

UAA11 Introduction à la programmation procédurale en Python

1 Bibliographie

1.1 Cours et formations

1. **[1-AppPy] Apprendre à programmer en python 3** - pdf de Gérard Swinnen servira de manuel de cours et de complément aux exercices.

N'hésite pas à le consulter si tu veux comprendre une notion.

Les exercices notés [1-AppPy] sont tirés de ce manuel.

Informatique au lycée NSI Première (France) - cours en ligne

<https://pixees.fr/informatiquelycee>

2. **Apprentissage de python au Lycée (site français)**

Les mémos de cours et les exercices d'apprentissage sont tirés de ce site.

<https://cours-nsi.forge.apps.education.fr/premiere/>

3. Openclassroom Apprenez à programmer en python - pdf ou cours en ligne clair.

4. OpenClassroom Démarrez votre projet avec python

<https://openclassrooms.com/fr/courses/4262331-demarrez-votre-projet-avec-python>

5. Tutoriel en ligne

<https://docs.python.org/fr/3/tutorial>

6. Zeste de savoir - cours python, sql, traitement d'images

https://zestedesavoir.com/rechercher/?q=python&models=content&from_library=on&category=informatique&subcategory=programmation-et-algorithmique

7. Apprendre à programmer en python

<https://www.pierre-giraud.com/python-apprendre-programmer-cours>

8. Learn python programming

<https://pythonbasics.org>

9. Lycée des Flandres, cours de python en 1^{ère} NSI (France)

https://qkzk.xyz/docs/nsi/cours_premiere/programmation

10. Programmer en python 3 d'Antoine Pernot (en Français)

<https://python.antoinepernot.fr/index.php>

11. Liste de ressources sur Python, notamment les documentations officielles

<https://lyc-84-bollene.gitlab.io/chambon/1-Python/3-ressources>

12. Python3 - objectif jeux

<https://www.apprendre-en-ligne.net/pj>

13. Python au lycée (math) - site exo7

<http://exo7.emath.fr>

14. Python au lycée - vidéos

<https://www.youtube.com/Pythonaulyc%C3%A9e>

15. Apprendre python - vidéos graven

<https://www.youtube.com/watch?v=psaDHhZ0cPs&list=PLMS9Cy4Enq5JmIZtKE5OHJCI3jZfpASbR&index=2>

1.2 Formations en ligne autonomes

20. **[FceIO] Olympiade française d'informatique**

<http://www.france-ioi.org/algo/chapters.php>

Les exercices sont classés selon les types définis par Fce-ioi :


- ✓ Lis les commentaires de la rubrique **Cours** et démarre par les exercices de **Découverte**
- ✓ **Validation** : te permet de valider le chapitre en cours et d'accéder au suivant.
- ✓ **Entraînement** : ces exercices supplémentaires sont de même type que les « validation » et te permettent de t'entraîner, à faire si besoin.
- ✓ **Challenge** : ces exercices de programmation sont plus difficiles et te permettent d'aller plus loin ! Pour de la remédiation, tu peux les sauter.

Quand tu abordes les boucles et compteurs, je te conseille d'exécuter ton code dans un éditeur python sur pc ou sur internet, la visualisation du résultat de ton code dans FranceIOI n'est pas claire.

21. Tutoriel en ligne future coder + outils de debug (snoop, Python Tutor, birdseye) <https://fr.futurecoder.io/>

Le site est récent, très bien fait, les explications sont claires.

Par contre, les exercices d'entraînement sont peu nombreux et vont peut-être devenir vite difficile.

- ✓ En cas de difficulté, tu peux débbugger ton code en l'exécutant pas à pas avec :

- ✓ Une fois connecté, dans les réglages, tu peux activer le mode « développeur » qui permet de passer une étape si tu es bloqué.
- ✓ Attention, tes exercices ne sont pas sauvegardés, mieux vaut travailler avec un fichier ouvert sur ton pc et faire des copier/coller des exercices résolus au fur et à mesure.

22. Tutoriel en ligne w3school <https://www.w3schools.com/python/default.asp>

23. Tutoriel en ligne learnpython <https://www.learnpython.org>

24. Tutoriel en ligne avec outil de debug <https://courspython.com>

1.3 Algorithmique

30. « Algorithmique et programmation pour non matheux » de C. Darmangeat
pise.info/algo/index.htm
31. « Informatique - Chap 9 Algorithmique »
www.nymphomath.ch
32. « Algorithmique pour le lycée » d'E. Sopena
33. « Algorithmique pour l'apprenti programmeur »
zestedesavoir.com et openclassrooms.com
34. « Algorithmique au lycée »
www.maths-et-tiques.fr

1.4 Outils

35. Editeur sur PC **Visual Studio Code** (sous Windows et linux)

<https://code.visualstudio.com/download>

36. Editeur sur PC **idle3** (sous Windows et linux)

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

37. Editeurs de code et terminal en ligne basthon

basthon.fr

38. Editeurs de code en ligne

<https://www.onlinegdb.com>

39. Cours + Editeurs future coder + outils de debug (snoop, Python Tutor, birdseye)

<https://fr.futurecoder.io>

40. Outil de debug **Python Tutor** (execution pas à pas)

<https://pythontutor.com/visualize.html#mode=edit>

1.5 Installation et configuration de Visual Studio Code

Installation sous linux

```
apt-get install wget gpg
wget -qO- https://packages.microsoft.com/keys/microsoft.asc | gpg --dearmor > packages.microsoft.gpg
install -o root -g root -m 644 packages.microsoft.gpg /etc/apt/trusted.gpg.d/
sh -c 'echo "deb [arch=amd64,arm64,armhf signed-by=/etc/apt/trusted.gpg.d/packages.microsoft.gpg]
https://packages.microsoft.com/repos/code stable main" > /etc/apt/sources.list.d/vscode.list'
rm -f packages.microsoft.gpg
apt install apt-transport-https
apt update
apt install code
```

Installation sous windows

L'exécutable d'installation est disponibles sous :

<https://code.visualstudio.com/download>

Configuration

- ✓ Crée un fichier `essai2.py`
- ✓ Quand tu click dans le code, accepte l'installation des extensions et écris :

```
print("Hello ...")
print("... en python")
for loop in range(5):
    print("Comment vas-tu ?")
print("T'es sûr que ça va ?")
```

- ✓ Exécute ton code et observe dans le terminal du bas:

Run -> Run without debugging


- ✓ Pointe avec la souris sur le numéro de la ligne 2 pour rajouter un point d'arrêt

-> clique droit

-> **add a breakpoint**

- ✓ Exécute ton code en mode debug

run -> start debugging ou F5 ou 

- ✓ Le code va s'arrêter sur la ligne 2, joue avec les commandes  pour avancer pas à pas dans la boucle et visualiser le contenu de la variable sur la gauche.
- ✓ Pour ouvrir un **terminal python** (dans lequel tu peux taper des instructions python) :

