

# Informatique 1ère année

Collège Sismondi

2025-2026



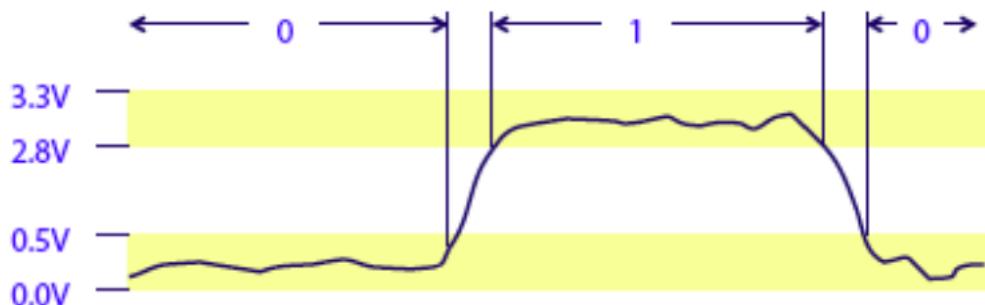
# REPRÉSENTATION NUMÉRIQUE DE L'INFORMATION

## 1.1 Introduction

Lorsque nous utilisons un logiciel, chaque action que nous effectuons avec la souris ou le clavier par exemple est traduite en langage informatique puis exécutée par l'ordinateur.

*“Transmettre des informations sous forme numérique suppose entre autres d’optimiser la taille des messages transmis pour éviter de surcharger les canaux de transmission, d’être capable de rectifier des erreurs apparues en cours de transmission, de crypter les contenus et d’authentifier les émissaires et les destinataires...”* (Dunod, 2006)

Les images, le son, le texte et les vidéos sont des informations qu'un ordinateur peut traiter. Un ordinateur est composé de circuits électriques qui fonctionnent à l'électricité. L'information est représentée physiquement par un **signal électrique ou magnétique qui, au-delà d'un certain seuil, correspond à la valeur 1 si non par 0**. Par conséquent, l'information est toujours sous forme d'un ensemble de nombres écrit en **base 2** par exemple **011001**.



En 1939, Claude Shannon a été le premier à faire un parallèle entre l'algèbre booléenne (une algèbre binaire, n'acceptant que vrai et faux comme valeurs, et trois fonctions logiques “Et (\*), Ou(+), Non(-)”) et le fonctionnement des circuits électriques. Le vrai/faux se transforme en 1 : le courant passe, 0 : il ne passe pas. C'est Shannon qui popularise le terme de bit :

### Définition 1 :

Le terme **bit** (b minuscule dans les notations) signifie *binary digit*, c'est-à-dire 0 ou 1 en

numérotation binaire. Il s'agit de la plus petite unité d'information manipulable par une machine numérique.

L'**octet** (en anglais **byte** ou **B** majuscule dans les notations) est une unité d'information composée de 8 bits. Il permet par exemple de stocker un caractère comme une lettre ou un chiffre.

D'autres termes sont aussi utilisés pour définir certaines longueurs de nombre :

- Une unité d'information composée de 16 bits est généralement appelée **mot** (en anglais **word**).
- Une unité d'information de 32 bits de longueur est appelée **mot double** (en anglais **double word**, d'où l'appellation **dword**).

Jusqu'en 1998 ; 1024 octets valaient 1 kilooctet. Et depuis les nouvelles unités standardisées par l'organisme international IEC sont les suivantes :

- Un kilooctet (**ko**) =  $10^3$  octets
- Un mégaoctet (**Mo**) =  $10^6$  octets
- Un gigaoctet (**Go**) =  $10^9$  octets
- Un téraoctet (**To**) =  $10^{12}$  octets
- Un pétaoctet (**Po**) =  $10^{15}$  octets
- Un exaoctet (**Eo**) =  $10^{18}$  octets
- Un zettaoctet (**Zo**) =  $10^{21}$  octets
- Un yottaoctet (**Yo**) =  $10^{24}$  octets

#### **Remarque 1 :**

Un mégaoctet devrait en principe valoir  $1000 \times 1000$  octets, c'est-à-dire 1'000'000 d'octets, mais il vaut  $1024 \times 1024$  octets en informatique, c'est-à-dire 1'048'576 octets... ce qui correspond à une différence de 4.63 % !

## 1.2 Les bases décimales, binaires et hexadécimales

### 1.2.1 La base décimale

Le système décimal (base 10) est celui que nous utilisons dans la vie quotidienne parce que nous avons commencé à compter avec nos doigts. Il comporte 10 symboles de 0 à 9. C'est un système positionnel, c'est-à-dire que l'endroit où se trouve le symbole définit sa valeur.

Par exemple :  $(9'875)_{(10)} = 9 \cdot 10^3 + 8 \cdot 10^2 + 7 \cdot 10^1 + 5 \cdot 10^0$

10 représente la base et les puissances de 0 à 3 le rang de chaque chiffre. Quelle que soit la base, le chiffre de droite est celui des unités. Celui de gauche est celui qui a le poids le plus élevé.

### 1.2.2 Binaire

Dans les domaines de l'automatisme, de l'électronique et de l'informatique, nous utilisons la base 2. Tous les nombres s'écrivent avec deux chiffres uniquement, 0 et 1. Car l'algèbre booléenne est à la base de l'électronique numérique. Par exemple, un interrupteur est ouvert ou fermé, une diode électroluminescente (ou LED en anglais) est allumée ou éteinte, une tension est présente ou absente, un champ magnétique est orienté Nord-Sud ou Sud-Nord.

Le chiffre binaire, qui peut prendre ces deux états, est nommé **Bit** (Binary digit). Les règles sont les mêmes que pour le décimal.

Par exemple,  $1101_{(2)} = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 13_{(10)}$

### Conversion binaire décimale

Une autre façon d'obtenir le nombre en base 10, connaissant son écriture en base 2, est d'établir la valeur de chaque chiffre du nombre : celui le plus à droite représente 1, le deuxième représente 2, le troisième représente  $2^2 = 4$ , le quatrième représente  $2^3 = 8$ , etc. comme dans la grille ci-dessous :

64	32	16	8	4	2	1
----	----	----	---	---	---	---

Par exemple si nous voulons connaitre la valeur décimale du nombre 10110, nous établissons d'abord une grille avec les différentes puissances de 2 ; ensuite, nous plaçons le nombre en binaire sous cette grille en mettant bien le chiffre des unités sous le carré le plus à droite et nous barrons les cases dont le chiffre associé est 0 :

64	32	16	8	4	2	1
		1	0	1	1	0

Il ne reste plus qu'à additionner les nombres qui restent :  $16 + 4 + 2 = 22$

1 Écrire en base 10 les nombres suivants :

- |                  |                   |
|------------------|-------------------|
| 1. $101_{(2)}$   | 4. $10101_{(2)}$  |
| 2. $10000_{(2)}$ | 5. $10_{(2)}$     |
| 3. $1111_{(2)}$  | 6. $111000_{(2)}$ |

Vous pouvez utiliser le tableau ci-dessous :

		64	32	16	8	4	2	1

### Conversion décimale binaire

Pour trouver l'écriture binaire d'un nombre écrit en base 10, il existe principalement deux méthodes.

La première méthode consiste à décomposer le nombre en somme de puissance de 2 et à utiliser la grille précédente.

Plus concrètement, si nous voulons écrire 99 en binaire. La plus grande puissance de 2 que nous pouvons prendre dans 99 est 64 (128, la suivante, est trop grande). Il reste ensuite :

**99-64 = 35.** Dans 35 on peut mettre 32, et il restera

**35-33 = 3.** Dans 3 on ne peut ni mettre 16, ni mettre 8, ni mettre 4. On peut mettre 2 :

**3-2 = 1** Il reste 1. Dans 1 on peut mettre 1 :

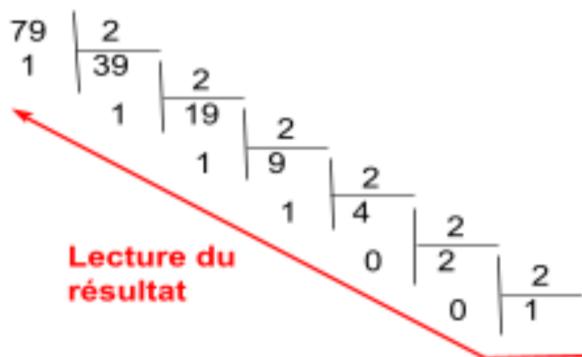
**1-1 = 0**

Cela signifie que  $99 = 64 + 32 + 2 + 1$ . Sur notre grille cela donne :

64	32	16	8	4	2	1
1	1	0	0	0	1	1

Donc  $99_{(10)} = 1100011_{(2)}$ .

La deuxième méthode consiste effectuer des divisions successives par 2. Le nombre en binaire se lira à l'aide des restes des différentes divisions :



Ainsi,  $79_{(10)} = 1001111_{(2)}$

**2** Les nombres suivants sont écrits en base 10. Donner leur écriture en base 2 :

- |        |         |
|--------|---------|
| 1. 75  | 5. 100  |
| 2. 12  | 6. 200  |
| 3. 27  | 7. 1000 |
| 4. 153 | 8. 2000 |

### 1.2.3 \*La base hexadécimale

C'est le code utilisé dans la programmation de certains automates et microprocesseurs. Il est composé de : 10 chiffres de 0 à 9, et 6 lettres de A à F. Les adresses MAC (adresses uniques associées à chaque carte réseau dans le monde) sont aussi écrites en base hexadécimale.

La manipulation des nombres écrits en binaire est difficile pour l'être humain et la conversion en décimal n'est pas simple. C'est pourquoi nous utilisons de préférence le système hexadécimal (base 16).

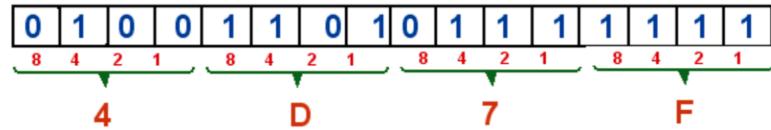
Le tableau ci-dessous montre la représentation des nombres de 0 à 15 dans les bases 10, 2 et 16.

Par exemple,  $B4F_{(16)} = B \cdot 16^2 + 4 \cdot 16^1 + F \cdot 16^0 = 11 \cdot 16^2 + 4 \cdot 16^1 + 15 \cdot 16^0 = 2895_{(16)}$

Décimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Binaire	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Hexadécimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

### Conversion en binaire

Pour convertir un nombre binaire en hexadécimal il suffit de remarquer que chaque groupe de 4 bits représente un chiffre en hexadécimal :



3

Écrire les nombres suivants en base hexadécimale :

1.  $10010010110_{(2)}$
2.  $111110_{(2)}$
3.  $1000110101110101_{(2)}$
4.  $11110000000011_{(2)}$

### Conversion en décimal

La méthode par division (par 16) s'applique comme en binaire (par 2).

4

Écrire les nombres suivants en hexadécimal :

1. 92
2. 256
3. 500
4. 1023

### 1.2.4 Opération sur les nombres en binaires

Tout comme pour l'addition en colonne, pour additionner deux nombres écrits en binaires il faut les additionner bit à bit.

5

Quel est le résultat des calculs binaires suivants :

- a)  $0 + 0 =$
- b)  $1 + 0 =$
- c)  $0 + 1 =$
- d)  $1 + 1 =$
- e)  $1 + 1 + 1 =$

À l'aide de l'exercice suivant effectué l'addition suivante :

$$1100101001 + 101110101$$

Pour cela, compléter l'addition en colonne suivante :

$$\begin{array}{r}
 1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \\
 + \ 1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \\
 \hline
 \end{array}$$

6

Effectuer les additions binaires suivantes :

- |                     |                      |
|---------------------|----------------------|
| 1. $1001 + 101$     | 3. $110001 + 11010$  |
| 2. $10011 + 110011$ | 4. $110101 + 101110$ |

7

1. On voudrait encoder des emojis en binaire ; en d'autres mots, on voudrait faire correspondre chaque emoji à un code binaire différent.
  - (a) Combien peut-on encoder de symboles (ici des emojis) différents sur 3 bits ?
  - (b) Sur 5 bits ?
  - (c) Sur 11 bits ?
2. Quel est le plus grand nombre qu'on peut écrire sur 5 bits ?
3. On se pose maintenant la question inverse :
  - (a) Combien me faut-il de bits pour encoder 8 symboles différents ?
  - (b) 9 symboles ?
  - (c) 69 symboles ?

## 1.3 Représentation des nombres entiers

Nous arrivons à écrire tous les nombres entiers naturels en binaire. Il faut maintenant utiliser un certain nombre de règles pour pouvoir représenter n'importe quel nombre avec un ordinateur : nombres entiers, nombres relatifs, nombres rationnels...

### 1.3.1 Représentation des nombres entiers naturels

*Rappel : Les nombres entiers naturels sont l'ensemble des nombres entiers et positifs. Il est désigné par la lettre  $\mathbf{N} = \{0, 1, 2, \dots\}$ .*

Nous pouvons par exemple décider que les nombres seront codés sur un octet (8 bits). Le plus petit nombre sera  $00000000 = 0$  et le plus grand sera  $11111111 = 255$ .

**Problème :** En faisant ainsi, nous ne pouvons pas représenter des nombres négatifs.

### 1.3.2 Représentation des nombres entiers relatifs signés

*Rappel : Les nombres entiers sont l'ensemble des nombres entiers négatifs et positifs. Il est désigné par la lettre  $\mathbf{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$ .*

La première façon pour représenter les nombres négatifs est de décider que le bit le plus gauche représente le signe du nombre.

Sur 8 bits plus grand nombre sera  $01111111 = 127$  et le plus petit sera  $11111111 = -127$ .

$104 = \boxed{0} \boxed{1} \boxed{1} \boxed{0} \boxed{1} \boxed{0} \boxed{0} \boxed{0}$

$-104 = \boxed{1} \boxed{1} \boxed{1} \boxed{0} \boxed{1} \boxed{0} \boxed{0} \boxed{0}$

 signe       magnitude

$$\begin{array}{r}
 104 \quad \boxed{0} \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \\
 + \quad 12 \quad \boxed{0} \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \\
 \hline
 116 \quad \boxed{0} \quad 1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0
 \end{array}$$

$$\begin{array}{r}
 104 \quad \boxed{0} \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \\
 + \quad 30 \quad \boxed{0} \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \\
 \hline
 \text{Résultat faux} \quad \boxed{1} \quad 1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0
 \end{array}$$

Il est possible d'additionner deux nombres pour autant qu'ils soient tous les deux positifs et que le résultat ne dépasse pas le nombre plus grand nombre pouvant être écrit :

Le résultat de  $104 + 30 = 134$  amène un dépassement de capacité. Le nombre entier signé maximum sur 8 bits est 127.

