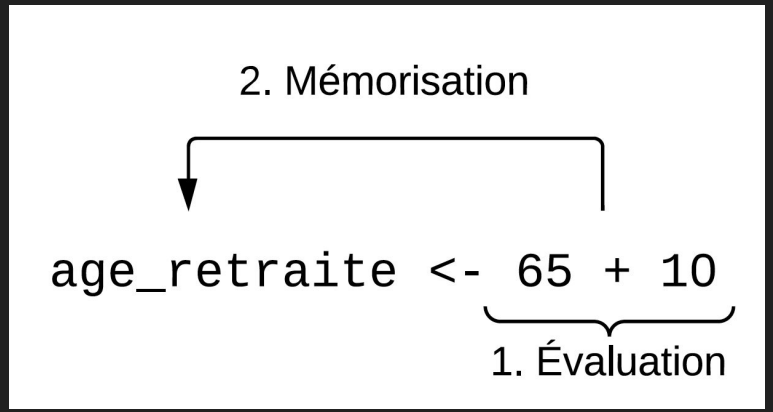
- Entrées (clavier, microphone, scanneur)
- Sorties (écran, imprimante, haut-parleur)
- Unité arithmétique et logique (effectue les différents calculs)
- CPU (chef d’orchestre)
- Mémoire RAM (volatile), perd son contenu lorsque l’ordinateur n’est plus alimenté en électricité.
- Mémoire du disque dur (non volatile), conserve son contenu même après extinction de l’ordinateur.

Opérations de gestion d’information

- **L’affectation** : pour déplacer de l’information
- **Les variables** : pour mémoriser de l’information
- **L’entrée (input)** : pour acquérir des données provenant de l’extérieur
- **La sortie (output)** : pour communiquer des données vers l’extérieur

Opération d’affectation

- Évalue l’expression à droite du symbole <-
- Mémorise le résultat de l’évaluation de l’expression dans la variable identifiée à gauche du symbole <-



Variables

variable <- ... : mémoriser une information

1. Trouver ou créer la variable nommée à gauche de <-
2. Évaluer la valeur de l'expression à droite de <-
3. Mettre cette valeur dans la variable nommée à gauche de <-
4. Mémoriser le type de cette valeur comme type de la variable

Variables

age_retraite <- 65

Nom	Adresse	Contenu	Type
...	1000	...	...
...	1001	...	...
...	1002	...	...
...	1003	...	...
...	1004	...	...

Variables

age_retraite <- 65

Nom	Adresse	Contenu	Type
...	1000	...	...
...	1001	...	...
age_retraite	1002	...	...
...	1003	...	...
...	1004	...	...

Variables

age_retraite <- 65

Nom	Adresse	Contenu	Type
...	1000	...	...
...	1001	...	...
age_retraite	1002	**65**	...
...	1003	...	...
...	1004	...	...

Variables

age_retraite <- 65

Nom	Adresse	Contenu	Type
...	1000	...	...
...	1001	...	...
age_retraite	1002	65	**entier**
...	1003	...	...
...	1004	...	...

Variables

age_embauche <- 60.0

Nom	Adresse	Contenu	Type
...	1000	...	...
...	1001	...	...
age_retraite	1002	65	entier
...	1003	...	...
...	1004	...	...

Variables

age_embauche <- 60.0

Nom	Adresse	Contenu	Type
...	1000	...	...
...	1001	...	...
age_retraite	1002	65	entier
age_embauche	1003	...	...
...	1004	...	...

Variables

age_embauche <- 60.0

Nom	Adresse	Contenu	Type
...	1000	...	...
...	1001	...	...
age_retraite	1002	65	entier
age_embauche	1003	**60.0**	...
...	1004	...	...

Variables

age_embauche <- 60.0

Nom	Adresse	Contenu	Type
...	1000	...	...
...	1001	...	...
age_retraite	1002	65	entier
age_embauche	1003	60.0	**réel**
...	1004	...	...

Variables

age_retraite <- 65.0

Nom	Adresse	Contenu	Type
...	1000	...	...
...	1001	...	...
age_retraite	1002	65	entier
age_embauche	1003	60.0	réel
...	1004	...	...

Variables

age_retraite <- 65.0

Nom	Adresse	Contenu	Type
...	1000	...	...
...	1001	...	...
age_retraite	1002	**65.0**	entier
age_embauche	1003	60.0	réel
...	1004	...	...

Variables

age_retraite <- 65.0

Nom	Adresse	Contenu	Type
...	1000	...	...
...	1001	...	...
age_retraite	1002	65.0	**réel**
age_embauche	1003	60.0	réel
...	1004	...	...

Définition

Une variable :

- Représente une donnée ;
- Porte un nom, idéalement significatif ;
- Est associée à une case de la mémoire (pour y stocker la donnée) ;
- A donc une **adresse mémoire**;
- Est du type de la dernière donnée qu'on lui a affectée ;
- Ce type est nécessaire pour interpréter le contenu de la variable.

En résumé…

- Un bon programmeur doit s’assurer de toujours avoir les connaissances déclarative et impérative associées au programme qu’il souhaite concevoir
- La programmation est à la base de toute application informatique
- Un langage de programmation est un ensemble de notations spéciales nous permettant de dialoguer avec l’ordinateur de manière non ambiguë.
- Il existe une grande variété de langages de programmations mais plusieurs notions fondamentales sont communes
- **Apprendre à programmer nécessite beaucoup de pratique et de patience**
- La programmation est un *outil* important pour tout scientifique

Lectures, travaux, exercices

Plan des activités du module 1

- Documents à lire
  - Chapitres 1 et 2 de G. Swinnen
- Travaux dirigés
  - Installation, survol et test des logiciels utiles
  - Premiers pas avec l’interpréteur, écriture d’un premier programme, consultation de l’aide interactive

Lexique

CPU	Central Processing Unit (Unité centrale de traitement)
EDI	Environnement de Développement Intégré (IDE en anglais)
Impl.	Implémentation

Questions?