-> View -> Command palette -> python : Start REPL

2 Quelques notions sur le fonctionnement d'un ordinateur

2.1 Introduction

Pour la machine, l'information est composée uniquement de données, il est donc intéressant de comprendre :

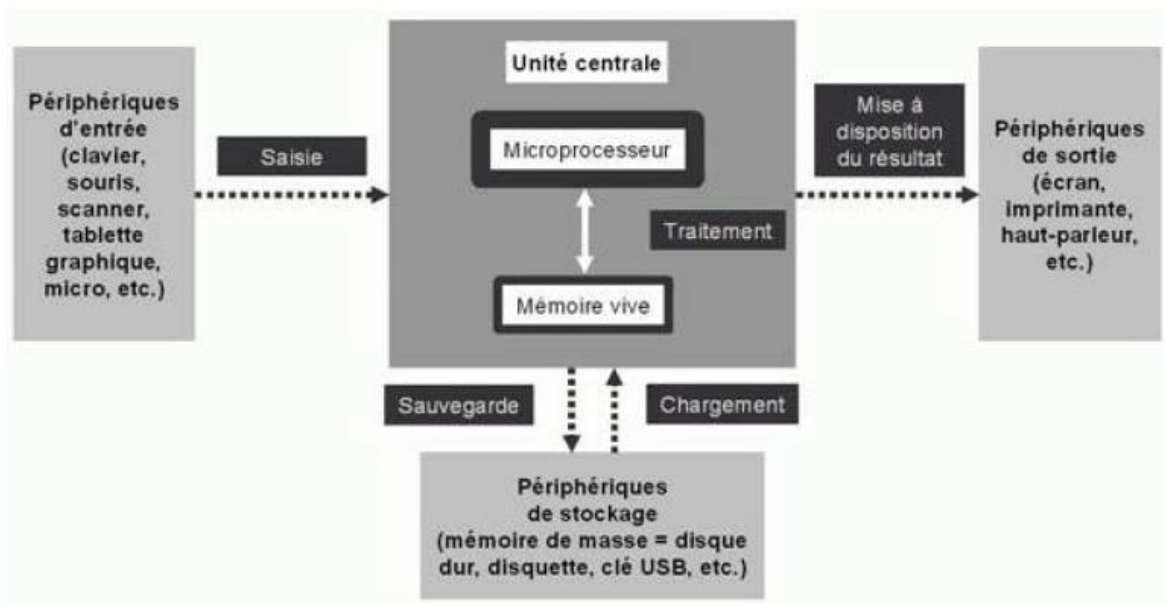
- comment l'ordinateur va traiter ces données
- comment sont représentées ces données dans la machine (sous forme de code)
- comment ces données peuvent être converties pour plus compréhensible et faciles à exploiter par l'Homme.

2.2 Rappels de définitions

Un **ordinateur** est dispositif capable de traiter de l'information.

L'**informatique** est la science du traitement automatique et rationnel de l'information.

2.3 Fonctions de base d'un ordinateur



Un ordinateur a pour fonction :

- ✓ De communiquer avec l'utilisateur en :
 - en gérant les informations en : les
 - en communiquant les générées par le traitement : les
- ✓ D'effectuer des (ou) en générant des résultats à partir des données. Ces calculs sont effectués par le processeur.
- ✓ De les opérations élémentaires c'est dire de les exécuter l'une après l'autre selon un certain plan.

Ce plan d'exécution est le

- ✓ De mémoriser :
 - les
 - le (le programme peut donc être manipulé comme une donnée).

Exemple : effectuer l'opération $12+13=25$

- ✓ La mémoire contient :
 - les données : 12 et 13.
 - le résultat : 25
 - le programme : l'addition « + »
- ✓ Le processeur va effectuer le calcul sur ces 2 nombres.
- ✓ Les données se déplacent de la mémoire vers le processeur et retournent ensuite dans la mémoire.
Le chemin emprunté par les données est appelé le (exemple : les nappes de câbles dans l'ordinateur)
Dans un programme, les données seront appelées les

2.4 Etats logiques, du transistor au processeur

a) Codage NRZ

- ✓ Vers la fin des années 1930, Claude Shannon démontra qu'à l'aide de "contacteurs" (interrupteurs) fermés pour "vrai" et ouverts pour "faux", il était possible d'effectuer des opérations logiques en associant le nombre " 1 " pour "vrai" et "0" pour "faux".
- ✓ Un ordinateur est une machine qui ne peut que comparer des hautes tensions et des basses tensions électriques. Si nous avons une tension de 5 Volt cela correspondra à l'état logique 1, et si nous avons 0 Volt, l'état logique sera 0.

Tension en Volt :	État logique :
0 V	0
5 V	1

C'est là l'origine du fait que l'ordinateur « compte en » : il compare des 0 et des 1.

- ✓ Ce codage, appelé codage en (Non Retour à Zéro), tient compte de seuils de tensions : si la tension est au-dessous d'un seuil donné, c'est un "0", si la tension est au-dessus d'un autre seuil, c'est un "1". Il n'existe pas d'état intermédiaire.

Les circuits qui manipulent des tensions n'ont pas une précision parfaite et une petite perturbation électrique pourrait alors transformer un "0" en "1". Ces 2 seuils permettent d'avoir une marge de sécurité et de limiter ces erreurs.



b) Transistors

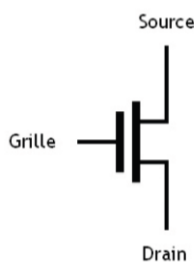
Cette information 0 ou 1 est stockée dans un

Un transistor est un composant électronique en

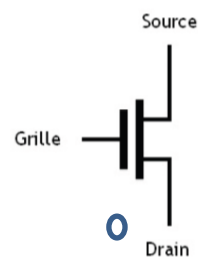
Le silicium est un métal dit: ça signifie qu'il n'est pas un isolant comme le plastique mais pas tout à fait un conducteur comme le cuivre.

Il est possible de choisir, en lui imposant une tension, s'il doit ou non conduire l'électricité.

Exemple : transistors NMOS / PMOS



NMOS



PMOS

Le transistor est un composant ayant 3 "fils" conducteurs ou broches appelées la Grille, la Source, le Drain ; on peut appliquer sur ces broches une tension électrique représentant un "0" ou un "1".

Le transistor PMOS peut être vu comme un interrupteur qui réagit en fonction de la grille:

- Quand on applique une tension sur la grille (Grille = « 1 »), le transistor est: il se ferme et applique la tension de Source sur le Drain.
- A l'inverse, si la Grille est à 0, le transistor est et ne relie pas la Source au Drain.

Dit autrement, le Grille commande la fermeture (Grille = « 0 ») ou l'ouverture (Grille = « 1 ») d'un "canal" entre la Source et le Drain. Quand on fournit une tension sur la Source, ce canal permet donc de transmettre (canal fermé) ou pas (canal ouvert) cette tension sur le Drain.