**Problème :** Bien que pratique, cette façon de coder les nombres négatifs présente deux problèmes :

1. zéro est représenté de deux façons différentes : 1000000 et 00000000.
2. si on additionne deux nombres opposés, nous n'obtenons pas 0.

### 1.3.3 \* Représentation des nombres en complément à deux

Cette représentation résout les problèmes de la représentation signée. Dans cette représentation, pour obtenir l'opposé d'un nombre positif écrit en binaire, nous allons effectuer les deux étapes suivantes :

1. On inverse tous les bits du nombre positif (on change les 0 en 1 et inversement). Cela s'appelle le **complément à 1**.
2. On ajoute 1.

Par exemple, nous aimerais écrire le nombre -80.

Premièrement, 80 s'écrit en binaire : **01010000**.

On prend le complément à 1 de ce nombre, ce qui donne : **10101111**

Puis on ajoute 1 :

$$\begin{array}{ccccccccc}
 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\
 + & & & & & & & & 1 \\
 \hline
 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0
 \end{array}$$

L'avantage de cette méthode est que la somme deux nombres opposés donnera bien 0.

En supposant que les nombres soient écrits sur 8 bits, vérifier que  $80 + (-80) = 0$ .

### 1.3.4 Résumé

n=4	non signé	signe-magnitude	complément à 2
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	8	0	-8
1001	9	-1	-7
1010	10	-2	-6
1011	11	-3	-5
1100	12	-4	-4
1101	13	-5	-3
1110	14	-6	-2
1111	15	-7	-1

**9** Donner la représentation signée sur un octet des nombres suivants :

- |         |         |
|---------|---------|
| 1. -100 | 3. -200 |
| 2. 38   | 4. -83  |

### \* Complément à deux

a) Donner la représentation en complément à deux des nombres suivants :

- (a) -95  
 (b) -64

b) Vérifier que  $95 + (-95)$  donne bien 0 sur un octet.

c) Vérifier que  $64 + (-6)$  donne bien 0111010 sur un octet.

Nous donnons trois nombres binaires :

11

- a) 00101010
- b) 1011
- c) 10111111

1. Que valent ces nombres en représentation non signée ?
2. Que valent ces nombres en représentation signée ?
3. \* Que valent ces nombres en complément à 2 ?

## 1.4 Les codes de caractères

### 1.4.1 La table ASCII

**Decimal - Binary - Octal - Hex – ASCII  
Conversion Chart**

Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	Space	64	01000000	100	40	@	96	01100000	140	60	`
1	00000001	001	01	SOH	33	00100001	041	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	002	02	STX	34	00100010	042	22	"	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	003	03	ETX	35	00100011	043	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	004	04	EOT	36	00100100	044	24	\$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	005	05	ENQ	37	00100101	045	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	006	06	ACK	38	00100110	046	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	007	07	BEL	39	00100111	047	27	'	71	01000111	107	47	G	103	01100111	147	67	g
8	00001000	010	08	Backspace	40	00101000	050	28	(	72	01001000	110	48	H	104	01101000	150	68	h
9	00001001	011	09	HT	41	00101001	051	29	)	73	01001001	111	49	I	105	01101001	151	69	i
10	00001010	012	0A	LF	42	00101010	052	2A	*	74	01001010	112	4A	J	106	01101010	152	6A	j
11	00001011	013	0B	VT	43	00101011	053	2B	+	75	01001011	113	4B	K	107	01101011	153	6B	k
12	00001100	014	0C	FF	44	00101100	054	2C	,	76	01001100	114	4C	L	108	01101100	154	6C	l
13	00001101	015	0D	CR	45	00101101	055	2D	-	77	01001101	115	4D	M	109	01101101	155	6D	m
14	00001110	016	0E	SO	46	00101110	056	2E	.	78	01001110	116	4E	N	110	01101110	156	6E	n
15	00001111	017	0F	SI	47	00101111	057	2F	/	79	01001111	117	4F	O	111	01101111	157	6F	o
16	00010000	020	10	DLE	48	00110000	060	30	0	80	01010000	120	50	P	112	01110000	160	70	p
17	00010001	021	11	DC1	49	00110001	061	31	1	81	01010001	121	51	Q	113	01110001	161	71	q
18	00010010	022	12	DC2	50	00110010	062	32	2	82	01010010	122	52	R	114	01110010	162	72	r
19	00010011	023	13	DC3	51	00110011	063	33	3	83	01010011	123	53	S	115	01110011	163	73	s
20	00010100	024	14	DC4	52	00110100	064	34	4	84	01010100	124	54	T	116	01110100	164	74	t
21	00010101	025	15	NAK	53	00110101	065	35	5	85	01010101	125	55	U	117	01110101	165	75	u
22	00010110	026	16	SYN	54	00110110	066	36	6	86	01010110	126	56	V	118	01110110	166	76	v
23	00010111	027	17	ETB	55	00110111	067	37	7	87	01010111	127	57	W	119	01110111	167	77	w
24	00011000	030	18	CAN	56	00111000	070	38	8	88	01011000	130	58	X	120	01111000	170	78	x
25	00011001	031	19	EM	57	00111001	071	39	9	89	01011001	131	59	Y	121	01111001	171	79	y
26	00011010	032	1A	SUB	58	00111010	072	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A	z
27	00011011	033	1B	ESC	59	00111011	073	3B	:	91	01011011	133	5B	[	123	01111011	173	7B	{
28	00011100	034	1C	FS	60	00111100	074	3C	<	92	01011100	134	5C	\	124	01111100	174	7C	
29	00011101	035	1D	GS	61	00111101	075	3D	=	93	01011101	135	5D	]	125	01111101	175	7D	}
30	00011110	036	1E	RS	62	00111110	076	3E	>	94	01011110	136	5E	^	126	01111110	176	7E	~
31	00011111	037	1F	US	63	00111111	077	3F	?	95	01011111	137	5F	_	127	01111111	177	7F	DEL

Le binaire permet de coder les nombres que les systèmes informatiques peuvent manipuler. Cependant, l'ordinateur doit aussi utiliser des caractères alphanumériques pour mémoriser et transmettre des textes par exemple de l'ordinateur vers l'imprimante, d'un automate programmable vers un terminal, d'un clavier vers un processeur, etc.

Le code **ASCII** (American Standard Code for Information Interchange) représente les caractères sur 7 bits (c'est-à-dire 128 caractères possibles, de 0 à 127). Les codes 0 à 31 ne sont pas des caractères. On les appelle caractères de contrôle car ils permettent de faire des actions telles que : retour à la ligne (CR). Bip sonore (BEL). Les codes 65 à 90 représentent les majuscules.

Les codes 97 à 122 représentent les minuscules. Le caractère A par exemple à pour code 65 soit 01000001 en binaire.

Par exemple :

- a) Le caractère f : 102
- b) le point d'interrogation ? : 63
- c) Le chiffre 2 : 50

Le code ASCII a été mis au point pour la langue anglaise, il ne contient donc pas de caractères accentués, ni de caractères spécifiques à une langue. Le code ASCII a donc été étendu à 8 bits pour pouvoir coder plus de caractères (on parle d'ailleurs de code ASCII étendu...)

### 1.4.2 Unicode

Le code ASCII est utilisable pour l'anglais mais limité pour les autres langues. Il n'y a que 95 caractères imprimables.

Le code Unicode est un système de codage des caractères sur 16 bits mis au point en 1991. Le système Unicode permet de représenter n'importe quel caractère par un code sur 16 bits, indépendamment de tout système d'exploitation ou langage de programmation (environ 64 000 caractères).

Il regroupe ainsi la quasi-totalité des alphabets existants (arabe, arménien, cyrillique, grec, hébreu, latin, ...) et est compatible avec le code ASCII.

L'ensemble des codes Unicode est disponible sur le site <http://www.unicode.org>.

### 1.4.3 Exercices

12

Traduire en code ASCII binaire les textes :

1. Sismondi
2. Cc cv ?

13

Traduire en lettres les mots suivants qui sont codés en ASCII (binaire) :

1. 01100011 01101111 01110101 01100011 01101111 01110101
2. 01001000 01100101 01101100 01101100 01101111 00100000 01110111 01101111 01110010  
01101100 01100100 00100000 00100001

14

\* En s'aidant de la table ASCII, classer par ordre croissant les caractères suivants (ne pas tenir compte des virgules) : ESC, A, NUL, Delete, m, 8, <, a, ?

15

Complétez le tableau ci-dessous :

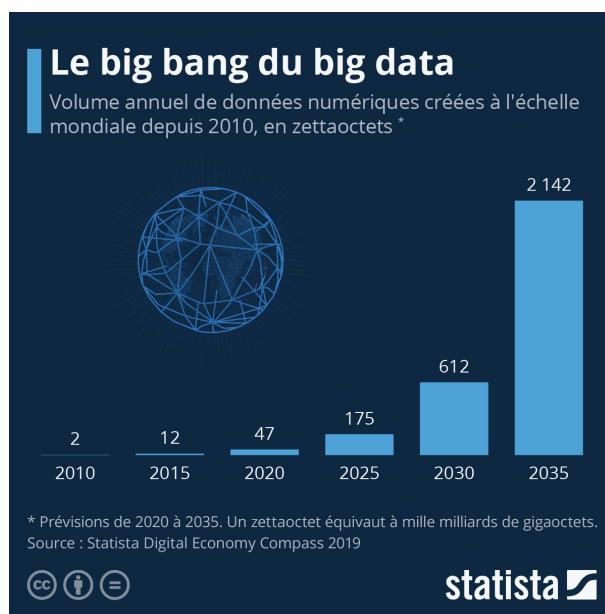
Nombres	Bases			
	2	10	16	ascii
11001110101 <sub>2</sub>				
23670 <sub>10</sub>				
FA2C5 <sub>16</sub>				

## CHAPITRE 4

# STOCKAGE DE DONNÉES

### 4.1 Introduction

Les besoins en stockage de données numériques, à l'échelle mondiale, ont été multiplié par plus de vingt au cours de la dernière décennie et devraient dépasser les 50 zettaoctets (10<sup>21</sup>) d'ici fin 2021. Comme le montre l'infographie ci-dessous, cette quantité de données apparaît finalement dérisoire en comparaison avec ce qui est attendu pour les quinze prochaines années. Les prévisions tablent en effet sur une multiplication par trois ou quatre du volume annuel de données créées tous les cinq ans. Avec ce rythme exponentiel de croissance, le seuil astronomique des 2'000 zettaoctets devrait être franchi à l'horizon 2035.



Il faudrait se procurer 500 millions de disques durs actuels (100 To) pour être capable de sauvegarder 50 [Zo]! Cette quantité de données pose autant de problèmes de préservation et d'intégrité des données que de sauvegarde. Sans annoncer les problèmes de places, de consommation électrique, de pollutions dues à la fabrication et au recyclage du matériel vieillissant, de coût et de sauvegarde.

## 4.2 Octet : unités de mesure en informatique

Rappel : un octet permet de représenter  $2^8$  nombres, soit 256 valeurs différentes.

Symbol	ko	Mo	Go	To	Po	Eo	Zo	Yo
Nom	kilooctet	mégaoctet	gigaoctet	téraoctet	pétaoctet	exaoctet	zettaoctet	yottaocet
Valeur	$10^3$	$10^6$	$10^9$	$10^{12}$	$10^{15}$	$10^{18}$	$10^{21}$	$10^{24}$

Exemples de tailles de fichier (ordre de grandeur) :

- 2 Ko : fichier texte (2'000 lignes de texte)
- 1 à 20 Mo : photo compressée (jpeg) d'un appareil photo numérique.
- 10 Mo : fichier MP3 d'une durée 5 min (débit 320 kbit/s)
- 700 Mo : film 1h30 (compression – qualité DVD)
- 3 Go : film 1h30 (compression – qualité Blu-ray)
- 25 Go : film Blu-ray

## 4.3 Support de données

Quel que soit le type de support, sa taille et son emplacement (local ou distant), ils ont tous certains points communs, comme :

1. Les supports contiennent des 1 et des 0 qu'il s'agisse de votre dernier livre ou de votre musique préférée.
2. Tous les supports ont un schéma d'organisation des données qui est dépendante du système d'exploitation.
3. Ils ont tous besoin d'être alimentés électriquement pour fonctionner.
4. Aucun support n'est fiable à 100 % !

### 4.3.1 Supports amovibles

Avant d'avoir les clés USB et les cartes mémoires que l'on retrouve dans les téléphones mobiles et les appareils photo, il existait d'autres supports externes. Parmi les premiers supports de données informatiques grand public se trouvait la disquette.



Inventée dans les années 1970, elle remplaçait les cartes perforées. Quelques exemples de supports amovibles :

1970 : disquette 8" : 1,2 Mo  
 1970 : disquette 5,25" 360k  
 1980 : disquette 3,5" : 1,44 Mo, le standard pendant 20 ans !  
 1990 : CD-ROM 700Mo (d'autre déclinaison jusqu'à 800Mo)  
 1990 : ZIP : 700Mo  
 2004 : DVD-ROM : 4,7 Go  
 2004 : DVD-ROM double couche : 9,4 Go  
 2006 : Blu-ray 25Go  
 2006 : Blu-ray double couche 50Go  
 2000 : clé USB et carte mémoire

#### 4.3.2 Supports distants

Depuis 2010, plusieurs entreprises offrent un service de stockage et de partage de fichiers dans le cloud (ou nuage). Il est donc nécessaire d'avoir une connexion Internet pour accéder à ces espaces de stockage.

##### Remarques 1 :

Le terme « cloud » est une forme abrégée de « cloud computing » ou l'informatique en nuage. Un cloud est constitué de serveurs situés à distance et accessibles de n'importe où et à n'importe quel moment via une connexion Internet sécurisée et protégée.



Quels sont les avantages et les inconvénients d'avoir ses données dans un cloud qui ne vous appartient pas (dont vous ne gérez pas l'infrastructure) ?

**Exemple 1.** Vous prenez un contrat chez l'entreprise Tartempion 5 chf par mois pour 1 To de place pour vos données.

##### Avantages principaux du cloud :

- + Vos données sont accessibles partout, à condition d'être connecté à Internet. Vos données sont donc accessibles depuis votre smartphone, votre PC, etc.
- + Possibilité de partager ses données. De plus, certains logiciels permettent la modification simultanée et gèrent l'accès concurrent.
- + Aucun investissement ni installation préalable requise, juste le prix de l'abonnement. Il vous suffit en général d'un navigateur web pour accéder à vos données.

- + Maintenance, sécurisation des données et mises à jour effectuées par le fournisseur.
- + Souplesse dans l'espace loué. Si vous avez besoin de 2 To, vous pourrez évoluer votre abonnement.
- + En cas de panne de votre ordinateur personnel, vous ne perdez pas les données qui sont dans le cloud.

