Chapitre 0 **Introduction générale et outils de travail**

Le socle de la matière SNT est l'**informatique**, domaine que l'on peut à ce stade présenter très simplement :

Traitement automatisé de l'information

(*Le mot « informatique » est une contraction des mots « information » et « automatique »*).

Depuis maintenant plusieurs dizaines d'années, le domaine de l'informatique se développe très rapidement en prenant de plus en plus de place dans notre quotidien.

Notre smartphone est l'illustration la plus frappante d'avancées technologiques permanentes.

Toutefois, malgré leur grande variété, ces avancées se fondent en fait sur l'universalité et la flexibilité d'un petit nombre de concepts en interaction :

- Les **données**, qui représentent sous une forme numérique ***unifiée*** des **informations** très diverses : textes, images, sons, mesures physiques, sommes d'argent, etc.

- Les **algorithmes**, qui spécifient de façon abstraite et précise des traitements à effectuer sur les données à partir d'opérations élémentaires. Nous aborderons l'algorithmique selon les besoins, essentiellement sous la forme de discussions préliminaires à l'élaboration de codes (de programmes), sans en faire un chapitre ou un thème explicite de notre matière.

  Nous pouvons tout de suite prendre un exemple : élaborer un algorithme qui représenterait la règle du jeu consistant à faire deviner un nombre entre 0 et 100 (discussion libre avec la classe).

- Les **langages**, qui permettent de traduire les algorithmes abstraits en **programmes** textuels ou graphiques de façon qu'ils soient exécutables par les machines. La langue humaine utilisée dans les langages de programmation est essentiellement l'anglais.

  - Des noms de langages de programmation : Python, html, css, c++, java, etc.

  - ***Un exemple en c++ (mesure et affichage d'une température toute les secondes à l'aide d'un microcontrôleur Arduino)*** *:*

```
int potPin = 0;
void setup()
{
Serial.begin(9600);
}
void loop()
{
int val;
int temp;
val=analogRead(0);
temp=val*(5/1023*100);
Serial.print("Tep:");
Serial.print(temp);
Serial.println("C");
delay(1000);
}
```

- ***Un exemple en Python (affichage de deux courbes représentant des fonctions mathématiques) :***

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 2*np.pi, 30)
y1 = np.cos(x)
y2 = np.sin(x)
plt.plot(x, y1)
plt.plot(x, y2)
plt.show()
```

- ***Les programmes présentés dans les deux exemples ci-dessus peuvent sembler difficile à comprendre et ils sont ici présentés volontairement nus. La plupart du temps ils sont accompagnés de commentaires qui sont une aide précieuse à la compréhension de chaque ligne d'instruction.***

Reprenons le programme « température » avec commentaires :

```
int potPin = 0;                 // initialise la broche analogique  sur laquelle sera branché
                                  //le capteur de température LM35
void setup()
{
Serial.begin(9600);          // réglage de la vitesse de transfert des données
}
void loop()
{
int val;                        // la variable « val sera un entier
int temp;                      // idem pour la variable « temp »
val=analogRead(0);          // val est affectée de la valeur reçue depuis la broche 0
                               // c'est-à-dire depuis le capteur de température
temp=val*(5/1023*100);    // une formule calcule la température à partir de la valeur récupérée
                               //précédemment et le résultat est affecté à la variable temp
Serial.print("Tep:");         // affiche « Tep = »
Serial.print(temp);          // affiche à la suite la valeur de la variable temp
Serial.println("°C");         //affiche  l'unité « °C »  à la suite et passe à la ligne
delay(1000);                 // attend une seconde avant de reprendre une mesure
}
```

Le programme de tracé des courbes en Python :