Le transistor **PMOS** fonctionne dans l'autre sens : il est passant quand la Grille est à 0, et bloqué quand on applique une tension sur la grille.

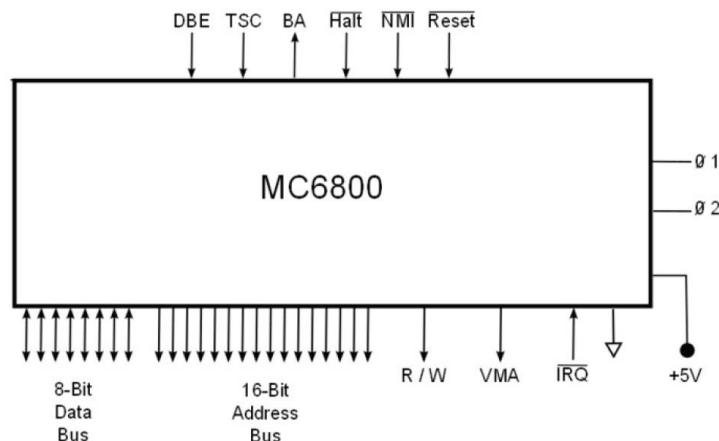
c) Portes logiques et processeur

- ✓ En assemblant quelques transistors ensemble, on est capable de créer des : ET, OU, Inverseur etc ...
- ✓ En associant plusieurs portes logiques, on est capable de créer des composants plus évolués pour effectuer des : compteur, additionneur, multiplieur, mémoire etc ...
- ✓ En associant beaucoup de ces portes logiques, on est capable d'effectuer un très grand nombre de fonctions complexes et de créer un

Un processeur possède un grand nombre d'entrées / sorties et de portes logiques. Les entrées vont piloter ces portes logiques qui vont réagir et piloter les sorties.

Exemple

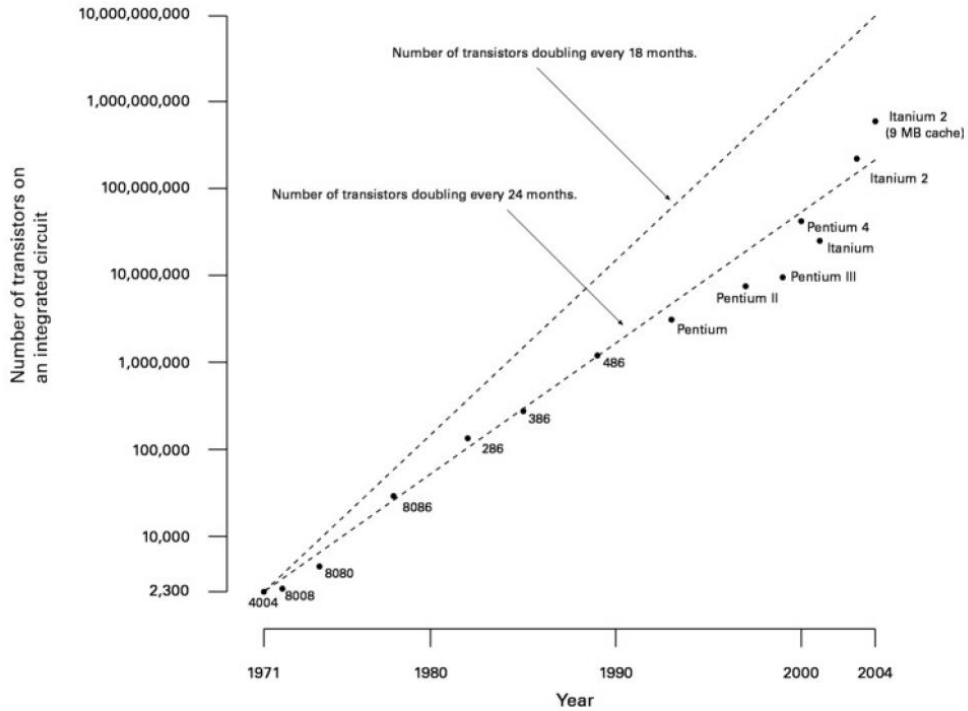
Le processeur des calculatrices TI-92 possède 68 000 transistors, possède 37 entrées / sorties et date de 1979.



- ✓ Un processeur actuel comporte **plusieurs milliards de transistors** et autour de 1000 entrées / sorties.

Le nombre de transistors sur un processeur suit à peu près la loi de Moore, du nom d'un des fondateurs d'Intel qui prédit alors cette loi empirique. Elle dit que la miniaturisation est telle que le nombre de transistors est doublé environ tous les deux ans.

Moore's Law



L'augmentation du nombre de transistors est possible par la miniaturisation des transistors. Mais ça ne sera pas possible indéfiniment : en 2013, les transistors sont gravés en 22 nanomètres, soit 22 milliardièmes des mètres : c'est la taille de quelques dizaines d'atomes !!!

⇒ Avez-vous entendu parler des ordinateurs quantiques ? ...

- ✓ Un programme est une suite d'instructions qui va piloter les portes logiques du processeur.

Le programme appellera par exemple des pour faire appel aux portes logiques, ou des pour faire des calculs simples.

2.5 Les mémoires

- ✓ Un ordinateur contient 4 principaux types de mémoires de taille et de vitesse différentes :
 - Les qui contiennent les informations qui doivent être conservées même après l'arrêt du système.
Il s'agit du auquel on accède très rarement (principalement lors du démarrage de l'ordinateur).
 - La qui stocke toutes les informations temporaires auxquelles le processeur doit accéder peu souvent mais qui doivent être conservées suffisamment longtemps.
Il s'agit par exemple de la **RAM**.
 - Les qui accélèrent l'accès de la mémoire principale.
Elles sont en accès direct à l'intérieur de la puce du processeur et sont souvent directement gérées ce-dernier.
 - Les très rapides qui contiennent les instructions ou les données que le processeur doit manipuler.
Ils sont à l'intérieur de la puce du processeur et sont gérés par le programme (lors de la compilation).
- ✓ On accède aux mémoires par : chaque case mémoire contient une donnée et se voit attribuer une adresse qui permet de l'identifier et de la sélectionner.
Les cases ci-dessous contiennent des données de 16 bits mais Windows 7 fonctionne avec des données de 32 ou 64 bits.

Exemple :

Adresse	Contenu mémoire
0	11101010 01011010
1	01111111 01110010
2	00000000 01111100
3	01010101 00000000
4	10101010 00001111
5	00000000 11000011

Un ordinateur ayant plusieurs mémoires, il existe une table qui définit à quel composants électroniques sont attribués les adresses.

2.6 Quelques notions d'algèbre de Boole

2.6.1 Définitions

Variable binaire

On appelle variable binaire (ou logique), une variable prenant ses valeurs dans l'ensemble $\{0, 1\}$.

Exemple : état d'un interrupteur, d'un bouton poussoir, la présence d'une tension,...