### Inconvénients principaux du cloud :

- Pas de contrôle total d'accès aux données par rapport à l'entreprise Tartempion. Confidentialité des données compromise si aucun outil n'est mis en place pour protéger les documents sensibles ;
- Savez-vous où sont réellement hébergées vos données ?
- Dépendance avec l'entreprise. Si Tartempion fait faillite que se passe-t-il ?
- Risque de cyberattaques si les données ne sont pas bien protégées.
- Obligation d'être connecté à Internet. Le service peut être dégradé si la liaison n'est pas fiable.
- Accès aux fichiers plus lents que sur un support local.
- Impact écologique : par ses serveurs nécessaires au fonctionnement du cloud, ce dernier induit une consommation d'énergie croissante.

## 4.4 Stratégie de sauvegarde

### 4.4.1 Introduction

Aucun support de stockage n'a une durée de vie illimitée et aucun support de stockage n'est fiable à 100%. Le risque zéro n'existe pas. Une perte de données numériques est donc très vite arrivée. Les causes principales sont :

- o des défaillances techniques matérielles comme un disque dur en panne ou une clé usb qui n'est plus accessible ;
- o des défaillances logicielles comme un fichier corrompu suite à une écriture erronée par le système d'exploitation, une erreur de synchronisation ;
- o des défaillances humaines comme un fichier effacé, un fichier écrasé ou une clé usb perdue ;
- o des catastrophes naturelles comme des incendies ou des dégâts des eaux ;
- o des programmes malveillants et des attaques de virus ou de ransomware ;
- o le vol du matériel comme votre téléphone portable, un disque d'un serveur, une clé usb, une carte mémoire ;

Le monde professionnel indique une durée de vie moyenne de 5 ans pour un disque dur mécanique ou un disque type SSD. Une clé usb ou une carte mémoire de qualité pourrait tenir 10 ans. Le cloud offert par les entreprises de stockage a une durée de vie quasi illimitée, du moins celle de l'entreprise.

### 4.4.2 Stratégie 3-2-1 de sauvegarde de données

La stratégie de sauvegarde 3-2-1 représente le b-a-ba et le minimum de la protection des données. Mise au point par un photographe souhaitant protéger ses clichés dans les années 1920, cette stratégie est devenue une référence, car elle permet une protection optimale des données, quel que soit leur format.

Le concept de base de la stratégie 3-2-1 repose sur trois principes :

- > **3 copies de vos données** : La stratégie recommande d'avoir 2 copies plus la source. Avec trois copies dont deux sauvegardes, le risque que les trois copies aient un problème en même temps est très faible surtout si elles sont stockées sur des supports différents.
- > **2 supports différents** : La notion de supports différents est essentielle quand on parle de données numériques, car quel est l'intérêt d'avoir deux sauvegardes si elles sont stockées sur le même support ? En cas d'incident, toutes les données seraient perdues. Il est donc important que la source et au moins une de ses copies soient sauvegardées sur des supports différents.
- > **1 copie hors site** : Quel que soit le support (NAS, disque dur externe, lecteur de bande, etc.), aucune stratégie de sauvegarde de données numériques ne peut être considérée comme sûre si au moins une des copies n'est pas stockée « hors site ». Le récent incendie du 10 mars 2021 d'OVH en est la preuve. En effet, certains utilisateurs avaient leur site en production sur un serveur et leurs copies sur un autre serveur. Certes, les copies étaient sur des supports différents, mais elles étaient stockées dans le même datacenter. Comme une grande partie du site contenant ces serveurs a cessé de fonctionner à cause de l'incendie, aucune des copies n'était utilisable. Or, si au moins une de ces copies avait été stockée chez un autre hébergeur (hors site), ces utilisateurs auraient pu rétablir rapidement leurs services.

Malgré tout les stratégies de sauvegarde ont leurs limites ! La quantité de données à sauvegarder peut prendre trop de temps et poser des problèmes d'intégrité des données. L'externalisation des sauvegardes selon la méthode 3-2-1 soulève aussi la question de la protection des données.

Et si vous avez bien conservé votre donnée numérique. Serez-vous capable de la relire ?

## 4.5 Exercices

**1** En réfléchissant aux derniers cours, essayez de calculez la taille que ferait un fichier texte **blabla.txt** contentant la chaîne de caractères **blablabla**. Vérifiez ensuite en le créant sur votre ordinateur.

**2** Reprenez le dossier **Nourriture** de l'exercice 2 du chapitre précédent. Compressez-le et comparez sa taille originale avec sa taille une fois compressé.

**3** Téléchargez l'image **sample\_1920x1280.bmp** en haute définition depuis moodle puis convertissez-la dans les formats suivants en comparant la taille et la qualité de l'image à chaque étape.

*Utilisez par exemple le site <https://online-converting.com/image> pour faire la conversion en jpg et <https://online-converting.com/image/convert2bmp> pour le bmp*

Format	Taille (ko)	Qualité visuelle
.bmp en 24 bits		
.bmp en 8 bits		
.jpg		
.png		



# PROGRAMMATION - LES BOOLÉENS

## 8.1 Vrai ou faux ?

Comme étudié dans le cas des test, un programme se doit de prendre des décisions. Dans notre vie de tous les jours, nos décisions dépendent le plus souvent des questions qui se répondent par *oui* ou par *non*. Par exemple, "est-ce qu'il pleut aujourd'hui" ou "êtes-vous mineur". La plupart des langages de programmation comme Python ne possèdent pas de type oui/non mais plutôt un type vrai/faux ( `True` / `False` ) qui joue un rôle similaire. Ce type s'appelle **booléen**. Dans ce contexte, les décisions du programme dépendent plutôt de la véracité d'une *affirmation*. Par exemple, au lieu de répondre à la question "êtes-vous mineur?", on s'intéressera à savoir si l'affirmation "vous êtes mineur" est vraie. Par exemple, en Python, cela se traduirait par

---

```
age = int(input("Quel est votre age"))
mineur = age < 18
```

---

Comme étudié dans le chapitre sur les tests, un programme est souvent confronté à devoir prendre des décisions.

### Définition 8.1

Une **booléen** est un type de variable à deux états, généralement noté *vrai* (ou 1) et *faux* (ou 0).

En langage `python`, le type d'une telle variable est `bool`, les deux valeurs possibles sont `True` ou `False`.

## 8.2 Opérateur de comparaison

Le plus souvent, un booléen est obtenu lorsque l'on évalue une expression qui compare des valeurs. Les opérateurs de comparaison sont les suivants

Opérateur	Expression	Signification
<code>==</code>	<code>x == y</code>	Égal
<code>!=</code>	<code>x != y</code>	Non égal
<code>&gt;</code>	<code>x &gt; y</code>	Plus grand que
<code>&lt;</code>	<code>x &lt; y</code>	Plus petit que
<code>&gt;=</code>	<code>x &gt;= y</code>	Plus grand ou égal à
<code>&lt;=</code>	<code>x &lt;= y</code>	Plus petit ou égal à
<code>is</code>	<code>x is y</code>	est identique
<code>is not</code>	<code>x is not y</code>	n'est pas identique

Il ne faut pas confondre la simple égalité "`=`" qui représente l'affectation de la double égalité "`==`" qui teste si deux valeurs sont égales. Par exemple, le programme



```
n = 2**4
m = 16
print(n == m)
```

affiche `True` car  $2^4$  est égal à 16.

### Exercice 1

Dans l'interpréteur Python, taper les lignes suivantes et compléter celles en pointillés.

```
>>> x=7
>>> y=17
>>> x==y
...
>>> x!=y
...
>>> x>y
...
>>> x>=y
...
>>> x<y
...
>>> x<=y
...
>>> x is y
...
>>> x is not y
...
```

### Exercice 2

Evaluer le résultat des expressions suivantes

```
3 == 1 + 2
1.0 == 1
"1" == 1
3 != 1 + 2
1.0 != 1
1 < 0
"c" <= "h"
"h" > "B"
1.0 is 1
1 + 2 is not 3
```



Si les expressions comparées ne sont pas du « même » type, une `TypeError` est générée (sauf pour l'opérateur `is`).

## 8.3 Opérations booléens

Mon réveil sonne le matin lorsque deux conditions sont satisfaites.

# REPRÉSENTATION NUMÉRIQUE DE L'INFORMATION - IMAGES

## 2.0.1 Image numérique

Grâce aux nombres, les ordinateurs stockent des dessins, des photos et d'autres types d'images. *Wikipedia* définit l'image numérique comme toute image (dessin, icône, photographie...) acquise, créée, traitée et stockée sous forme binaire.

On distingue deux types d'images numériques :

- L'image matricielle (ou bitmap) :** Elle est composée de points appelés pixels que l'on voit pas à l'œil nu. Lors de l'agrandissement l'image peut devenir floue car les pixels ressortent (les carrés sur l'écran).
- L'image vectorielle :** Elle est composée de lignes et de segments qui sont liés entre eux par des formules mathématiques. Grâce à la vectorisation, chaque élément a une place définie qui empêche la déformation de l'image. Au lieu de mémoriser une mosaïque de points élémentaires, on stocke la succession d'opérations conduisant au tracé.

L'usage de ce type d'image concerne les schémas générés par certains logiciels de DAO(Dessin assisté par ordinateur) ou pour les animations Flash utilisées sur internet pour la création de bannières publicitaires.

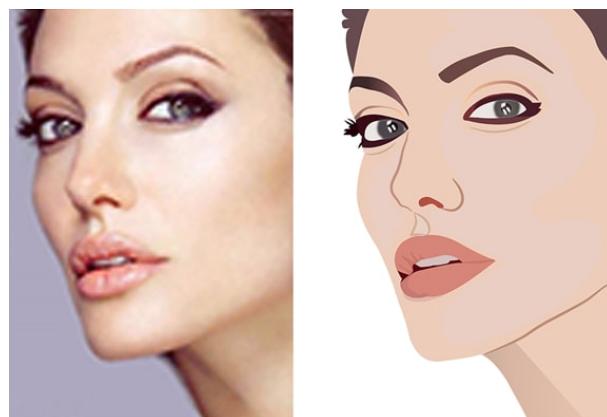


FIGURE 2.1 – Différence entre une image bitmap (à gauche) et vectorielle (à droite)

**Remarque 1 :**

Il n'est pas possible de tout vectoriser car les photos et les dégradés de couleurs ne se vectorisent pas. En effet, la vectorisation aplatis les couleurs et élimine des dégradés.

**2.0.2 Coder une image en noir et blanc****Première méthode****Définition 1 :**

Les écrans d'ordinateur sont divisés en une grille de petits points appelés pixels (picture elements, qui signifie éléments d'image)

Chaque pixel de l'image est codé soit par un 0, représentant un point noir ou un 1, représentant un point blanc. Donc un bit permet de coder un pixel d'une image en noir et blanc.

Pour stocker une image en noir et blanc, il suffit de convertir le noir par 0 et le blanc par 1.

1	1	1	0	0	0	1	1
1	1	0	0	0	0	1	1
1	1	1	1	0	1	1	1
1	1	1	0	0	0	1	1
1	1	0	0	1	0	0	1
1	1	0	0	1	0	0	1
1	1	1	0	0	0	1	1
1	1	0	0	1	0	0	1

**Deuxième méthode \***

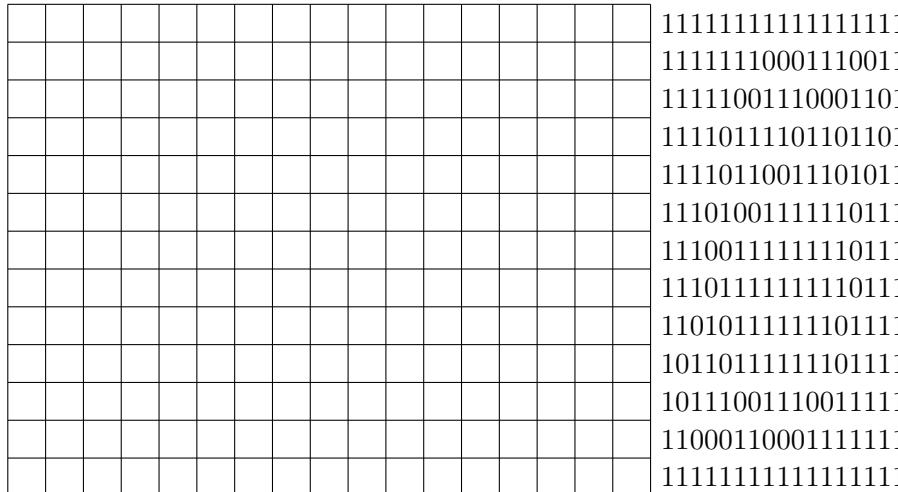
Une autre façon consiste à indiquer pour chaque ligne le nombre de points blancs et de points noirs consécutifs.

					1,3,1
					4,1
					1,4
					0,1,3,1
					0,1,3,1
					1,4

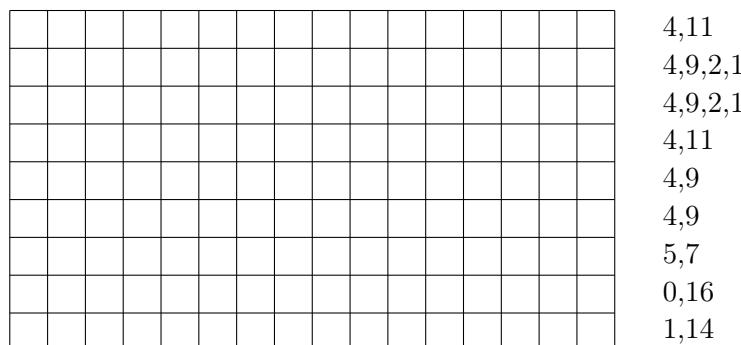
L'image de la lettre "a" nous montre comment une image peut être représentée par des nombres. Le premier nombre représente toujours le nombre de pixels blancs. Si le premier pixel est noir, la ligne commencera par un 0.

- La première ligne est représentée par 1, 3, 1.
- La première ligne contient 1 pixel blanc, 3 noirs puis 1 blanc.
- La quatrième ligne est représentée par 0, 1, 3, 1.

## 1 Décoder l'image suivante :



Décoder l'image suivante : \*



### 2.0.3 Coder une image en niveaux de gris

Plutôt que de simplement avoir 2 couleurs possibles, noir et blanc, on peut décider d'avoir N gris différents allant du plus sombre, le noir, au plus clair, le blanc.

Pour stocker une image en mémoire, il suffit de convenir que chaque point de l'image est représenté par un nombre.

Ce dernier correspond à une certaine couleur.

Pour une image en 16 niveaux de gris, il faut que chaque point de l'image soit codé par un nombre en 4 bits.