```
import numpy as np                  # pour pouvoir utiliser des fonctions mathématiques
import matplotlib.pyplot as plt   # Pour pouvoir tracer des graphes
x = np.linspace(0, 9, 30)    # définition de la coordonnée horizontale x,
                                    # bornes et nombre de points
y1 = np.cos(x)                 # définition de la coordonnée verticale y1 (la fonction cosx)
y2 = np.sin(x)                 # idem y2
plt.plot(x, y1)                # construction de la courbe f(x) = cosx
plt.plot(x, y2)                #construction de la courbe g(x) = sinx
plt.show()                     # affichage dans la fenêtre graphique
```

- ***Pour commencer à comprendre le langage Python : voir l'annexe « coder en python »***

- Les **machines**, et leurs systèmes d'exploitation, qui permettent d'exécuter des programmes en enchaînant un grand nombre d'instructions simples, assurant la persistance des données par leur stockage, et de gérer les communications. On y inclut les objets connectés et les réseaux.
  La machine ne comprend que deux actions, états, commandes, … symbolisés par deux chiffres : 0 et 1. Elle ne fonctionne qu'en manipulant des données et instructions qui ne peuvent pas s'exprimer autrement que sous la forme de combinaisons de 0 et de 1.

Voir l'annexe « Le langage binaire »

À ces concepts s'ajoute un élément transversal : les interfaces (un clavier d'ordinateur est une interface, idem pour un écran tactile, une webcam, …) qui permettent la communication avec les humains (l'acronyme « IHM », fréquemment utilisé signifie « Interface Homme Machine »…), la collecte des données (capteurs, …) et la commande des systèmes.

Quelques dernières remarques avant de nous lancer :

- Le monde de l'informatique peut être philosophiquement complémentaire du monde de la physique. La physique tente d'expliquer la nature, l'informatique est un monde artificiel (en opposition à « naturel ») créé par les êtres humains. On ne peut pas se considérer comme le « maitre du monde » lorsque l'on est physicien, on peut le croire lorsque l'on est informaticien, mais, rappelons-le, ce monde que l'on maitrise est un monde artificiel.

- Notre cursus de SNT pourra s'accompagner d'un regard critique sur ce monde numérique que l'on peut trouver envahissant ou qui n'est pas systématiquement la solution à tous les problèmes que nous rencontrons.

- Un environnement appelé PIX, les compétences informatiques à acquérir d'ici la fin du cursus lycée. (*explications pendant la séance*)

SNT :

- **« Sciences » (dictionnaire) :** ***Corps de connaissances constituées et articulées par l'observation, l'expérience, le calcul et la déduction logique.***

- **« Numériques » :** ***Les machines automatisées (les ordinateurs) ne comprennent qu'un langage chiffré, qui plus est constitué de seulement deux chiffres, 0 et 1 (on l'appelle donc langage binaire). C'est à cause du fonctionnement des composants fondamentaux d'un ordinateur, qui sont appelés à fonctionner selon deux modes bien distincts que l'on pourrait appeler « haut » et « bas » et auxquels on a préféré associer ces deux valeurs (0 et 1).***

- **« Technologiques » (dictionnaire) :** ***au stade technologique, les procédés ont été largement validés, les machines sont correctement mises au point, nous nous confrontons donc à un ensemble qui peut fonctionner à l'échelle industrielle ou qui peut être destiné au plus grand nombre.***

Les thèmes abordés :

- Réseaux (internet, le web, les réseaux sociaux).
- Les données structurées et leur traitement.
- Localisation, cartographie et mobilité.
- Informatique embarquée et objets connectés.
- La photographie numérique.

Annexe n°1 : je cause en binaire

Nous l'avons déjà présenté, le langage binaire ne dispose que de 2 caractères : 0 et 1

1) Compter

Si nous souhaitons compter en binaire de 0 à 31 (soit les 32 premiers nombres entiers positifs), nous procédons logiquement de la façon suivante (les quatre dernières lignes sont à compléter) :

Décimal	Binaire
0	0
1	1
2 (c'est 2^1)	10
3	11
4 (c'est 2^2)	100
5	101
6	110
7	111
8 (c'est 2^3)	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111
16 (c'est 2^4)	10000
31	?
32	?
255	?
$256 = 2^8$	?

Même si $256 = 2^8$, c'est bien 255 qui est la 2^8 ème valeur de la liste.