Soit x la variable associée à l'état d'un bouton poussoir, alors $x = 0$ (faux ou bas) signifie qu'il n'est pas actionné, $x = 1$ (vrai ou haut) signifie qu'il est actionné.

Equation logique

On appelle équation logique une combinaison de plusieurs variables logiques donnant l'état d'une variable dite de sortie associée. Cette combinaison est réalisée à l'aide d'opérations logiques.

Table de vérité

La table de vérité représente l'état de la variable de sortie pour chacune des combinaisons des variables d'entrée.

2.6.2 Opérateur NON (NOT)

L'opération (ou opérateur) NON est la fonction unaire qui affecte à la variable de sortie S l'état complémentaire de la variable d'entrée x .

Exemple en français :

Je souhaite acheter une voiture NON polluante. Les voitures correspondant à mon choix seront donc écologiques.

Equation :



(prononcer « x barre »)

Table de vérité :

Schéma électrique:

Algorithme:

2.6.3 Opérateur ET (AND)

L'opération ET est le produit logique. Le signe est celui de la multiplication (un point), mais on lit « et ». C'est un opérateur binaire qui affecte à la variable de sortie y l'état 1 si et seulement si les variables d'entrée x et y sont à 1 simultanément.

Exemple en français :

Je souhaite acheter une voiture bleue ET climatisée. Les voitures correspondant à mon choix seront donc à la fois bleues et climatisées.

Equation :



(penser à l'intersection d'ensembles)

Table de vérité :

Schéma électrique:

Algorithme:

2.6.4 Opérateur OU (OR)

L'opération OU est la somme logique. Le signe est celui de l'addition (+), mais on lit « ou ».

C'est un opérateur binaire qui affecte à la variable de sortie S l'état 1 si et seulement si une variable d'entrée x ou y est à 1.

Exemple en français :

Je souhaite acheter une voiture bleue OU climatisée. Les voitures correspondant à mon choix seront donc soit bleue, soit climatisée, soit les deux à la fois.

Equation :



(penser à l'union d'ensembles)

Table de vérité :

Schéma électrique:

Algorithme:

2.7 Introduction à la numération

2.7.1 Principes fondamentaux

La numération s'est construite autour d'un des 2 principes suivants :

1) Principe d'additivité

Le nombre de symboles est réduit et on répète autant de fois que nécessaire ... et il suffit de faire une somme pour connaître ce nombre.

C'est donc fastidieux pour les grands nombres.

Exemple: chiffres romains

MMM CC L V III se lit 3 1000 2 100 50 5 3 = 3253

2) Principe de position

- Un entier est représenté par une suite de symboles ou **chiffres** pris dans un ensemble ou alphabet donné.
- Ces chiffres ont une **position (ou poids)** précise dans la suite de symboles considérée. La signification du chiffre dépend donc de cette position.
- La **base** indique le nombre de symboles disponibles

Exemple, en base 10 :

$$\begin{aligned} 7659 &= 7 \text{ 1000 } 6 \text{ 100 } 5 \text{ 10 } 9 \\ &= 7 \times 1000 + 6 \times 100 + 5 \times 10 + 9 \times 1 \\ &= 7 \times 10^3 + 6 \times 10^2 + 5 \times 10^1 + 9 \times 10^0 \end{aligned}$$

3) Principe du zéro

Le zéro indique une position où il n'y a pas d'éléments.

Ainsi 10 signifie 0 unités et 1 dizaine en base 10, et de façon générale, en base b, 10 (lire « 1 » « 0 ») représente le nombre b.

$$(10)_2 = (2)_{10} \quad (10)_8 = (8)_{10} \quad (10)_{16} = (16)_{10}$$

4) Représentation d'un nombre par son polynôme

- Un nombre entier est une suite de symboles ou
- Pour 7659 : 7, 6, 5, 9
- Ces chiffres occupent une ou rang précis
- ici : 7, 6, 5, 9 occupent respectivement les positions 3, 2, 1, 0
- La indique le nombre de symboles disponibles
- ex : 10 en base 10 ...
- correspond au

ex : 7, 6, 5, 9 ont respectivement des poids de $10^3=1000$, $10^2=100$, $10^1=10$, $10^0=1$

- est le chiffre à la

- Conversion d'un nombre entier de 4 chiffres de base b en base 10 :

La suite de chiffres en base b s'interprète dans le système décimal (base 10) grâce à un polynôme :

.....

2.7.2 Les différentes bases utilisées

- **Base décimale** $b = 10$ avec les chiffres : $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.
C'est la base usuelle, utilisée en particulier dans le système métrique des mesures physiques.
- **Base binaire** $b = 2$ avec les chiffres : $\{0, 1\}$.
Utilisée en informatique, en électricité, base de la logique booléenne ou de l'algèbre de Boole, ...
C'est aussi la base "minimale" : on ne peut avoir une base qui ne contienne qu'un élément.
- **Base octale** $b = 8$ avec les chiffres : $\{0, 1, 2, 3, 4, 5, 6, 7\}$.
Utilisée par exemple pour les droits d'accès aux fichiers sous Linux, `chmod 755` au lieu de `rwxr-xr-x`
- **Base hexadécimale** $b = 16$ avec les chiffres :
 $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$.
Très utilisée en micro-informatique (langage assembleur), cette base fournit une représentation compacte des données binaires.
- **Base 5** avec les chiffres : $\{0, 1, 2, 3, 4\}$.
Compter jusqu'à 30 avec ses deux mains : les doigts d'une main expriment les unités ceux de l'autre expriment des paquets de cinq.
- **Base 12** établie sur les signes du zodiaque.
Utile à cause de la divisibilité par 2, 3, 4 et 6.
Utilisée dans le commerce (oeufs, huîtres, ...) et pour les heures dans une journée (RV à 2h, cours de 11h à 12h, ...).
- **Base 20** construite à partir des doigts et des orteils.
Quelques traces en sont "quatre-vingts", "quatre-vingt-dix", Hôpital des Quinze Vingts, ...
- **Base 60**, il y a 360° dans le cercle ; 60 minutes dans une heure, 60 secondes dans une minutes ;
On a toujours l'expression "soixante-dix", ...

⇒ Nous nous limiterons aux conversions entre les bases 10, 2, 8, 16

2.7.3 La base 2 : le binaire

1) Le Bit

Dont l'acronyme signifie « binary digit », c'est-à-dire « nombre binaire ». C'est la plus petite unité d'information manipulable par une machine numérique.

Il est possible de représenter physiquement cette information binaire :

- par un signal électrique ou magnétique, qui, lorsqu'il atteint une certaine valeur, correspond à la valeur 1 (ex : +5v en électricité)
 - par des aspérités géométriques dans une surface (ex : les creux dans un cd-rom)
- ✓ 1 bit permet donc de représenter 2 états différents :

0
1

- ✓ 2 bits permettent donc de représenter $2^2 = 4$ états différents

0	0

- ✓ bits permettent donc de représenter $2^3 = 8$ états différents
- ✓ bits permettent donc de représenter $2^4 = 16$ états différents
- ✓ n bits permettent donc de représenter 2^n états différents

Exercices :

Combien d'états différents peut représenter une donnée composée de 64 bits ?