Si chaque point est codé sur 8 bits l'image peut contenir 256 niveaux de gris différents. Si l'on souhaite que l'image contient plus de couleurs, on pourra utiliser plus de bits pour chaque pixel.

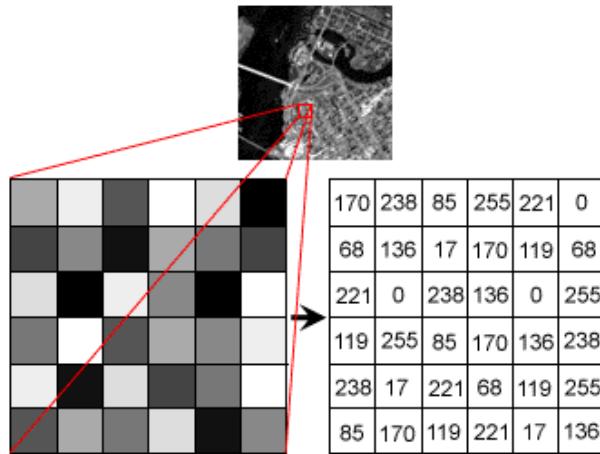


FIGURE 2.2 – Image en niveaux de gris

#### 2.0.4 Coder une image en couleur

Pour représenter des images avec des couleurs réalistes, on peut utiliser la même méthode que pour les niveaux de gris, mais en utilisant 3 nombres différents, 1 pour chaque couleur primaire.

Le code le plus utilisé est le codage **RVB** (Rouge, Vert, Bleu) (RGB en anglais).

Chacune de ces couleurs est codée sur 1 octet (8 bits). Avec ce système, chaque pixel est codé sur 3 octets pour un total de 24 bits.

La valeur décimale pour chaque couleur peut s'étendre de 0 à 255, cette limite est fixée par la valeur maximale qu'un octet peut représenter avec le système binaire.

L'avantage du système RVB est sa simplicité, il s'appuie sur les caractéristiques de la vision humaine pour fournir à l'utilisateur une reproduction des couleurs au plus proche de la réalité.

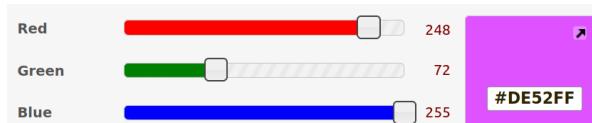


FIGURE 2.3 – Image RGB

codage pixel en décimal

Couleur	R	V	B
Gris moyen	127	127	127
Blanc	255	255	255
Rouge	255	0	0
Jaune	255	255	0
Vert	0	255	0
Cyan	0	255	255
Bleu	0	0	255
Magenta	255	0	255
Noir	0	0	0

Comme pour les systèmes de codage noir & blanc et en niveau de gris, le système RVB utilise une norme pour le codage des couleurs.

Le codage sur 16 bits permet d'obtenir 65536 couleurs différentes. Le codage sur 24 bits permet d'obtenir plus de 16 millions de couleurs ( $2^{24}$  couleurs).

On considère souvent qu'un codage sur 32 bits permet de coder plus de couleurs que l'œil humain peut en distinguer.

Une photo de haute qualité est codée sur 32 bits.

On peut se contenter de 24 bits (soit 3 octets). Chaque octet représente une nuance des trois couleurs (rouge, vert et bleu), qui sont mélangées.

## Compression

### Remarque 2 :

La taille (poids) de l'image augmente en augmentant le nombre de nuances de couleurs.

Une technique consiste à réduire l'information nécessaire pour représenter une image ou un son.

Pour pallier les inconvénients des images trop lourdes en particulier lors du stockage ou de la transmission et publication sur le web.

Ce qui permet de réduire les exigences quant à l'espace disponible sur le disque ainsi que les coûts de communications lorsque l'image ou le son sont téléchargés.

Malgré les techniques de compression, les images ou sons occupent encore souvent une place importante dans le programme.

## Les formats d'images

### Définition 2 :

Un format d'image est une représentation informatique de l'image, associée à des informations sur la façon dont l'image est codée et fournissant éventuellement des indications sur la manière de la décoder et de la manipuler.

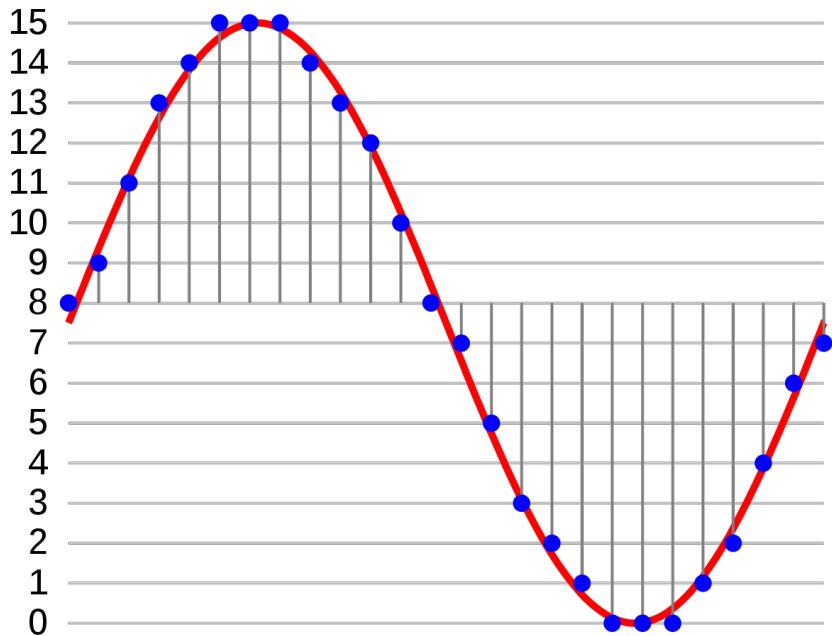
Voici quelques formats d'images très utilisés :

- a) **BMP** : (prononcé *bitmap*) pas de compression, profondeur de couleurs au choix
- b) **JPG ou JPEG** : (prononcé *j-peg*) compression possible ; qualité entre 0 et 100%. Perte de qualité. Profondeur de couleurs au choix. Bien adapté aux photos.
- c) **GIF** : compression sans perte de qualité. 256 couleurs max.
- d) **PNG** : compression sans perte. Intéressant pour les images avec peu de couleurs (schémas), mais pas pour des photos.

## 2.1 Représentation du son

### Échantillonage

Un son est un signal physique qui correspond à la pression de l'air au cours du temps. C'est ces variations de pressions que captent nos oreilles. Une fois enregistré, ce son peut-être encodé de manière similaire aux images : on sélectionne une valeur à intervalles de temps régulières et on encode chacune de ces valeurs en binaire. Cela revient à "pixeliser" le son.



Ce processus est nommé l'échantillonnage et on échantillonne en général les sons à 44 kHz (ce qui correspond à une valeur toutes les  $1/44000$  secondes, soit toutes les  $23 \mu s$ ), soit 2x plus que le son le plus aigu que l'oreille humaine peut entendre (22kHz).

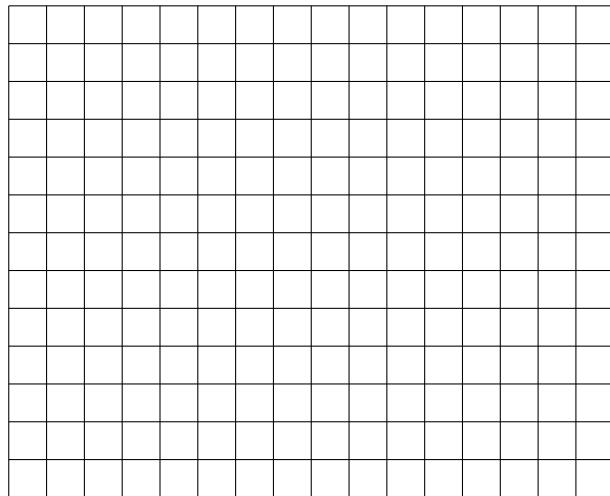
## Les formats audios

1. Cette technique est celle utilisée dans certains formats de sons "bruts" comme le RAW ou WAV.
2. D'autres formats comme le MP3, compressent encore ces informations, comme le fait le JPG pour les images.
3. D'autres formats, comme le MIDI, n'utilise pas cette méthode et font le choix de noter directement l'équivalent des notes de musique et des instruments sur lesquels elles sont jouées. Ce format est adapté aux musiques composées sur synthétiseurs, mais pas aux sons enregistrés dans la nature.

## 2.2 Exercices

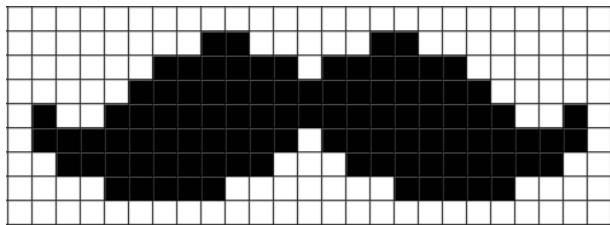
3

Décoder l'image suivante :



6,5,2,3  
4,2,5,2,3,1  
3,1,9,1,2,1  
3,1,9,1,1,1  
2,1,11,1  
2,1,10,2  
2,1,9,1,1,1  
2,1,8,1,2,1  
2,1,7,1,3,1  
1,1,1,1,4,2,3,1  
0,1,2,1,2,2,5,1  
0,1,3,2,5,2  
1,3,2,5

4 Encoder l'image suivante :



5

1. Combien de niveaux de gris pourrait-on encoder sur 10 bits ?
2. Sur 8 bits, quel gris est le plus foncé entre celui représenté par la valeur 42 et celui représenté par la valeur 233 ?
3. Quel est d'après vous le code RGB du violet foncé ?
4. Quel est d'après vous la couleur correspondant au code RGB suivant : (255, 128, 0) ?



# STRUCTURE DE DONNÉES FINIE ET BASES D'ALGORITHMIQUE

## 6.1 La variable

### 6.1.1 Définition

#### Définition 1 :

Une *structure de données* est une structure logique destinée à stocker et organiser des données de façon à faciliter leur utilisation. Parmi ces structures, on retrouve la *variable*, le *tableau*, le *graphe*, l'*arbre*....

En programmant en Makecode Microbit ou en Python nous aurons souvent besoin de faire appel à des variables. Nous allons maintenant les définir :

#### Définition 2 :

La *variable* est une structure de données finie. C'est un élément qui associe un nom (l'identifiant de la variable) à une valeur. Nous pouvons différencier la *constante* qui, une fois qu'une valeur lui a été attribuée, ne change plus, et la *variable mutable* qui elle peut changer de valeur tout au long du programme.

#### Remarques 1 :

1. En Microbit, la variable est créée avant qu'une valeur ne lui soit attribuée. Pour régler ce problème, dès qu'une variable est créée, Microbit lui associe la valeur 0 par défaut. La valeur peut être ensuite modifiée en utilisant le bloc :
2. En Python, une variable est créée lorsqu'on lui assigne une valeur.

---

```
1 variable = 5
2 nombre_eleve = 16
```

---

1 Regarder le programme suivant, et deviner ce qu'il va se passer quand on va activer le Microbit :

**2** Regarder le programme en python suivant et deviner ce qu'il va se passer quand on va l'exécuter :

---

```

1  eleve = 20
2  enseignant = 2
3  print( eleve )
4  print( enseignant )

```

---

### 6.1.2 Nom d'une variable

Le nom choisi pour chaque variable est important pour faciliter la lisibilité de votre programme. Il vous permet de vous souvenir du rôle de chaque variable. Imaginez des milliers de blocs d'instructions Microbit dans votre programme avec des centaines de variables. Si vous nommez vos variables avec des noms peu explicites comme v1, v2, v3, v4 ... v100 seriez-vous capable de retrouver à quoi sert chaque variable ?

Bien que nous soyons toujours libre d'écrire le nom d'une variable comme bon nous semble, un certain nombre de convention sont utilisées :

- 1) Un nom pertinent : essayez d'être le plus précis et concis possible, mais préférez la compréhension à la longueur du nom de la variable.
- 2) Toujours commencer par une minuscule
- 3) Pas de caractères spéciaux ou d'accents
- 4) Si une variable est un mot composé, deux solutions s'offrent à vous :
  - a) Notation *camel case* : voiciUnExemple. Chaque nouveau mot commence par une majuscule.
  - b) Notation *snake case* : voici\_ un\_ exemple. Chaque nouveau mot est séparé par un *underscore* :\_.

**3** Parmi les noms de variables suivants, lesquels respectent les conventions d'écritures des variables ?

1. laVariable
2. la variable
3. nom\_de\_famille
4. la\_quantité
5. nombreDeVoitures
6. prenom
7. leNumeroDeTelephone
8. le\_Nom
9. iztrdr3fsfs

### 6.1.3 Évolution d'une variable

#### En Microbit

En Microbit il existe deux façon de modifier une variable. La première consiste à lui attribuer une valeur, comme on l'a vu précédemment, avec le bloc *Définir variable à ...* :

L'autre façon de modifier une variable consiste à lui ajouter ou soustraire un certain nombre avec le bloc :

On peut dire qu'on **incrémente** la variable *test\_2*. **Incrémenter** signifie ajouter une certaine valeur, souvent 1, à un nombre ou une variable. Dans le bloc précédent, la valeur de *test\_2* va changer et son contenu passera de 0 à 1.

On parler de **décrémenter** quand on soustrait une certaine valeur. Par exemple, dans le bloc suivant la variable *test\_2* qui valait 1, voit son contenu passer de 1 à 0.

Il est aussi possible de mettre un calcul dans l'attribution d'une valeur à la variable. Le prochain bloc d'instruction définit la valeur de la variable *var\_1* au résultat du calcul  $2 + 4$ .

Ou même d'utiliser la valeur de la variable elle-même :

Dans ce cas de figure, on additionne 4 au contenu de la *var\_1*. La variable *var\_1* verra ainsi sa valeur actuelle incrémentée de 4.

## En Python

En Python, pour donner une valeur à une variable, nous allons utiliser le signe "`=`" :

---

`1 classe = 16`

---

Pour incrémenter une variable nous allons écrire :

---

`1 classe = classe + 1`

---

Cette écriture peut prêter à confusion :

1. Elle n'a pas de sens mathématique.
2. Le signe "`=`" n'a donc pas la même signification en informatique et en mathématique !
3. Il s'agit d'un symbole d'affectation (nous plaçons un certain contenu dans une variable) et non un symbole d'égalité.
4. Il faut le lire comme : *la variable classe prend comme nouvelle valeur son ancienne valeur à laquelle on ajoute 4*.

Il est donc très important de comprendre cette finesse liée à la programmation !