Nous comprenons que la 2^n ème valeur de la liste se présentera sous la forme d'une suite de n valeurs 1 en langage binaire.

Si nous savons que nous n'allons pas compter au-delà de 255 (partant de 0), nous savons que nous n'aurons pas besoin de nombre binaire plus grand que 11111111. Nous pouvons alors noter le plus petit nombre (0) : 00000000.
Un nombre (ou un caractère quelconque, ou une instruction, etc.) est ainsi codé sous la forme d'un **octet**.

2) Bits, octets, ko, Mo, Go, etc.

La plus petite information circulant dans une machine informatique est le bit, il s'agira d'un 0 ou d'un 1.

Nous raisonnons volontiers en octet : 1 octet = 8 bits…
… puis en multiples de l'octet (kilo octet : 1ko = 1000 octets = 10^3 octets,
Mégaoctet : 1Mo = 10^6 octets,
Etc.

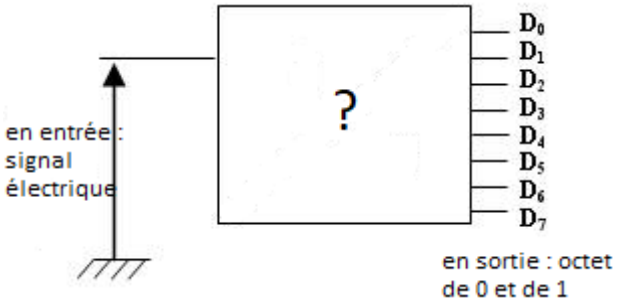
Lorsque nous décrivons un transfert de données entre deux machines, nous manipulons deux informations :
- la taille des données, en Mo, Go, etc.
- La vitesse de transfert des données, elle s'exprime la plupart du temps en bit par seconde (« bps », $bit.s^{-1}$, bits/s etc.). On trouve souvent une unité plus ancienne, le baud (Bd) qui ne correspond pas forcément au bit/s, mais nous considèrerons dans la plupart des cas que ces unités sont identiques.

Ces unités et données sont bien entendu très utiles pour de nombreux calculs.
Exemple :
Une photo est stockée sous la forme d'un fichier numérique de 10 Mo. Elle est transféré du smartphone à l'ordinateur par le canal sans fil bluetooth.à 2 Mbits/s.
Quelle est la durée du transfert ?

3) Numérisation des informations fournies à la machine

Comment, par exemple, l'octet 00011100 est-il transmis à la machine ?

Cette opération ne sera pas détaillée, mais l'important est de comprendre qu'elle passe obligatoirement par la conversion d'un signal électrique en signal numérique qui doit se présenter sous la forme d'un octet de 0 et de 1.



Le circuit électronique représenté par le carré « ? » convertit le signal électrique d'entrée en un ensemble de 8 signaux électriques représentant chacun des 1 et des 0 : un **octet**.
Selon l'instruction attendue (une lettre, une opération, …) le circuit électronique n'est pas le même, bien entendu.
Il n'est pas question pour nous de connaître la constitution des circuits électroniques (et bientôt quantiques) permettant ces transformations.

4) *A chaque caractère son code binaire : la table ASCII*

Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char	Dec	Hex	Oct	Bin	Char
0	0x00	000	0000000	NUL	32	0x20	040	0100000	space	64	0x40	100	1000000	@	96	0x60	140	1100000	`
1	0x01	001	0000001	SOH	33	0x21	041	0100001	!	65	0x41	101	1000001	A	97	0x61	141	1100001	a
2	0x02	002	0000010	STX	34	0x22	042	0100010	"	66	0x42	102	1000010	B	98	0x62	142	1100010	b
3	0x03	003	0000011	ETX	35	0x23	043	0100011	#	67	0x43	103	1000011	C	99	0x63	143	1100011	c
4	0x04	004	0000100	EOT	36	0x24	044	0100100	$	68	0x44	104	1000100	D	100	0x64	144	1100100	d
5	0x05	005	0000101	ENQ	37	0x25	045	0100101	%	69	0x45	105	1000101	E	101	0x65	145	1100101	e
6	0x06	006	0000110	ACK	38	0x26	046	0100110	&	70	0x46	106	1000110	F	102	0x66	146	1100110	f
7	0x07	007	0000111	BEL	39	0x27	047	0100111	'	71	0x47	107	1000111	G	103	0x67	147	1100111	g
8	0x08	010	0001000	BS	40	0x28	050	0101000	(	72	0x48	110	1001000	H	104	0x68	150	1101000	h
9	0x09	011	0001001	TAB	41	0x29	051	0101001	)	73	0x49	111	1001001	I	105	0x69	151	1101001	i
10	0x0A	012	0001010	LF	42	0x2A	052	0101010	*	74	0x4A	112	1001010	J	106	0x6A	152	1101010	j
11	0x0B	013	0001011	VT	43	0x2B	053	0101011	+	75	0x4B	113	1001011	K	107	0x6B	153	1101011	k
12	0x0C	014	0001100	FF	44	0x2C	054	0101100	,	76	0x4C	114	1001100	L	108	0x6C	154	1101100	l
13	0x0D	015	0001101	CR	45	0x2D	055	0101101	-	77	0x4D	115	1001101	M	109	0x6D	155	1101101	m
14	0x0E	016	0001110	SO	46	0x2E	056	0101110	.	78	0x4E	116	1001110	N	110	0x6E	156	1101110	n
15	0x0F	017	0001111	SI	47	0x2F	057	0101111	/	79	0x4F	117	1001111	O	111	0x6F	157	1101111	o
16	0x10	020	0010000	DLE	48	0x30	060	0110000	0	80	0x50	120	1010000	P	112	0x70	160	1110000	p
17	0x11	021	0010001	DC1	49	0x31	061	0110001	1	81	0x51	121	1010001	Q	113	0x71	161	1110001	q
18	0x12	022	0010010	DC2	50	0x32	062	0110010	2	82	0x52	122	1010010	R	114	0x72	162	1110010	r
19	0x13	023	0010011	DC3	51	0x33	063	0110011	3	83	0x53	123	1010011	S	115	0x73	163	1110011	s