On veut représenter chacune des sept couleurs de l'arc-en-ciel par un mot, les sept mots devant être distincts et de même longueur. Quelle est la longueur minimale de ces mots ?

Trouvez trois informations de la vie courante qui peuvent être exprimées par un booléen

2) L'octet (byte)

- ✓ (en anglais byte) est une unité d'information composée de Il permet de stocker un caractère, tel une lettre, un chiffre ... Ce regroupement de nombres par série de 8 permet une lisibilité plus grande, au même titre que l'on apprécie, en base décimale, de regrouper les nombres par trois pour pouvoir distinguer les milliers.

Par exemple, le nombre 1 256 245 est plus lisible que 1256245.

- ✓ Une unité d'information composée de est généralement appelée (en anglais)
- ✓ Une unité d'information de de longueur est appelée (en anglais, d'où l'appellation dword).
- ✓ Pour un octet, on a $2^8=256$ possibilités de valeurs différentes:
 - le plus petit nombre est 0 (représenté par huit "0" 00000000),
 - le plus grand est 255 (représenté par huit chiffre "1" 11111111),

Les unités de mesures de l'octet

Auparavant (avant 1998), on utilisait les unités suivantes pour mesurer les différentes valeurs de l'octet :

- Un kilooctet (Ko) = 2^{10} octets = 1024 octets
- Un mégaoctet (Mo) = 220 octets = 1024 Ko = 1 048 576 octets
- Un gigaoctet (Go) = 230 octets = 1024 Mo = 1 073 741 824 octets
- Un téraoctet (To) = 240 octets = 1024 Go = 1 099 511 627 776 octets

Cette notation est encore très répandue, notamment dans certains logiciels et mêmes certains systèmes d'exploitation. Cependant, en 1998, l'IEC (un organisme international) a statué sur cette notation et a décidé d'un standard que voici :

- Un kilooctet (Ko) = 1000 octets
- Un mégaoctet (Mo) = 1000 Ko = 1 000 000 octets
- Un gigaoctet (Go) = 1000 Mo = 1 000 000 000 octets
- Un téraoctet (To) = 1000 Go = 1 000 000 000 000 octets

Il est à noter que l'utilisation du terme "byte" est plus fréquemment utilisée par la communauté informatique, que le terme "octet" qui, lui, est purement francophone. Ce qui donne les notations suivantes : kilobyte (KB), megabyte (MB), Gigabyte (GB) et terabyte (TB)

À ne pas confondre dans les notations : b et B. En effet, *b* représente un bit et *B* représente un Byte.

3) L'addition en base 2

Pour additionner 2 nombres en base 2 il suffit de se rappeler que $1+1 = (2)_{10} = (10)_2$

Dès que la somme des 2 chiffres dépasse $(2)_{10}$, on pose une retenue.

Exemple : $11\ 1111 + 1\ 1000 = ?$

2.7.4Table de conversion jusqu'à 15 en bases 10, 2, 8, 16

Arbre	Nb binaire Base 2				Nb octal Base 10	Nb dec Base 10	Nb hexa Base 16

2.7.5 Conversions de nombres binaires, octals, hexadécimaux en décimaux

Méthode par décomposition polynomiale

- ⇒ Remplir un tableau de 3 lignes contenant : les chiffres, leur position, leur poids et utilise la formule de la décomposition polynomiale.

Exemple : conversion de $(1001\ 1011)_2$ en décimal

Exemple : conversion de $(5427)_8$ en décimal

Exemple : conversion de $(A12F)_{16}$ en décimal

Exercices

Convertis ces nombres du **binaire** vers le **décimal** :

10111100, 00110100, 01111001, 00100101, 11011000

Convertis ces nombres de l'**octal** vers le **décimal** :

1127, 72, 654, 21, 577, 123, 257, 606, 3761, 5735

Convertis de l'**hexadécimal** vers le **décimal** :

AF18, 5C12, B83, E411, ABC, ABBA, ACDC, A12, 112

2.7.6 Conversions de nombres décimaux en binaires, octals, hexadécimaux

1) Méthode des divisions successives

⇒ Effectuer la division euclidienne successive par la base jusqu'à obtenir 0 puis récupérer les restes des divisions.

Attention on calcule d'abord les chiffres de poids faible puis de poids fort, il faut donc « inverser » la lecture des restes.

Exemple : décomposition de 7659 par division euclidienne

$$\begin{array}{rcl} \dots & & 7659 = \\ 7650 & +9 & \\ (760 & +5) \cdot 10 & +9 \\ ((70 & +6) \cdot 10 & +5) \cdot 10 & +9 \\ ((7 \cdot 10 & +6) \cdot 10 & +5) \cdot 10 & +9 \\ 7 \cdot 1000 & +6 \cdot 100 & +5 \cdot 10 & +9 \end{array}$$

Exemple : conversion de $(20)_{10}$ en binaire par division euclidienne

$$\begin{array}{rcl} \dots & & (20)_{10} = \\ 10 \cdot 2 & +0 & \\ (5 \cdot 2 & +0) \cdot 2 & +0 \\ ((2 \cdot 2 & +1) \cdot 2 & +0) \cdot 2 & +0 \\ (((1 \cdot 2 & +0)) \cdot 2 & +1) \cdot 2 & +0) \cdot 2 & +0 \\ 1 \cdot 16 & +0 \cdot 8 & +1 \cdot 4 & +0 \cdot 2 & +0 \end{array}$$

Exemple : conversion de $(882)_{10}$ en octal par division euclidienne

Exemple : conversion de $(976)_{10}$ en hexadécimal par division euclidienne

2) Méthode par soustraction

On calcule les poids de chaque chiffre et on effectue des soustractions successives.

Exemple : décomposition de 7659 par soustraction

...

Exemple : conversion de $(20)_{10}$ en binaire par soustraction

...