**Exercices**

**4** Quel nombre sera affiché par le programme ?

**5** Quel nombre sera affiché par le programme ?

**6** Quel nombre sera affiché par le programme ?

**7** Quels nombres seront affichés par le programme ?

**8** Que fait ce programme ?

---

```

1 argent =10
2 argent=argent +5
3 argent=argent -3
4 argent=argent *2
5 print (argent)

```

---

**9** Que fait ce programme ?

---

```

1 benefice =12
2 benefice=benefice+benefice
3 benefice=benefice-benefice
4 print (benefice)

```

---

**10** Que fait ce programme ?

---

```

1 benefice = 12
2 perte = -3
3 benefice = benefice + perte
4 perte = perte + benefice
5 benefice = benefice + benefice
6 print (benefice)
7 print (perte)

```

---

**11** Que fait ce programme ?

---

```

1 premier = 15
2 deuxieme = 10
3 difference = premier - deuxieme
4 deuxieme = premier + difference
5 premier = difference - deuxieme
6 difference = deuxieme - premier
7 print (premier)
8 print (deuxieme)
9 print (difference)

```

---

## 6.2 Les types

### 6.2.1 Motivation

#### En Microbit

Une variable n'aura pas toujours comme valeur un nombre. Dans l'onglet *Avancé* ensuite *Texte* de l'éditeur makecode.microbit.org, il est possible de choisir le symbole " " représentant une chaîne de caractères. Ainsi il est possible d'attribuer comme valeur une chaîne de caractères :

Ainsi la variable *var\_2* est une chaîne de caractères. Essayons d'imaginer ce qu'il va se passer si nous faisons le programme suivant :

Que se passe-t-il ?

Lorsque Microbit repère un problème, il le signale. Dans l'exercice qui suit, essayez de comprendre le problème qui a été rencontré :

**12** Quel problème a été rencontré dans chacun de ces programmes ?

#### Remarque 2 :

Chaque langage de programmation aura ses propres spécificités par rapport aux différents types d'erreurs qui peuvent apparaître. Par exemple, en Microbit, le programme ci-dessous fonctionne, et affiche 22 !!

#### En Python

En Python, contrairement à Microbit, une variable peut d'abord valoir un nombre puis ensuite un caractère ou un mot.

---

1 a=2  
2 print (a)  
3 a="bonjour"  
4 print (a)

---

Le programme précédent afficher d'abord le nombre 2 puis le mot "bonjour".

13

A votre avis, que fait le programme suivant ?

(*indication : le programme fonctionne*)

---

```

1 a=5
2 b="2"
3 d=2
4 c=a*d
5 print(c)
6 c=a*b
7 print(c)
```

---

### 6.2.2 Définition

On a vu que suivant la valeur attribuée à une variable, le programme va permettre de faire certaines actions et en refuser d'autre. On définit cela formellement :

#### Définition 3 :

Le *type* d'une variable est l'interprétation que fera l'ordinateur de la valeur associée à cette variable. Ce type pourra être, entre autre, un nombre entier (*integer*), un nombre décimal (*float*), une chaîne de caractère (*string*), un booléen (*boolean*)...

Selon le langage de programmation que l'on utilise, le type d'une variable peut ou ne peut pas être changé. On parlera de **typage statique** ou **typage dynamique**. Par exemple, Microbit a un typage statique (Une variable définit comme un nombre ne peut plus être changé en une chaîne de caractères) alors que Python a un typage dynamique.

On regarde maintenant chacun de ces types en détail.

### 6.2.3 Les types *entier* (**integer**) et *décimal* (**float**)

La première catégorie de variables avec laquelle on aimerait travailler est le nombre. Cette catégorie se divise en 2 catégories principales. Comme nous l'avons vu, un nombre entier (positif ou négatif) n'utilise pas le même encodage qu'un nombre à virgule (qui utilise plutôt un encodage sous la forme d'une écriture scientifique). L'espace mémoire n'est donc pas le même.

#### Définition 4 :

Le type *entier* (*integer*) caractérise une variable dont la valeur est un nombre entier. Le type *décimal* (*float*) caractérise une variable dont la valeur est un nombre décimal.

Pour un programme, **1** sera un *entier* alors que **1.0** sera un *décimal*.

Pour Microbit ou Python, ainsi que pour de nombreux langages de programmation, il n'y a pas de problèmes à passer d'un nombre entier à un nombre décimal au cours du programme sans devoir changer de variable et de type.

### 6.2.4 Le type *chaîne de caractères* (**string**)

Une autre valeur que peut prendre une variable est la chaîne de caractères, ou le type *string*.

**Définition 5 :**

Le type *chaîne de caractères* (string) caractérise une variable dont la valeur est une suite de caractères. Cette suite peut aussi contenir des nombres qui sont alors interprétés comme des caractères.

Le programme suivant n'a par exemple aucune signification pour Microbit :

Le même problème apparaît en Python. Un message d'erreur nous dira que nous ne pouvons pas faire "+" entre un "int" et un "str" :

---

```

1 a=5
2 b="2"
3 c=a+b

```

---

Nous ne pouvons pas effectuer d'opération avec les chaînes de caractères (sauf le + que l'on verra par la suite). Il est par contre possible de comparer des chaînes de caractères :

Dans ce cas, comme la lettre a (contenu dans *var\_1*) se trouve avant b (contenu dans (*var\_2*) en prenant compte de l'ordre alphabétique, le texte contenu dans la *var\_1* s'affiche. Le Microbit affiche le texte a.

Prenons un autre exemple :

Dans ce cas, Microbit affiche C. Pourquoi le programme considère t'il que C est plus petit que b ? Tout simplement parce qu'il se base sur son code ASCII pour comparer ! Le code ASCII de C est 67 (base 10) alors que b est 98 (base 10).

L'opération principale sur les chaînes de caractères est la **concaténation** :

**Définition 6 :**

La *concaténation* est une action qui prend en entrée plusieurs chaînes de caractères et renvoie une unique chaîne de caractères composée des différentes chaînes de caractères misent bout à bout.

Par exemple, *concaténation*("abc"; "d2")="abcd2".

Dans Microbit, l'opération + a le même effet que la concaténation si on lui donne deux chaînes de caractères. Ainsi, le programme suivant donnera "32" :

Alors que le programme ci-dessous va afficher 5 :

En Python, la concaténation passe par le `+`. Le programme suivant affichera "5d" :

---

```

1 a="5"
2 b="d"
3 c=a+b
4 print( c )

```

---

### Affichage d'une variable

Les guillemets précisent qu'on a à faire à une chaîne de caractères. Cette différence est importante comme on peut le noter dans le programme suivant :

---

```

1 test=25
2 print( test )
3 print( "test" )

```

---

Ce programme affichera d'abord la variable `test`, donc 25. Puis affichera le texte "test".

#### 6.2.5 Le type *booléen* (*boolean*)

En informatique nous devons souvent tester une condition ("le nombre suivant est-il pair?", "la variable vaut-elle 2 ?"...). Pour cela, nous avons besoin d'un type de variable qui ne correspondra ni à un nombre ni à une chaîne de caractères mais qui pourra nous permettre de résoudre un problème du type : si "il pleut", alors "il faut prendre un parapluie". Le test "il pleut" peut prendre deux valeurs : soit c'est vrai, soit c'est faux. C'est ce que nous appelons une **variable booléenne** :

##### Définition 7 :

Le type *booléen* caractérise une variable qui peut avoir une des deux valeurs suivantes : "vrai" ou "faux".

#### En Microbit

Dans Microbit, le type booléen est noté par un hexagone vert. Dans le cas suivant la variable `var_1` est initialisée avec la valeur vrai (*true*).

On remarque qu'une variable booléenne ne peut pas être affichée comme un nombre dans Microbit :

Mais il peut être affiché comme un texte (le Microbit affiche *true* même si votre éditeur de code est configuré en français) :

### En python

En Python, un booléen va souvent être sous la forme d'une comparaison ou d'une égalité. On voudra par exemple vérifier que l'âge de quelqu'un est strictement inférieur à 12 ans pour avoir une réduction :

---

```
1 if age < 12:
2     print("Vous avez le droit à une réduction")
```

---

"age<12" est un booléen, c'est soit vrai, soit faux. Si c'est vrai, alors on affichera "Vous avez le droit à une réduction". Et si c'est faux, rien ne sera fait.

D'autres comparaisons peuvent être faites :

1. l'égalité (attention au double "==" qui montre que l'on vérifie si il y a égalité. le symbole "==" seul est gardé pour assigner une valeur à une variable) :

---

```
1 if age == 12:
2     print("Vous avez 12 ans.")
```

---

2. la différence :

---

```
1 if age != 12:
2     print("Vous n'avez pas 12 ans.")
```

---

3. plus petit ou égal :

---

```
1 if age <= 12:
2     print("Vous avez 12 ans ou moins.")
```

---

### 6.2.6 Le type *liste* (list)

On peut avoir besoin de rassembler en une même variable un ensemble d'éléments : par exemple les noms des élèves d'une classe, ou les cartes d'un jeu de cartes. Pour cela, il existe un type qui est la *liste* :

#### Définition 8 :

Le type *liste* caractérise une variable qui contient une série d'éléments qui peuvent être ou ne pas être du même type (cela dépend du langage de programmation).

En Python, les éléments d'une liste ne sont pas forcément du même type et un élément d'une liste peut aussi être une liste. Pour créer une liste contenant les éléments 1,2 et 3 nous écrirons :

---

```
1 lst = [1, 2, 3]
```

---

Une liste peut être affiché avec la commande : `print(lst)`. On peut rajouter un élément à une liste en l'additionnant (comme pour une concaténation) à la liste existante. Si par exemple nous voulons rajouter le 4 dans notre liste, nous écrirons :

---

```
1 lst = lst + [4]
```

---

Si nous voulons accéder à un élément d'une liste, nous indiquons l'indice de l'élément recherché :

- lst[0] correspond au premier élément de la liste lst, qui ici est 1.
- lst[1] correspond au deuxième élément de la liste lst, qui ici est 2... et ainsi de suite.

## Exercices

14

Définir quel type (entier, décimal, chaîne de caractères, booléen, liste) devrait-on associer à une variable si cette variable représente :

- a) l'âge d'une personne ;
- b) si il pleut ;
- c) les prénoms des enfants d'une famille ;
- d) un mois de l'année ;
- e) le prix d'un vêtement ;
- f) si une personne est majeure ;
- g) une année de naissance ;
- h) les sports pratiqués par un élève ;
- i) l'aire d'une forme géométrique ;
- j) le nom d'une personne ;
- k) si un végétal donné est un fruit ;

15

Donner le type de chaque variable de ce programme puis dire si des erreurs vont être reconnues par Microbit (tester le programme si besoin) :

16

Reconnaître les différents types apparaissant dans ce programme :

---

```
1 pi = 3.14
2 rayon = 5
3 petit_et_grand = [40, 100]
4 circonference = pi * 2 * rayon
5 reponse = "C'est un petit cercle"
6 if circonference < petit_et_grand[0]:
7     print(reponse)
```

---

17

Dire ce qui va être affiché pour chacun des programmes suivants :

---

```
1. _____
1 x = 5
2 y = 12
3 print(x)
4 print("y")
5 print(x < y)
```

---

---

```

2. _____
1 x = 12
2 y = 2 * x
3 print(x == 2 * y)
4 print(y == 2 * x)

```

---

## 6.3 Bases de l'algorithme

Nous allons commencer à discuter ce que représente un algorithme et comment peut-on décomposer une tâche en tâches plus élémentaires dans le but de résoudre un problème ou d'accomplir un travail. La pensée algorithmique est un pan essentiel de l'informatique. Une fois correctement formulée ou précisée, elle permet, via un langage de programmation, d'implémenter différents traitements sur les machines (par exemple votre ordinateur).

### 6.3.1 Définition

**Définition 9 :**

Un algorithme est une suite finie et non-ambiguë d'opérations ou d'instructions permettant de résoudre un problème.

On connaît depuis l'antiquité des algorithmes sur les nombres, comme par exemple l'algorithme d'Euclide qui permet de calculer le *pgdc*<sup>1</sup> de deux nombres entiers.

Pour le traitement de l'information, on a développé des algorithmes opérant sur des données non numériques : les algorithmes de tri, qui permettent par exemple de ranger par ordre alphabétique une suite de noms, les algorithmes de recherche d'une chaîne de caractères dans un texte, ou les algorithmes d'ordonnancement, qui permettent de décrire la coordination entre différentes tâches, nécessaire pour mener à bien un projet.

Algorithme provient du nom du mathématicien perse *Al-Khawarizmi* ( $\sim 820$ ), le père de l'algèbre.

### 6.3.2 Les structures de contrôle

Pour construire un algorithme, nous avons besoin d'une suite d'instructions. Ces instructions sont prises parmi un ensemble de *fonctions* pour Python ou de *blocs* pour Microbit et sont exécutées les unes après les autres. Afin de *répéter* une même action un certain nombre de fois ou de faire une action *si* une condition est vraie, nous avons besoin de structure de contrôle :

**Définition 10 :**

Une *structure de contrôle* détermine dans quel ordre vont être exécutées les actions. Il en existe trois :

1. La *séquence* : les instructions sont effectuées les unes après les autres
2. La *sélection* : les instructions sont effectuées si une certaine condition est respectée (par exemple la structure *si*)
3. La *répétition* : les instructions sont répétées en boucle *tant qu'* une certaine condition est respectée.

---

1. pgdc : plus grand diviseur commun.

Ses structures sont récurrentes et apparaissent toujours sous une forme ou sous une autre dans presque tous les langages de programmation. Nous allons maintenant étudier les différentes structures qui sont à notre disposition.

### 6.3.3 La condition *si* (if)

#### Le *if simple*

Une des premières structures qui nous intéresse est la structure de sélection *si* ou *if* en anglais. Sa syntaxe en pseudo-code est de la forme :

```
si condition  
alors action
```

Le *si* vérifie si la **condition** est *vraie* et si c'est le cas, la ligne avec l'**action** est effectuée. La **condition** est donc forcément un **booléen** qui sera soit vrai soit faux.

Par exemple, on peut avoir le code suivant :

```
si il pleut  
alors je prends mon parapluie
```

**il pleut** est bien un booléen. Cela est soit vrai soit faux. Si cela est vrai, la ligne **je prends mon parapluie** est exécutée. Si c'est faux, cette ligne ne sera pas évaluée.

Pour un code Microbit, cela donnerait :

---

En Python, l'expression d'une condition commence par un **if**, suivi d'un booléen puis de deux points ":". Tout ce qu'il faudra faire si la condition est vérifiée doit être indenté :

---

```
1 variable=1
2 if variable == 2:
3     print( variable )
```

---

L'indentation est très importante. Pour le comprendre, voici trois exemple et ce qu'ils affichent :

---

```
1 variable=1
2 if variable == 2:
3     print( variable )
4     print( "fin" )
```

---

Dans ce programme, rien n'est fait. La condition n'est pas vérifiée, donc les instructions incluses dans le **if** (celles avec une indentation) ne sont pas effectuées.