20	0x14	024	0010100	DC4	52	0x34	064	0110100	4	84	0x54	124	1010100	T	116	0x74	164	1110100	t
21	0x15	025	0010101	NAK	53	0x35	065	0110101	5	85	0x55	125	1010101	U	117	0x75	165	1110101	u
22	0x16	026	0010110	SYN	54	0x36	066	0110110	6	86	0x56	126	1010110	V	118	0x76	166	1110110	v
23	0x17	027	0010111	ETB	55	0x37	067	0110111	7	87	0x57	127	1010111	W	119	0x77	167	1110111	w
24	0x18	030	0011000	CAN	56	0x38	070	0111000	8	88	0x58	130	1011000	X	120	0x78	170	1111000	x
25	0x19	031	0011001	EM	57	0x39	071	0111001	9	89	0x59	131	1011001	Y	121	0x79	171	1111001	y
26	0x1A	032	0011010	SUB	58	0x3A	072	0111010	:	90	0x5A	132	1011010	Z	122	0x7A	172	1111010	z
27	0x1B	033	0011011	ESC	59	0x3B	073	0111011	;	91	0x5B	133	1011011	[	123	0x7B	173	1111011	{
28	0x1C	034	0011100	FS	60	0x3C	074	0111100	<	92	0x5C	134	1011100	\	124	0x7C	174	1111100	|
29	0x1D	035	0011101	GS	61	0x3D	075	0111101	=	93	0x5D	135	1011101	]	125	0x7D	175	1111101	}
30	0x1E	036	0011110	RS	62	0x3E	076	0111110	>	94	0x5E	136	1011110	^	126	0x7E	176	1111110	~
31	0x1F	037	0011111	US	63	0x3F	077	0111111	?	95	0x5F	137	1011111	_	127	0x7F	177	1111111	DEL

D'autres problèmes se posent rapidement (et nous ne détaillerons pas les réponses correspondantes) :

- Comment représente-t-on les nombres négatifs ? Les nombres à virgule ?
- Les caractères issus d'autres alphabets sont-ils codés encore différemment ou bien reprennent-ils des codes déjà existants dans le tableau ASCII ?
- Etc.

5) Il n'y a pas que les caractères, il y a les instructions (cas d'une opération) :

L'information va se présenter souvent sur 32 bits, étant sous-entendu que le 1er octet en partant de la gauche désigne par exemple une opération (00000000 est le code signifiant « addition », 001001110 est celui de la multiplication, etc.) Les 24 bits suivants permettront de coder les opérandes (les valeurs impliquées dans l'opération à réaliser).

Annexe n° 2 : je code en python

Quelle est l'utilité d'un programme informatique ?

Il nous permet de réaliser automatiquement une série d'opérations et de résoudre ainsi rapidement un problème donné. Le programme s'exécute sur la base d'instructions que nous avons rédigées dans un langage approprié.

Prenons pour commencer le problème suivant (même exemple que dans la présentation des algorithmes) : il faut deviner le nombre entre 0 et 100 qui a été écrit au dos d'un papier par l'un d'entre nous qui est seul à connaître le nombre. Nous n'avons pas le droit de poser de question, nous ne pouvons faire que des propositions auxquelles la personne qui a choisi le nombre répondra par : « c'est trop grand » ou par : « c'est trop petit ».
Le jeu consiste à trouver le nombre avec le moins de propositions possibles.

Comment écrire un programme qui permettra de remplacer la personne qui connait le nombre à deviner ?

Etape 1 : Nous élaborons un algorithme (voir précédemment)

Etape 2 : Nous rédigeons le programme (ou le code) dans un langage évolué (« haut niveau »). Le langage choisi au lycée est Python. Une fois le programme vérifié, nous lançons son exécution.

Etape 3 : lorsque notre programme s'exécute, il est converti (nous pouvons plutôt dire « compilé » ou « interprété ») afin d'être compréhensible par la machine, qui, on le rappelle, ne comprend que deux valeurs, 0 et 1, mais qui pourra traiter rapidement des combinaisons, même très complexes, de 0 et de 1 (c'est le langage machine).
Remarque : le processus qui vient d'être décrit passe en fait par un langage intermédiaire, l'assembleur, qui se présente sous la forme d'un ensemble d'instructions très simples. Une instruction dans un programme rédigé dans un langage de haut niveau (Python) peut être interprétée sous la forme de plusieurs instructions basiques en langage assembleur.

L'autre intérêt d'un langage haut niveau est que les instructions qu'il contient ont du sens pour nous (si nous prenons un peu d'habitude et si nous avons quelques bases en anglais) et leur lecture renverra clairement à l'algorithme initialement élaboré pour envisager la résolution du problème.

Sans préalablement se lancer dans un cours détaillé sur le langage Python…

Sans passer par la présentation détaillée d'un algorithme mais toutefois en élaborant un brouillon d'algorithme suite à une série d'échanges oraux entre les élèves et le professeur…

Voici directement le programme proposé pour le jeu du nombre à deviner :