Exemple : conversion de $(882)_{10}$ en octal par soustraction

Exemple : conversion de $(976)_{10}$ en hexadécimal par soustraction

3) Exercices

Convertis ces nombres **décimaux** en **binaires** en utilisant la 1^{ère} méthode:

9, 15, 23, 28, 67, 255, 234

Convertis ces nombres **décimaux** en **binaires** en utilisant la 2^{ème} méthode:

19, 136, 119, 176, 28, 199

Convertis ces nombres **décimaux** en **octal** :

366, 752, 1024, 2154, 97, 121, 2017, 88, 1875, 333

Convertis ces nombres **décimaux** en **hexa** :

762, 1128, 512, 145, 362, 555, 1245, 2569, 321, 111

2.7.7 Conversions de binaires en octal, hexadécimaux et inversement

1) Utilisation de la table de conversion

Toutes ces bases sont des puissances de 2: $2^1, 2^3, 2^4$

⇒ Pour écrire 1 chiffre en **octal** (soit $8=2^3$ symbols), on a besoin de **3** bits

$(0)_8 = (000)_2 \quad \dots \quad (7)_8 = (111)_2$

⇒ Pour écrire 1 chiffre en **hexa** (soit $16=2^4$ symbols), , on a besoin de **4** bits

$(0)_{16} = (0000)_2 \quad \dots \quad (15)_{16} = (1111)_2$

⇒ et il suffit d'utiliser la table de conversion du paragraphe 2.3

2) Exercices

Convertis ces nombres **octals** en **binaires**

557, 326, 112, 241, 771, 677, 652, 213, 421

Convertis ces nombres **hexa** en **binaires**

C12, 2A8, A4, 52C, ABF

Convertis ces nombres **binaires** en **octal** :

111 110 011 010 111, 110 100 100 010, 001 110 010 010, 110 010 100,

111 011 001 101, 111 001 010 100 101

Convertis ces nombres **binaires** en **hexa** :

1100 1010 1111, 0101 0101 0101, 1011 1001 0011 1010, 1101 0010 0011 1010,

0111 1011 1001 1101

3 Introduction à l'algorithmique et à la programmation

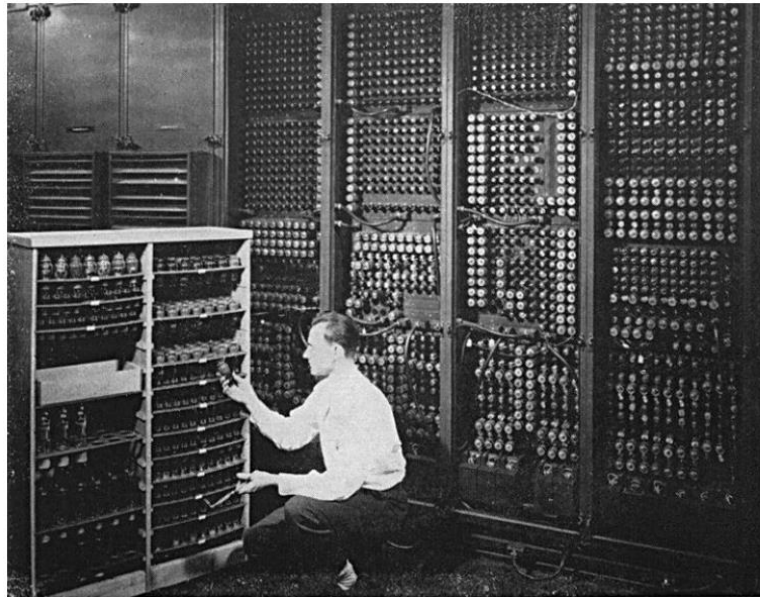
3.1 Un peu d'histoire

3.1.1 Bref historique des instruments de calculs

- 1100 : Boulier
- 1650-1970 : Règle à calcul
- 1642 : Machine à additionner, à multiplier (ex : la "Pascaline" de Blaise Pascal)
- 1938-1946 : Machines électroniques (Z3, ENIAC)
- 1972 : Calculatrice (scientifique) de poche électronique
- 1978 : Ordinateur personnel

Exemple : ENIAC

- 30 tonnes sur 167 m².
- Nombres signés de 10 chiffres : 5.000 additions simples/seconde,
- 357 multiplications/sec, 38 divisions/sec



3.1.2 Histoire des langages informatiques

- ⇒ Voir Chapitre 6 Programmation et langages du site ci-dessous
<https://www.apprendre-en-ligne.net/info/programmation/index.html>
- ✓ En 1936, la publication de l'article fondateur de la science informatique *On Computable Numbers with an Application to the Entscheidungsproblem*, par Alan Mathison **Turing**, allait donner le coup d'envoi à la création de l'ordinateur programmable. Il y présente sa **machine de Turing**, le premier calculateur universel programmable, et invente les concepts et les termes de programmation et de programme.
- ✓ En 1948, Konrad **Zuse** publie un article sur son langage de programmation qu'il a développé entre 1943 et 1945 : le **Plankalkül**. Zuse le considère comme étant le premier langage de haut niveau.

- ✓ C'est à partir des années 50 que l'on verra apparaître les premiers langages de programmation modernes. Voici les créateurs des langages les plus utilisés :
 - 1954 : John **Backus**, inventeur de Fortran
 - 1958 : John **McCarthy**, inventeur de LISP
 - 1959 : Grace **Hopper**, surnommée « la mère du langage COBOL »
 - 1963 : John George **Kemeny**, concepteur du BASIC
 - 1972 : Dennis **Ritchie** et Ken **Thompson**, inventeurs du langage C
 - 1977 : Niklaus **Wirth** inventeur de Pascal (1970) et Modula-2
 - 1985 : Bjarne **Stroustrup**, développeur de C++
 - 1991 : Guido **van Rossum**, créateur de Python
 - 1991 : James **Gosling** et Patrick **Naughton**, créateurs de Java.

3.1.3 Evolution des langages informatiques

On distingue aujourd'hui cinq générations de langages.

- ✓ **La première génération** est le , ou code machine. On parle aussi de langage natif. Il est composé d'instructions et de données à traiter codées en binaire. C'est le seul langage qu'un ordinateur peut traiter directement.

Voici à quoi peut ressembler un programme en langage machine :

A1 01 10 03 06 01 12 A3 01 14

Il s'agit de la représentation d'un programme permettant d'additionner les valeurs de deux cases mémoire et de stocker le résultat dans une troisième case. On voit immédiatement la difficulté d'un tel langage...

- ✓ **La deuxième génération** est le : le code devient lisible et compréhensible par un plus grand nombre d'initiés. Il existe en fait un langage assembleur par type de processeur.

Le programme précédent écrit en assembleur donnerait ceci :

```
MOV AX, [0110]
ADD AX, [0112]
MOV [0114], AX
```

Il reste utilisé dans le cadre d'optimisations, mais a été supplanté en popularité par les langages plus accessibles de troisième génération.

- ✓ **La troisième génération** utilise une syntaxe Proposés autour de 1960, ces langages ont permis un gain énorme en lisibilité et en productivité. Ils ne dépendent plus du processeur, comme c'était le cas des générations précédentes, mais d'un compilateur spécifique du processeur. L'idée de portabilité des programmes était lancée.

La plupart des langages de programmation actuels sont de troisième génération. On trouve dans cette catégorie tous les grands langages : **Ada, Algol, Basic, Cobol, Eiffel, Fortran, C, C++, Java, Perl, Pascal, Python, Ruby, ...** Cette

génération couvre d'ailleurs tant de langages qu'elle est souvent subdivisée en catégories, selon le paradigme particulier des langages.