---

```
1 variable=1
2 if variable == 2:
3     print( variable )
4 print( "fin" )
```

---

Dans ce programme, "fin" est affiché. La condition du **if** n'est pas vérifiée et donc l'instruction "print(variable)" n'est pas faite. Par contre, en retournant à la ligne par la suite, on indique que nous sortons du **if** et l'instruction "print("fin")" est effectué.

### Multiples conditions

Une condition *si* peut vérifier d'autres conditions, qui débutent par *sinon si* (else, if) et peut aussi finir par un *sinon* (else) qui ne sera exécuté que si toutes les autres conditions étaient fausses :

```
Si j'ai entre 500 et 1000 .-
    alors je pars en Europe
sinon, si j'ai entre 1000 et 2000 .-
    alors je pars en Asie
sinon, si j'ai plus que 2000.-
    alors je pars en Amérique
sinon je reste chez moi
```

Le programme équivalent en Microbit serait :

En python, une condition commence par un **if** et est suivie par autant de **elif** (contraction de *else, if* (sinon, si)) nécessaire et fini, si besoin, par un **else** (sinon) qui lui ne vérifie aucune condition.

---

```
1  if  500 <= argent and argent <= 100 :
2      print( "Europe" )
3  elif 1000 <= argent and argent <= 2000:
4      print( "Asie !" )
5  elif 2000 <= argent :
6      print( "Amerique !!!" )
7 else :
8     print( "Maison . . ." )
```

---

### Exercices

18 Qu'affiche ce programme ?

19 Qu'affiche ce programme ?

20

Dans chacun des cas suivants, dire ce qu'affichera le programme :

1. si  $var\_1 = 5$ ,  $var\_2 = 9$  et  $var\_3 = 2$ .
2. si  $var\_1 = 7$ ,  $var\_2 = 3$  et  $var\_3 = 3$ .
3. si  $var\_1 = 3$ ,  $var\_2 = 5$  et  $var\_3 = 6$ .
4. si  $var\_1 = 4$ ,  $var\_2 = 4$  et  $var\_3 = 4$ .

21

Qu'affiche le programme suivant :

---

```

1  nombre_1 = 3
2  nombre_2 = 7
3  nombre_3 = 10
4  if nombre_1 == nombre_2 :
5      print(nombre_1)
6  elif nombre_2 < nombre_3 :
7      print(nombre_2)
8  elif nombre_3 == 10 :
9      print(nombre_3)
10 if nombre_2 != 7 :
11     print(nombre_2)
12 else :
13     print(nombre_3)

```

---

22

---

```

1  if x < y :
2      if x < z :
3          print(z)
4      else :
5          print(x)
6  elif x < z :
7      if x > y :
8          print(y)
9      elif y < z :
10         print(z)
11 if z < y :
12     print(y)
13 print(x)

```

---

Dans chacun des cas suivants, dire ce qu'affichera le programme :

1. si  $x = 3$ ,  $y = 5$  et  $z = 1$ .
2. si  $x = 3$ ,  $y = 5$  et  $z = 9$ .
3. si  $x = 9$ ,  $y = 5$  et  $z = 1$ .
4. si  $x = 6$ ,  $y = 3$  et  $z = 4$ .

### 6.3.4 La boucle répéter (repeat)

#### En Microbit

La condition *si* est une structure nous permettant de coder des conditions mais elle n'est pas suffisante. Pour certain problème, nous avons besoin de répéter une certain nombre de fois une action. Le nombre de répétition peut-être connue à l'avance ou ne pas l'être.

Nous aimerais, par exemple, doubler la valeur de *var\_1* 5 fois de suite. La première possibilité est la suivante :

Lorsque nous répétons plusieurs fois la même action, il est plus facile d'utiliser une boucle *répéter* (*repeat*), qui effectuera une même tâche un certain nombre de fois :

L'intérêt de *répéter* est que nous pouvons aussi utiliser une variable pour le nombre de fois que nous voulons répéter une tâche et donc obtenir une grande flexibilité dans notre programme.

### En Python

En Python, pour répéter une même tâche on peut utiliser la fonction *for*, qui permet de faire de nombreuses choses, dont la répétition. Considérons le programme suivant qui répète plusieurs fois la même tâche :

---

```

1 x = 2
2 x = 2 * x
3 x = 2 * x
4 x = 2 * x
5 x = 2 * x
6 x = 2 * x
7 print(x)

```

---

On peut écrire le programme suivant qui effectue la même tâche, mais en moins d'instructions :

---

```

1 x = 2
2 for k in range(5) :
3     x = 2 * x
4 print(x)

```

---

On verra par la suite que la fonction *for* permet de faire beaucoup plus qu'une répétition. Dans l'écriture *for k in range(5)*, *k* est une variable et pourrait être nommée avec le nom qu'on souhaite. Il faut par contre éviter de le nommer avec un nom de variable déjà utilisé précédemment.

### Exercices

23

Que fait le programme suivant ?

24

Que fait le programme suivant ?

25

1. Que fait ce programme si *flocon\_x* = 2 ?
2. Que fait ce programme si *flocon\_x* = 4 ?

26

Que fait le programme suivant quand on l'exécute ?

---

```

1 i = 3
2 for k in range(7) :
3     print(i)

```

---

27

Que fait le programme suivant quand on l'exécute ?

---

```

1 i = 3
2 for k in range (7) :
3     i = 2 * i
4 print (i)

```

---

28

Que fait le programme suivant quand on l'exécute ?

---

```

1 i = 5
2 n = 10
3 for k in range (n) :
4     print (i)
5     i = i + 1

```

---

### 6.3.5 La boucle *tant que* (while)

*répéter* est une structure de répétition qui ne nous permet pas de faire "efficacement" toutes les répétitions, ou boucles, que nous rencontrons en programmation. Pour palier à cette difficulté, deux autres structures se rencontrent souvent : la boucle *tant que* (while) et la boucle *pour* (for). Nous allons voir dans ce chapitre la première.

Le format de la boucle *tant que* se présente sous la forme :

*tant que condition est vraie*  
faire une certaine séquence d'instructions

La **condition** consiste souvent à vérifier qu'une variable ne dépasse pas une certaine valeur. Il faudra alors **incrémenter** cette variable avant chaque répétition de la boucle. Lorsque cette valeur est dépassée, on sort de la boucle.

*repeat* et *while*

Il est souvent assez facile de passer d'une boucle *tant que* à une boucle *répéter*. Par exemple, les deux programmes suivants, utilisant les structures de répétition *répéter* et *tant que* sont équivalents :

### En Python

Pour montrer comment on utilise la boucle *while* en Python, on va partir d'une situation concrète : Alice gagne 1200.- par mois. Et son salaire monte de 130.- par mois. Bob gagne lui 2900.- par mois, et son salaire augmente de 70.- par mois. Combien de mois faudra-t-il pour qu'Alice gagne plus que Bob ?

Il existe bien sur des techniques mathématiques pour résoudre ce problème mais nous pouvons aussi représenter la situation par un programme. Nous aurons une variable pour chacun des salaires d'Alice et Bob, et une variable qui compte le nombre de mois, et qui vaudra au début 0. On va répéter nos calculs mensuels tant que le salaire d'Alice est plus petit que le salaire de Bob. Et chaque fois, on va calculer les nouveaux salaires d'Alice et de Bob et ajouter un mois en plus. Ce qui donne :

---

```

1  salaire_alice = 1200
2  salaire_bob = 2900
3  nombre_de_mois = 0
4  while salaire_alice <= salaire_bob :
5      salaire_alice = salaire_alice + 130
6      salaire_bob = salaire_bob + 70
7      nombre_de_mois = nombre_de_mois + 1
8  print(nombre_de_mois)

```

---

Il faudra faire attention lorsqu'on écrit une boucle *while* à ne pas avoir une condition qui demeure toujours vrai, sinon le programme ne s'arrêtera jamais.

### Exercices

**29** Que fait le programme suivant ?

**30** Que fait le programme suivant ?

**31** Que fait le programme suivant ?

**32** Que fait le programme suivant quand on l'exécute ?

---

```

1  n = 2
2  compteur = 0
3  limite = 100
4  while n < limite :
5      n = 2 * n
6      compteur = compteur + 1
7  print(compteur)

```

---

**33**

---

```

1  abo_a = 50
2  abo_b = 0
3  nombre_seances = 0
4  while abo_b < abo_a :
5      abo_a = abo_a + 9
6      abo_b = abo_b + 15
7      nombre_seances = nombre_seances + 1
8  print(nombre_seances)

```

---

**34**

---

```
1 x = 0
2 n = 10
3 compteur = 0
4 while x < n :
5     z = 0
6     while z < x :
7         compteur = compteur + 1
8         z = z + 1
9     x = x + 1
10 print(compteur)
```

---

# PROGRAMMATION - STRUCTURES DE CONTRÔLE

## 8.1 Les Boucles

Il est souvent nécessaire en programmation de devoir répéter un bloc d'instructions un certain nombre de fois, que ce soit pour parcourir une liste, pour faire une action de manière cyclique, etc. Ce nombre d'itérations est parfois connu en avance et doit parfois être déterminé au fur et à mesure des répétitions. En fonction de la situation, il existe plusieurs structures de contrôle permettant ces répétitions. Ces structures sont appelées des *boucles*.

### 8.1.1 La boucle `for`

#### Définition 1 :

En python, la boucle `for` permet de répéter un bloc d'instructions pour chaque élément d'un container (tableau, chaîne de caractères, etc.). Elle est introduite par le mot-clé `for` suivi d'un nom de variable, puis du mot-clé `in` et du container à parcourir, et enfin du symbole ":"

**Exemple 2.** L'exemple suivant permet d'afficher 10x le mot "Coucou". À chaque itération de la boucle, le programme va exécuter la ligne contenant l'instruction `print`.

Listing 8.1 – Boucle for - parcours tableau

---

```
1 for k in range(10):
2     print("Coucou")
```

---

#### Remarque 1 :

Dans les cas où l'on veut exécuter un bloc d'instructions un nombre précis de fois, la fonction `range` (`begin`, `end`) permet de générer un tableau de `begin` à `end-1` qui peut ensuite être utilisé avec une boucle `for`. L'exemple 3 peut ainsi être écrit :

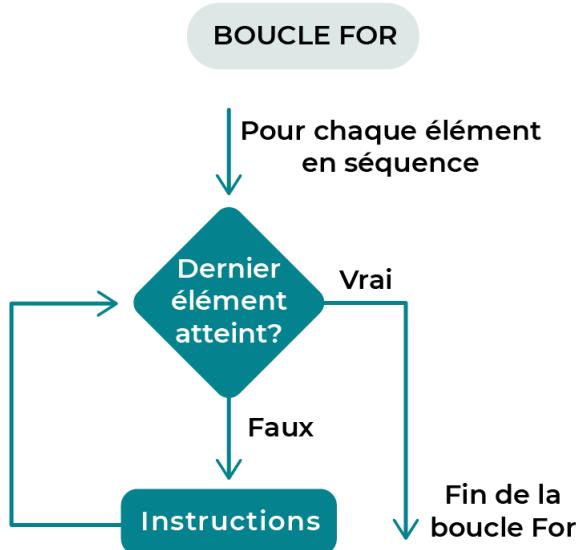


FIGURE 8.1 – Schéma fonctionnel d'une boucle for

Listing 8.2 – Boucle for - compteur trivial

---

```

1 maxCompteur = 10
2 for compteur in range(maxCompteur + 1):
3     print(compteur)

```

---

### 8.1.2 La boucle while

Si la boucle `for` est la plus pratique à utiliser dans des cas précis, il existe une boucle plus modulable. C'est la boucle `while`.

#### Définition 2 :

La boucle `while` permet de répéter un bloc d'instructions tant qu'une condition n'est pas remplie. En python, elle est introduite par le mot-clé `while` suivi d'un test terminé par le symbole ":".

**Exemple 3.** L'exemple suivant permet d'afficher un compteur allant de 1 à 10. À chaque début de boucle, le programme va évaluer la condition `compteur <= maxCompteur` puis exécuter le bloc d'instruction si elle vaut `True` et sortir de la boucle si elle vaut `False`.

Listing 8.3 – Boucle while - compteur trivial

---

```

1 maxCompteur = 10
2 compteur = 1
3 while compteur <= maxCompteur:
4     print(compteur)
5     compteur = compteur + 1

```

---

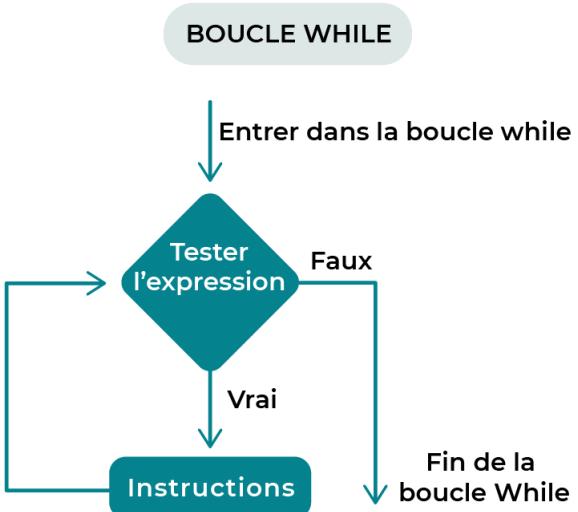


FIGURE 8.2 – Schéma fonctionnel d'une boucle while

### 8.1.3 Exercices sur les ordinateurs

```
#####  
#####  
#####  
#####  
#####
```

- 6 Demander à l'utilisateur-trice de saisir un mot jusqu'à ce qu'iel saisisse un mot de plus de 5 caractères.
- 7 Demander à l'utilisateur-trice de saisir un nombre jusqu'à ce qu'iel saisisse un nombre pair.
- 8 Afficher le menu d'un programme jusqu'à ce que l'utilisateur-trice saisisse "q" pour quitter.  
-- Menu --  
1. Option 1  
2. Option 2  
q. Quitter  
Choisissez une option :
- 9 Écrire un programme qui demande à l'utilisateur-trice de saisir une phrase et qui affiche cette phrase sans les espaces.
- 10 Écrire un programme qui demande à l'utilisateur-trice de saisir un mot et qui affiche ce mot à l'envers.
- 11 Imaginer :  
a) une boucle infinie, n'atteignant jamais sa condition de fin.  
b) une boucle qui n'exécute jamais son bloc d'instructions.
- 12 Écrire un programme énumérant les 100 premiers nombres de la suite de Fibonacci.
- 13 Écrire un programme permettant d'additionner les 100 premiers entiers naturels (1, 2, ..., 100).
- 14 En vous inspirant de l'exercice 20, écrire un programme trouvant le minimum entre N nombres.  
Testez avec N=10.