```
from random import *
x = randint(0,100)
e = int(input("Choisis un nombre entre 0 et 100 :"))
i = 1
if e = = x :
   print("Bravo ! Gagné du premier coup !")
else :
   while e != x :
      if e < x :
         print("Trop petit")
      else :
         print("trop grand")
      e = int(input("essaie encore..."))
      i = i + 1
print("Gagné en", i, "essais")
```

Nous allons tacher de commenter tout ce que nous trouvons dans ce programme.

Chaque élève ouvre son environnement Python (à priori Spyder à partir du navigateur Anaconda)
Les différentes parties de la fenêtre Spyder sont décrites :

- *Zone de code*
- *Console d'exécution*
- *Zone d'affichages divers (aide, graphes résultant de l'exécution, etc.)*

Il écrit le code au fur et à mesure du commentaire proposé.
Le code doit être enregistré (clé en début d'année, espace perso dossier SNT par la suite)

*« from random import **
x= randint(0,100) » :

« Random » est un module, c'est-à-dire un ensemble de programmes permettant un grand nombre d'applications dans le domaine du l'aléatoire (« random » signifie « hasard »).
*L'instruction « import * » signifie que nous importons dans notre programme toutes les fonctions du module Random, ce qui n'était peut-être pas indispensable.*

Nous utilisons donc la fonction randint qui randint (qui est en fait un autre programme) permettant de sélectionner au hasard un entier. (0,100) délimite le domaine de valeurs entières qui peuvent être choisies)
*Au lieu de « * », nous aurions donc pu choisir d'importer seulement randint.*
Au lieu d'importer le module random, nous aurions pu nous même concevoir une série d'instructions correspondant au choix aléatoire d'un nombre entier entre 0 et 100.

Citer d'autres modules : maths, turtle,, tkinter

Le caractère « = »
Il n'a pas le sens de l'égalité mathématique.
x est une variable à laquelle nous allons affecter (« = ») un nombre entier entre 0 et 100 (« (0,100) » choisi au hasard (« randint », « int » (integer) pour entier)
Dans l'ordinateur, un emplacement est maintenant réservé pour la variable x.
Nous n'avons pas besoin de préciser à l'avance que le type de la variable x est le type nombre entier. Le fait de lui affecter une valeur entière impose par défaut que x sera codée en tant qu'entier.

Mêmes constations pour i (« i=1 »)

Nous pouvons rajouter type(x) ou type(i) dans le programme pour constater la réponse : ***int***

L'utilisateur peut modifier le code en choisissant lui-même la valeur entière à deviner, auquel cas l'importation des modules de la bibliothèque random est inutile :

Nous pouvons remplacer