- ✓ Les langages de **quatrième génération**, abrégés L4G, souvent associée à des , se situent un niveau au-dessus, en intégrant la gestion de l'interface utilisateur et en proposant un langage moins technique, plus proche de la syntaxe naturelle.

Ils sont conçus pour un travail spécifique : gestion de base de données (**Microsoft Access**, **SQL**), production graphique (**Postscript**), création d'interface (**4D**).

- ✓ **La cinquième génération** de langages correspond aux langages destinés à résoudre des problèmes à l'aide de contraintes, et non d'algorithmes écrits. Ces langages reposent beaucoup sur la logique et sont particulièrement utilisés en intelligence artificielle. Parmi les plus connus, on trouve **Prolog**, dont voici un exemple :

```
frère_ou_sœur(X,Y) :- parent(Z,X), parent(Z,Y), X \= Y.  
parent(X,Y) :- père(X,Y).  
parent(X,Y) :- mère(X,Y).  
mère(trude, sally).  
père(tom, sally).  
père(tom, erica).  
père(mike, tom).
```

Il en résulte que la demande suivante est évaluée comme vraie :

```
?- frère_ou_sœur(sally, erica).  
oui.
```

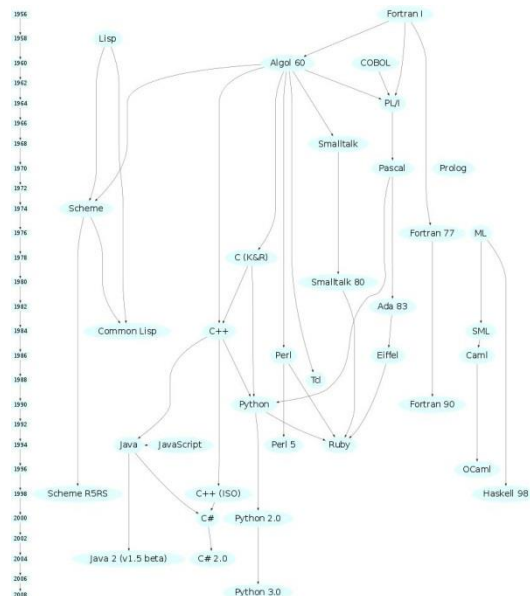
Ce qui signifie que Sally et Erica sont sœurs. En effet, Sally et Erica ont le même père (Tom).

3.1.4 Quelques langages courants

Il existe environ 2500 langages de programmation, certains étant très généraux (**C**, **Java**), d'autres hyper-spécialisés.

Par exemple, **RobotProg** permet de déplacer un robot virtuel, **R** est un langage pour la statistique, etc

Par comparaison, une majorité de spécialistes s'accordent sur le chiffre d'environ 6000 langues humaines parlées et/ou écrites de par le monde.



3.1.5 « Hello world ! »

C'est dans un memorandum interne de Brian **Kernighan**, *Programming in C : A tutorial*, écrit en 1974 dans les laboratoires *Bell* , que l'on trouve la première version d'un mini-programme affichant à l'écran « Hello World! ». Voici comment cela s'écrit dans divers langages:

Ada

Le nom **Ada** a été choisi en l'honneur d'Ada Lovelace, qui est supposée avoir écrit le premier programme de l'histoire.

La première version d'**Ada** remonte à 1983.

```
with Ada.Text_IO;
use Ada.Text_IO;
procedure Bonjour is
begin -- Bonjour
  Put("Hello world!");
end Bonjour;
```

Assembleur X86 sous DOS

```
cseg segment
  assume cs:cseg, ds:csegorg 100h
  main proc jmp debut
  mess db 'Hello world!$' debut:
  mov dx, offset mess mov ah, 9
  int 21h ret
  main endp cseg endsend main
```

Basic

BASIC est un acronyme pour Beginner's All- purpose Symbolic Instruction Code. Il a été conçu à la base en 1963 par John George **Kemeny** et Thomas Eugene **Kurtz**.

```
10 PRINT "Hello world!"
```

C

Inventé au des années 1970 avec UNIX, **C** est devenu un des langages les plus utilisés.

```
#include <stdio.h>
int main()/* ou int argc, char *argv[] */
{
    printf("Hello world!\n");
    return 0;
}
```

C++

Bjarne **Stroustrup** a développé **C++** au cours des années 1980. Il s'agissait d'améliorer le langage **C**.

```
#include <iostream>
int main()
{
    std::cout << "Hello world!" << std::endl;return 0;
}
```

Fortran 77

Fortran (FORMula TRANslator) est utilisé dans les applications de calcul scientifique.

```
PROGRAM BONJOUR
WRITE (*,*) 'Hello world!'
END
```

Java

Java a été créé par Sun Microsystems, et présenté officiellement le 23 mai 1995.

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```

Javascript

JavaScript est un langage de programmation de scripts utilisé dans les pages web interactives .

```
document.write("Hello world!");
```

Python 1 et 2

```
print "Hello world!"
```

Python 3

```
print("Hello world!")
```

3.2 Qu'est-ce que l'algorithmique ?

- ✓ Pour plus d'explications, consultez [31]
- ✓ Un algorithme 'est une suite d'instructions, qui une fois exécutées correctement, conduit à un résultat donné. Si l'algorithme est juste, le résultat est le résultat voulu, si l'algorithme est faux, le résultat est aléatoire ...
- ✓ Pour fonctionner, un algorithme doit donc contenir uniquement des instructions compréhensibles par celui qui devra l'exécuter, en l'occurrence les ordinateurs.

Historiquement, on peut faire remonter la notion d'algorithme aux Babyloniens (1800 av. JC, résolution de certaines équations) et à Euclide (3e siècle av. JC, pgcd, division entière).

Algorithme est un terme dérivé du nom du mathématicien Muhammad ibn Musa al-Khwarizmi (Bagdad, 783-850) qui a notamment travaillé sur la théorie du système décimal et sur les techniques de résolution d'équations du 1er et 2ème degré.

3.3 Faut-il être matheux pour être bon en algorithmique ?

La maîtrise de l'algorithmique requiert deux qualités complémentaires:

- ✓ il faut avoir une certaine **intuition**, car aucune recette ne permet de savoir à priori quelles instructions permettront d'obtenir le résultat voulu. C'est certain, il y a des gens qui possèdent au départ davantage cette intuition que les autres ; cependant, les réflexes, cela s'acquiert, et ce qu'on appelle l'**intuition n'est finalement que de l'expérience tellement répétée que le raisonnement, au départ laborieux, finit par devenir « spontané »**.
- ✓ il faut être **methodique et rigoureux**. En effet, chaque fois qu'on écrit une série d'instructions qu'on croit justes, il faut **systematiquement se mettre mentalement à la place de la machine** qui va les exécuter, armé d'un papier et d'un crayon, afin de vérifier si le résultat obtenu est bien celui que l'on voulait. Cette opération ne requiert pas la moindre once d'intelligence. Mais elle reste néanmoins indispensable, si l'on ne veut pas écrire à l'aveuglette.