## 8.2 Les conditions

Avec les concepts vus jusqu'ici, un programme exécuté 100 fois de suite donnera 100 fois les mêmes résultats, ce qui présente assez peu d'intérêt. Pour que l'exécution de notre programme dépende de différents paramètres, on utilise des structures de contrôle nommées *conditions*.

La condition est un élément de base d'algorithme que l'on utilise au quotidien.

**Exemple 4.** *S'il fait beau, j'irai en montagne.*

Ces structures permettent d'exécuter ou de ne pas exécuter certaines instructions en fonction de certaines conditions et de changer ainsi le déroulement du programme.

Étapes	Instructions
1	Se lever
2	Regarder la météo
3.1	S'il fait beau, aller en montagne

### 8.2.1 L'instruction if

Un premier moyen d'exprimer des conditions en programmation est l'instruction if. En python, elle s'utilise de la manière suivante :

Listing 8.4 – Instruction if

---

```

1 if meteo == "beau":
2     print("Aujourd'hui, je vais en montagne.")

```

---

#### Définition 3 :

L'instruction `if` doit être suivie d'une expression retournant un booléen (`True` ou `False`) puis d'un symbole `:`. Cette expression peut être une variable contenant un booléen ou une opération retournant un booléen comme les opérateurs `<`, `>`, `==` ou `!=`.

Dans cet exemple, si la variable `meteo` a été initialisée à `"beau"`, la phrase `Aujourd'hui, je vais en montagne.` va s'afficher. Si elle a été initialisée à une autre valeur, par exemple `meteo = "moche"`, rien ne s'affichera.

**15** Quelle est la différence entre ces deux programmes ? Qu'afficheront-ils respectivement :

- a) dans le cas `meteo = "beau"` ?
- b) dans le cas `meteo = "moche"` ?

Listing 8.5 – Programme A

---

```

1 if meteo == "beau":
2     print("Aujourd'hui, ")
3     print("je vais en montagne.")

```

---

Listing 8.6 – Programme B

---

```

1 if meteo == "beau":
2     print("Aujourd'hui, ")
3     print("je vais en montagne.")

```

---

**Réponse :**

### 8.2.2 L'instruction else

L'instruction `if` peut être suivie d'une instruction `else` qui est exécutée lorsque le résultat du test est `False`. Par exemple, on pourrait préciser la situation précédente :

**Exemple 5.** *S'il fait beau, j'irai en montagne. Sinon, j'irai au cinéma.*

Notre algorithme deviendrait alors :

Étapes	Instructions
1	Se lever
2	Regarder la météo
3.1	S'il fait beau, aller en montagne
3.2	S'il ne fait pas beau, aller au cinéma

En python, cela s'écrirait de la manière suivante :

Listing 8.7 – Instruction if - else

---

```

1 if meteo == "beau":
2     print("Aujourd'hui, je vais en montagne.")
3 else:
4     print("Aujourd'hui, je vais au cinéma.")

```

---

### 8.2.3 L'instruction elif

Dans le cas d'une situation non binaire, on peut introduire des conditions supplémentaires qui sont examinées si les conditions précédentes ne sont pas remplies. On peut par exemple nuancer notre situation ainsi :

**Exemple 6.** *S'il fait beau, j'irai en montagne. S'il ne fait pas beau et qu'il pleut, j'irai au cinéma. Sinon, j'irai courir au bord du Rhône.*

**Remarque 2 :**

Ici, le terme *Sinon* signifie "S'il ne fait pas beau et qu'il ne pleut pas". C'est pareil en python, l'instruction conditionnée par `else` est exécutée si toutes les conditions précédentes sont fausses.

Notre algorithme devient alors :

Étapes	Instructions
1	Se lever
2	Regarder la météo
3.1	S'il fait beau, aller en montagne
3.2	S'il ne fait pas beau et qu'il pleut, aller au cinéma
3.3	S'il ne fait pas beau et qu'il ne pleut pas, aller courir au bord du Rhône

En python, on utilise l'instruction `elif` qui est la contraction de *else if* :

Listing 8.8 – Instruction if - else

---

```

1 if meteo == "beau":
2     print("Aujourd'hui, je vais en montagne.")
3 elif meteo == "pluie":
4     print("Aujourd'hui, je vais au cinéma.")
5 else:
6     print("Aujourd'hui, je vais courir au bord du Rhône.")

```

---

### 8.2.4 Exercices sur papier

16

Que vaut la variable solution à la fin du programme suivant ? Réponse :

Listing 8.9 – Exercice

---

```

1 a = True
2 b = ((2+2) > 4)
3 c = (2==3)
4
5 if c:
6     solution=42
7 elif b:
8     solution=2021
9 elif a:
10    solution=1202
11 else:
12    solution=73

```

---

17

Que vaut la variable solution à la fin du programme suivant ? Réponse :

Listing 8.10 – Exercice

---

```

1 a = 42
2 b = 2021
3 c = 1202
4 d = 73
5
6 if c < (d+a):
7     solution = "A"
8 elif b+c == a:
9     solution = "B"
10 elif a<=42:
11     solution = "C"
12 else:
13     solution = "D"

```

---

### 8.2.5 Exercices sur les ordinateurs

18

Écrivez un programme vérifiant si un nombre A est un multiple d'un autre nombre B. Utilisez pour cela l'opérateur modulo "%", qui, pour  $x\%y$ , retourne le reste de la division euclidienne de x par y.

Testez avec A=2941 et B=17.

19

Écrivez un programme vérifiant si un nombre A est un multiple de 2 nombres B et C.

Testez avec A=26469, B=17 et C=9.

**20**

Écrivez un programme trouvant le minimum entre 3 nombres.

**21**

Écrivez un programme déterminant si une année est bissextile.

### 8.3 Bibliographie

- *L'Informatique autrement*, De la pensée computationnelle aux applications numériques, Dimitri Racordon, Damien Morard, Emmanouela Stachtiari, Aurélien Coet, Alexandre-Quentin Berger, Didier Buchs
- <https://openclassrooms.com/fr/courses/7168871-apprenez-les-bases-du-langage-python>
- <https://docs.python.org/fr/3/tutorial/index.html>

## CHAPITRE 5

# ORDINATEURS

### 5.1 Qu'est-ce qu'un ordinateur (computer) ?

#### Définition 1 :

Un ordinateur est une machine électronique capable de traiter (calculer, stocker) des données de manière automatisée et optimisée à l'aide de logiciels, ainsi que de les partager au travers d'un réseau et de périphériques.

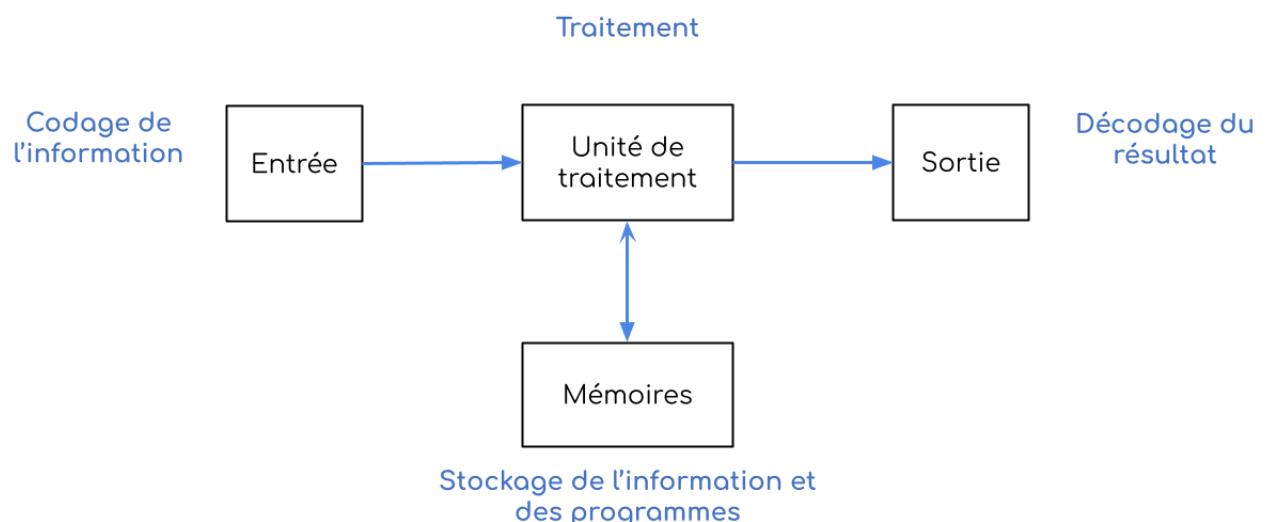


FIGURE 5.1 – Schéma fonctionnel de l'ordinateur

On parle souvent de PC (Personnal Computer) ou ordinateur personnel. C'est tout simplement un ordinateur qui répond aux besoins des utilisateurs (humains).



FIGURE 5.2 – Exemple d'ordinateur personnel

Il existe des ordinateurs plus puissants qui sont spécialement conçus pour fournir des informations et des logiciels à d'autres ordinateurs reliés via un réseau. On les appelle des **serveurs**. Ces derniers sont capables de traiter des charges de travail plus importantes et d'exécuter davantage d'applications.

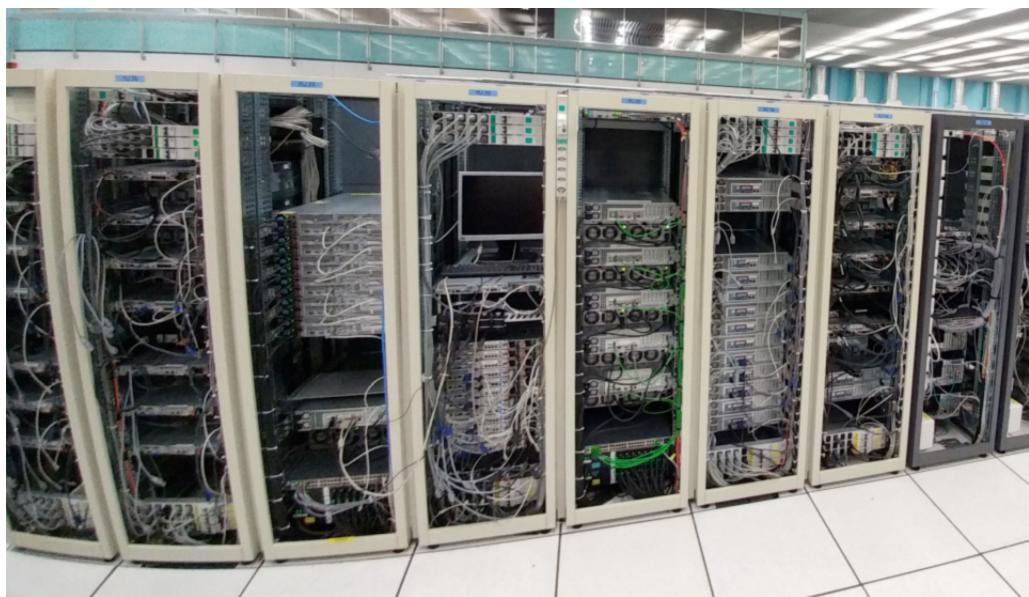


FIGURE 5.3 – Salle de serveurs du CERN

Les ordinateurs les plus puissants au monde sont appelés supercalculateur. Ils contiennent des centaines, voire des milliers de processeurs. Ils sont souvent utilisés dans la recherche médicale, les applications scientifiques (cf Figure 5.3), le domaine de la finance, la météorologie et à des fins militaires.

## 5.2 L'intérieur d'un ordinateur

### 5.2.1 Les composants

Pour fonctionner, l'ordinateur est constitué de composants électroniques comme la mémoire (4) ou le processeur (2). Ces composants sont regroupés sur une carte mère (1) qui est le circuit imprimé<sup>1</sup> principal. Elle permet de prendre en charge la mémoire vive (4), la lecture du disque dur (7) et l'utilisation du processeur. Cette carte mère est fixée à l'unité centrale également nommée : la tour, le boîtier, desktop, etc.

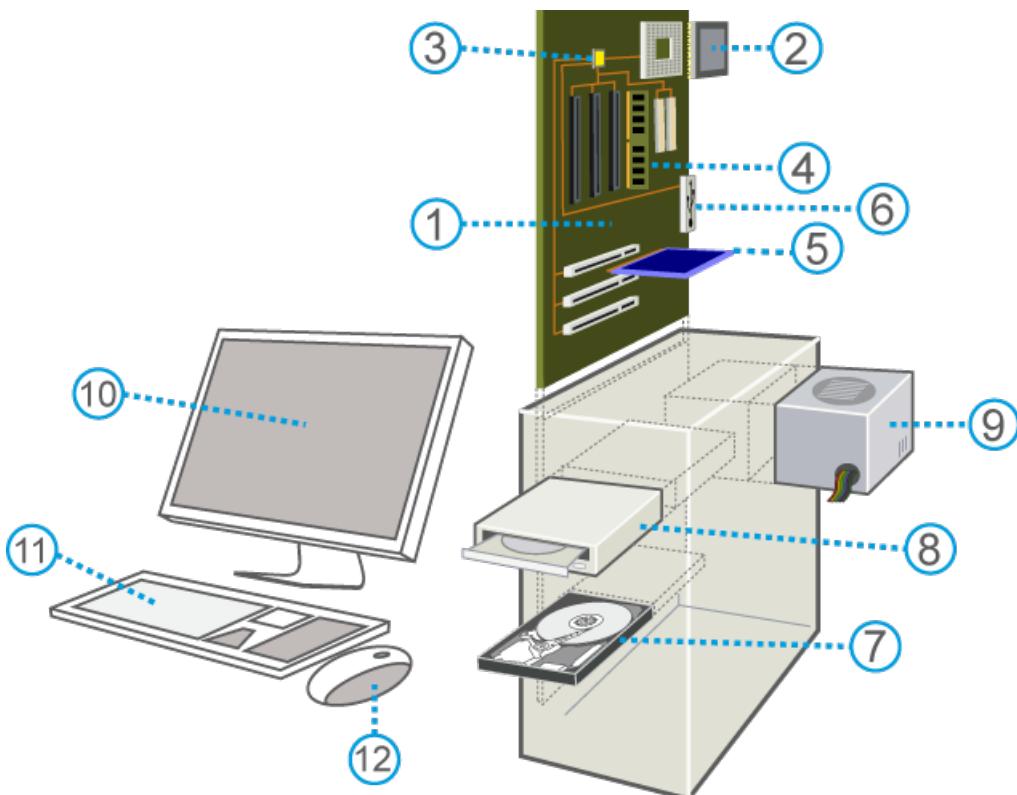


FIGURE 5.4 – L'ordinateur, l'unité centrale et sa composition

Les principaux composants constituant un ordinateur sont les suivants :

#### La carte mère

Une **carte mère** (1) se distingue principalement par la génération de processeur qu'elle peut accueillir, par le type de mémoire vive pouvant être connecté, le nombre et les différents types de connecteurs (6) (IDE, sata, PCI, etc.) qu'elle comporte et par son jeu de composants électriques (chipset) lui permettant de gérer l'échange de données entre les disques, la mémoire, la carte graphique et les différents périphériques externes d'entrée/sortie. La carte mère comporte également un ensemble de fonctions nommé BIOS (*Basic Input Output System*) ou plus récemment UEFI (*Unified Extensible Firmware Interface*) lui permettant, lors de sa mise sous tension, d'identifier le matériel connecté sur la carte mère.