```
from random import *
x = randint(0,100)
```

par

```
x = 48    (par exemple)
```

```
e = int(input("Choisis un nombre entre 0 et 100 :"))
```

Variable e à laquelle on affecte :

- *Un entier (« int »)*
- *Qui va devoir être entré par le joueur dans la zone d'exécution (« input »)*
- *Après affichage de la chaine de caractère entre guillemets*

Cela montre qu'un programme ne s'exécute pas forcément tout seul, il peut y avoir intervention extérieure, entrée de données en cours d'exécution.

« If e == x » : si le contenu de la variable e est le même que celui de la variable x.
Nous avons bien noté la syntaxe de l'égalité mathématique : « == »
Nous avons affaire à une instruction conditionnelle, elle est suivi obligatoirement de « : » et les instructions suivantes sont dites indentées, décalées vers la droite de 4 espaces.

```
print("Bravo ! Gagné du premier coup !")
```
: pas de commentaire (« print » veut dire afficher, présenter, etc.)

```
else
```

- *Signifie « sinon » dans le cas ou la condition précédente n'est pas remplie (« if… »)*
- *Se trouve à égalité de niveau d'instruction avec if…, donc pas d'indentation.*
- *Est suivi de « : », donc indentations pour la série d'instruction suivantes*

```
while e != x :
```

Une autre instruction conditionnelle qui signifie « tant que » (« while ») e est différent (syntaxe « != ») de x
Suivie de « : » et indentation aux lignes suivantes. La série d'instructions constitue une boucle qui va se répéter tant que e est différent de x

```
    if e < x :
        print("Trop petit")
    else :
        print("trop grand")
    e = int(input("essaie encore..."))
```
Pas d'aide pour ces instructions qui sont assez claires.

```
i = i+1
```
La variable i est augmentée d'une unité (cela compte le nombre d'essais). Pour cette même instruction on trouve aussi la syntaxe : « i+ = i »

Fin de la boucle « while » puisque l'instruction suivante n'est pas indentée. C'est aussi la fin de l'instruction conditionnelle « if…/ else… » puisque l'instruction finale n'est pas du tout indentée.

```
print("Gagné en", i, "essais")
```

Affichage du score. On note que l'on affiche en fait trois choses à la suite les unes des autres (séparées par des virgules au sein des parenthèses associées à l'instruction « print » :

- *Une chaine de caractère car présentée entre guillemets.*
- *La valeur de la variable i.*
- *Une autre chaine de caractères.*