Et petit à petit, à force de pratique, vous verrez que vous pourrez faire de plus en plus souvent l'économie de cette dernière étape : l'expérience fera que vous « verrez » le résultat produit par vos instructions, au fur et à mesure que vous les écrirez. Naturellement, cet apprentissage est long, et demande des heures de travail patient. Aussi, dans un premier temps, évitez de sauter les étapes : **la vérification methodique, pas à pas, de chacun de vos algorithmes représente plus de la moitié du travail à accomplir... et le gage de vos progrès.**

3.4 Les 4 grandes familles d'instructions

L'ADN, qui est en quelque sorte le programme génétique, l'algorithme à la base de construction des êtres vivants, est une chaîne construite à partir de quatre éléments invariables. Ce n'est que le nombre de ces éléments, ainsi que l'ordre dans lequel ils sont arrangés, qui vont déterminer si on obtient une puce ou un éléphant. Et tous

autant que nous sommes, avons été construits par un « programme » constitué uniquement de ces quatre briques.

Enfin, les ordinateurs, quels qu'ils soient, ne sont fondamentalement capables de comprendre que quatre catégories d'ordres (en programmation, on n'emploiera pas le terme d'ordre, mais plutôt celui d'**instructions**).

Ces quatre familles d'instructions sont :

- ✓ ,
- ✓ ,
- ✓ ,
- ✓

Un algorithme informatique se ramène donc toujours au bout du compte à la combinaison de ces quatre petites briques de base. Il peut y en avoir quelques-unes, quelques dizaines, et jusqu'à plusieurs centaines de milliers dans certains programmes de gestion. Cependant, la taille d'un algorithme ne conditionne pas en soi sa complexité : de longs algorithmes peuvent être finalement assez simples, et de petits très compliqués.

3.5 Algorithmique et programmation

Pourquoi apprendre l'algorithmique pour apprendre à programmer ? En quoi a-t-on besoin d'un langage spécial, distinct des langages de programmation compréhensibles par les ordinateurs ?

Parce que l'algorithmique exprime les instructions résolvant un problème donné indépendamment des particularités de tel ou tel langage.

Apprendre l'algorithmique, c'est apprendre à manier la structure logique d'un programme informatique. Cette dimension est présente quelle que soit le langage de programmation ; mais lorsqu'on programme dans un langage (en C, Python ...) on doit en plus apprendre les problèmes de syntaxe, ou de types d'instructions, propres à ce langage. **Apprendre l'algorithmique de manière séparée, c'est donc cerner les difficultés pour mieux les vaincre.**

3.6 Pseudo code

Historiquement, plusieurs types de notations ont représenté des algorithmes.

Il y a eu notamment une représentation graphique, avec des carrés, des losanges, etc. qu'on appelait des organigrammes. Aujourd'hui, cette représentation est quasiment abandonnée, pour le « pseudocode ».

Le « **pseudocode** » est une notation qui ressemble à un langage de programmation authentique dont on aurait évacué la plupart des problèmes de syntaxe.

3.7 Etapes de programmation : l'algorithmique

Avant d'écrire ton programme, tu dois décrire avec **des mots, un papier et un crayon** ce que doit faire le programme : c'est que l'on appelle l'algorithme.

Il est composé de 4 sections :

- ✓ **Liste des variables** utilisées par l'algorithme 'à remplir au fur et à mesure de la résolution de l'exercice. Les noms doivent être claires, commentés si besoin pour que le code soit compréhensible par une autre personne ou par toi-même dans 6 mois.
- ✓ **Données ou entrées**: description des données à fournir à l'algorithme, par exemple les valeurs entrées au clavier
- ✓ **Résultat ou sorties**: description du ou des résultats délivrés par l'algorithme, par exemple le résultat d'une formule
- ✓ **Algorithme** : description des formules ou des étapes de résolution du problème
- ✓ Si une variable change de valeurs au cours du programme (comme dans une boucle répétitive), dessine une table d'états reprenant :
 - une colonne pour chaque variable,
 - leurs valeurs initiales (ie valeurs des variables avant le démarrage de la boucle),
 - les valeurs prises successivement au cours des itérations,
 - l'expression de remplacement afin de modifier l'état de la variable à chaque itération.On effectue ainsi « à la main » le travail de l'ordinateur en indiquant ligne par ligne les valeurs que prendront les variables au fur et à mesure des itérations successives.

Cette étape d'écriture de l'algorithme est indispensable et est très souvent plus difficile que la programmation !!!

- ⇒ **En dehors des tous premiers exercices de découverte, tu devras décrire ces 4 sections en commentaire au début de ton programme.**

3.8 L'algorithmique : un changement fondamental

- ✓ Allez voir la vidéo de Bernard Chazelle (collège de France)
www.youtube.com/watch?v=qXASWVv8Ec&feature=youtu.be

Table des matières

1	Bibliographie.....	1
1.1	Cours et formations	1
1.2	Formations en ligne autonomes	1
1.3	Algorithmique.....	2
1.4	Outils.....	2
1.5	Installation et configuration de Visual Studio Code.....	3
2	Quelques notions sur le fonctionnement d'un ordinateur.....	5
2.1	Introduction.....	5
2.2	Rappels de définitions	5
2.3	Fonctions de base d'un ordinateur.....	5
2.4	Etats logiques, du transistor au processeur.....	6
2.5	Les mémoires.....	10
2.6	Quelques notions d'algèbre de Boole.....	11
2.6.1	Définitions.....	11
2.6.2	Opérateur NON (NOT)	11
2.6.3	Opérateur ET (AND)	12
2.6.4	Opérateur OU (OR)	13
2.7	Introduction à la numération.....	14
2.7.1	Principes fondamentaux	14
2.7.2	Les différentes bases utilisées	15
2.7.3	La base 2 : le binaire	16
2.7.4	Table de conversion jusqu'à 15 en bases 10, 2, 8, 16.....	18
2.7.5	Conversions de nombres binaires, octals, hexadécimaux en décimaux.....	19
2.7.6	Conversions de nombres décimaux en binaires, octals, hexadécimaux.....	20
2.7.7	Conversions de binaires en octal, hexadécimaux et inversement	22
3	Introduction à l'algorithmique et à la programmation.....	23
3.1	Un peu d'histoire.....	23
3.1.1	Bref historique des instruments de calculs.....	23
3.1.2	Histoire des langages informatiques.....	23
3.1.3	Evolution des langages informatiques	24
3.1.4	Quelques langages courants.....	25
3.1.5	« Hello world ! ».....	26
3.2	Qu'est-ce que l'algorithmique ?.....	28
3.3	Faut-il être matheux pour être bon en algorithmique ?.....	28
3.4	Les 4 grandes familles d'instructions.....	28
3.5	Algorithmique et programmation.....	29
3.6	Pseudo code.....	29
3.7	Etapes de programmation : l'algorithmique.....	29
3.8	L'algorithmique : un changement fondamental.....	30