1. Le circuit imprimé est un support, en général une plaque, permettant de maintenir et de relier électriquement un ensemble de composants électroniques entre eux.



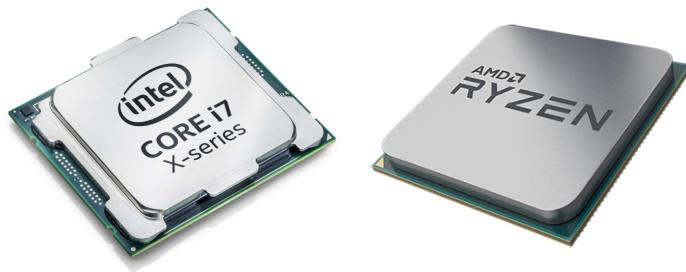
## Processeur

Le **processeur** (CPU, Central Processing Unit) (2) : il gère tous les calculs pour permettre à l'ordinateur de fonctionner. Il va exécuter une suite d'instructions cadencées par une horloge (la fréquence du processeur). Il permet donc d'exécuter différents programmes. Il va s'appuyer sur la mémoire vive pour stocker ses calculs.

Un processeur se distingue principalement par sa fréquence de fonctionnement et son architecture. Sa **fréquence** s'exprime en **GigaHertz [GHz]**. Concrètement, une fréquence de 2 GHz signifie que le processeur peut réaliser deux milliards d'opérations à la seconde. Son **architecture** désigne le **nombre de bits utilisés pour chaque nombre manipulé**, souvent 32 ou 64 bits pour les ordinateurs de bureau.

Le processeur réalise un nombre impressionnant d'opérations par seconde, mais toujours une à la fois, de manière séquentielle. Il peut cependant intégrer **plusieurs cœurs**, qui lui permettent de réaliser plusieurs opérations en parallèle.

Parmi les entreprises réputées dans la conception de processeur, on trouve : AMD, IBM, Intel, Hewlett-Packard, Hitachi, Oracle, Motorola, Texas Instrument.



## Mémoire vive

La **mémoire vive** (Random Access Memory) (4) est une mémoire très rapide où se trouvent les informations traitées par le processeur. Lorsque l'ordinateur est éteint, les informations contenues en mémoire vive ne sont plus maintenues, on perd l'information.

Qu'il s'agisse de **mémoire vive** ou **mémoire permanente**, la mémoire est principalement caractérisée par sa capacité de stockage et sa rapidité d'accès en lecture et en écriture. Pour la mémoire vive, on parle de [Go] de mémoire vive.



### Mémoire permanente

La **mémoire permanente** (7) est une mémoire moins rapide que la mémoire vive, mais de plus grande capacité utilisée pour sauvegarder des données. Le disque dur (hard drive) (7) est un exemple de mémoire permanente, c'est le système historique de stockage de données mais est lentement remplacé par les disques SSD qui sont plus rapides.

Actuellement, sa capacité est traduite en gigaoctets [Go] ou Téraoctets [To]. La vitesse de lecture et d'écriture est donnée en Mo/s.

### L'alimentation

**L'alimentation** (9) est branchée sur le réseau électrique 220 [V], elle alimente tous les composants électroniques.

Elle est caractérisée par sa puissance. La puissance de l'alimentation détermine la quantité d'électricité délivrée en watts [W]. Elle permet de fournir un courant continu nécessaire aux circuits électroniques. Les alimentations comportent une certification (label) donnant une garantie d'économie d'énergie. Ce label est accordé aux alimentations dont le rendement – rapport entre puissance fournie et consommée – est supérieur à 80 %.



### La carte graphique

La **carte graphique** (5) transmet les images qu'elle possède en mémoire sur les écrans. Elle permet donc d'afficher à l'écran des images, du texte, des vidéos, etc. en allumant des points lumineux (pixels). À noter que certains processeurs comportent une unité graphique rendant obsolète l'ajout d'une carte graphique sur un ordinateur, typiquement pour un usage de type bureautique.

La carte graphique se caractérise principalement par l'architecture de son processeur graphique (GPU) et sa mémoire vive. À ce jour, Nvidia et AMD sont leaders dans la conception des processeurs graphiques. On distingue très facilement les cartes graphiques les plus puissantes par le système de refroidissement qu'elle possède.



### Le lecteur CD/DVD/Blu-ray

Le lecteur/graveur CD/DVD/Blu-ray (8) lit l'ensemble des CD, DVD et Blu-ray, que ce soit de la musique, des logiciels, des jeux, des films, etc.

#### Remarque 1 :

Dans un smartphone, c'est le **SoC, (system on a chip)** qui rassemble quasiment tous ces composants : CPU, GPU (processeur de la carte graphique), modem (connexion réseau), mémoire RAM... Dans une petite puce (M1 ou A11 par exemple chez apple ou Exynos chez Samsung) se retrouve concentrée la puissance d'un ordinateur d'il y a quelques années.

### 5.2.2 Récapitulatif des unités

[HTML]C0C0C0Composant	[HTML]C0C0C0Caractéristique	[HTML]C0C0C0Unité	[HTML]C0C0C0
Processeur	Fréquence d'horloge Technologie (Architecture) Nombre de coeurs	Hertz bits coeurs	2 [GHz] 64 bits 8 coeurs
Mémoire vive	Technologie (Architecture) Capacité	octet	DDR3, DDR4 32 [Go]
Mémoire permanente	Technologie (Architecture) Connecteur Capacité	octet	SSD, mécanique sata, usb, M.2 2 [To]
Alimentation	Puissance	watt	450 [W]

## 5.3 Périphériques externes

### 5.3.1 Exemples de périphériques

Pour interagir avec l'ordinateur, des périphériques d'entrée/sortie sont nécessaires. Il en existe de toutes sortes et ces derniers sont connectés à l'ordinateur par différent type de câbles (réseau,

USB, DVI, série, etc.) ou à l'aide d'une connexion sans fil (infrarouge, wifi, Bluetooth, lasers, etc).

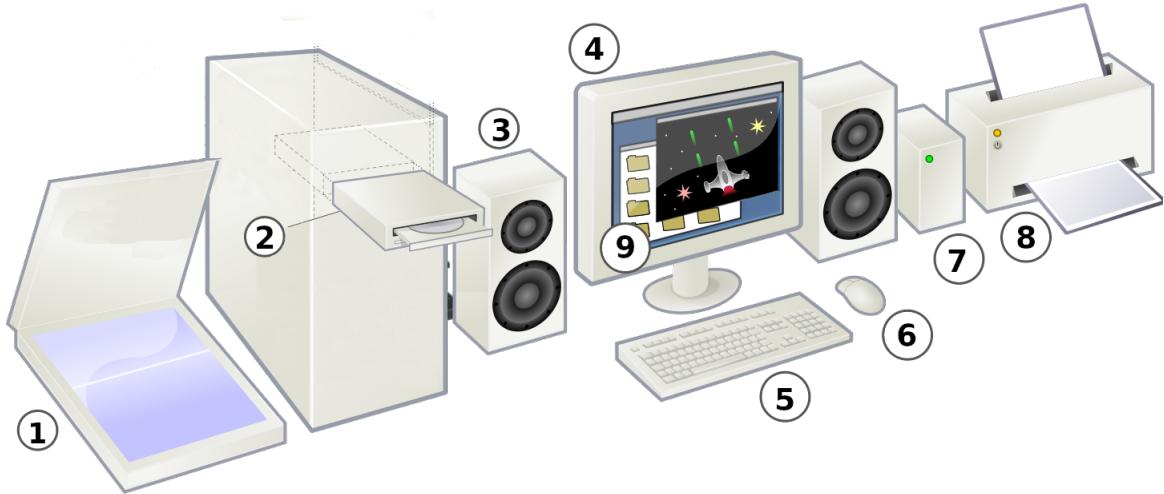


FIGURE 5.5 – Périphériques d'un ordinateur

Les différents types de périphériques externes sont :

- 1) Le **scanner** permet la numérisation de document dans une certaine résolution.
- 2) Le **lecteur** CD ou DVD ou Blu-ray sont des lecteurs permettant de lire un support amovible à l'aide d'un faisceau laser.
- 3) Le **haut-parleur** permet de diffuser du son.
- 4) L'**écran** permet d'afficher les informations, mais lorsqu'il est tactile, il offre la possibilité de se passer de la souris et du clavier. Il est caractérisé par sa taille et sa définition d'écran qui est le nombre de points ou pixels que peut afficher un écran. Actuellement, les définitions (nombre de points H x V) les plus courantes sur un écran pour ordinateur sont :
  - XGA : 1024 x 768
  - HD : 1280 x 768
  - Full HD : 1920 x 1080
  - WUXGA : 1920 x 1200
  - WQHD : 2560 x 1440
  - 8K Full Format : 8192 x 4320
- 5) le **clavier**. utilisé pour écrire, mais également se déplacer rapidement dans un texte ou dans des menus, voire même effectuer une copie d'écran (screenshot).
- 6) La **souris** sert à déplacer le curseur sur l'écran, de choisir des applications et aussi de cliquer sur des icônes pour démarrer des logiciels. Elle permet de naviguer sur votre ordinateur, mais également de dessiner, sélectionner des textes, etc. Elle est mobile avec ou sans fil.
- 7) Le **disque externe** est un disque dur qui n'est pas intégré dans l'ordinateur. En fonction de sa taille, il peut être facilement déplacé. En fonction du modèle, les disques externes sont soit alimentés par un branchement USB, soit avec une alimentation sur secteur 220[V].
- 8) L'**imprimante** 2D permet l'impression de texte, image, graphique. Il existe plusieurs technologies d'impressions telles que laser, jet d'encre, sublimation, etc. Depuis les années 2010, l'impression 3D prend de l'ampleur avec l'arrivée de nouvelles technologies innovantes basées sur de nouveaux matériaux comme le plastique, la cire, le métal, la céramique, le verre. Il est même possible d'imprimer en 3D une forme en chocolat !

- 9) L'**interface graphique** ou l'**environnement graphique** est un dispositif logiciel permettant le dialogue entre l'utilisateur·trice et l'ordinateur. En fonction de son utilisation, un périphérique est considéré soit en entrée, soit en sortie, soit comme étant bidirectionnel. Le tableau suivant résume le type de périphérique.

### 5.3.2 Périphériques d'entrée et de sortie

En fonction de son utilisation, un périphérique est considéré soit en entrée, soit en sortie, soit comme étant bidirectionnel. Le tableau suivant résume le type de périphérique.

Si le périphérique envoie des données à l'ordinateur, on l'appelle un **périphérique d'entrée**.

Si le périphérique reçoit des données de l'ordinateur, on l'appelle un **périphérique de sortie**.

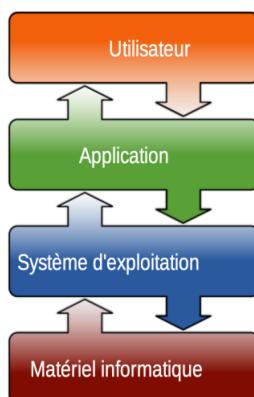
Un périphérique peut aussi envoyer et recevoir des données de l'ordinateur : c'est un **périphérique d'entrée et de sortie**.

## 5.4 Système d'exploitation

Pour qu'un ordinateur soit capable de faire fonctionner un programme informatique (appelé parfois application ou logiciel), la machine doit être en mesure d'effectuer un certain nombre d'opérations préparatoires afin d'assurer les échanges entre le processeur, la mémoire, et les ressources physiques (périphériques). C'est le système d'exploitation qui s'en occupe :

#### Définition 2 :

Le **système d'exploitation** (noté SE ou OS, abréviation du terme anglais Operating System) est chargé d'assurer la liaison entre les ressources matérielles, l'utilisateur et les applications (traitement de texte, jeu vidéo..).



Ainsi, lorsqu'un programme désire accéder à une ressource matérielle, il ne lui est pas nécessaire d'envoyer des informations spécifiques au périphérique, il lui suffit d'envoyer les informations au système d'exploitation, qui se charge de les transmettre au périphérique concerné via son pilote :

#### Définition 3 :

Un **pilote** (ou **driver**) est un programme particulier qui permet la bonne liaison entre l'ordinateur et un périphérique particulier (scanner, imprimante...).

En l'absence de pilotes, il faudrait que chaque programme reconnaisse et prenne en compte la communication avec chaque type de périphérique !

Le système d'exploitation permet ainsi de « dissocier » les programmes et le matériel, afin notamment de simplifier la gestion des ressources et offrir à l'utilisateur une interface humain – machine simplifiée afin de lui permettre de s'affranchir de la complexité de la machine physique.

Il existe de nombreux systèmes d'exploitation. Les plus connus à ce jour sont : Microsoft Windows, GNU/Linux, Apple Mac OS, Oracle Solaris, Google Android, etc.

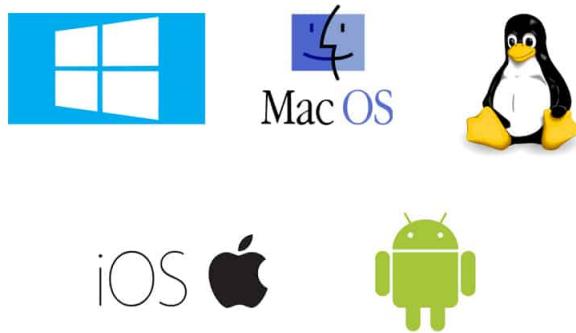


FIGURE 5.6 – Différents systèmes d'exploitation

## 5.5 Exercices

1 Relie chaque composant à son nom :



Mémoire vive (RAM)



Carte mère



L'alimentation



Carte graphique



Le disque dur



Processeur

2

Réponds aux questions suivantes :

1. Je suis le circuit imprimé principal où tous les composants sont reliés.

Je suis :

2. Je gère tous les calculs pour permettre à l'ordinateur de fonctionner.

Je suis :

3. Je suis connecté à l'ordinateur et j'affiche tout ce qu'il m'envoie comme du texte, des dessins, des photos, des films, etc.

Je suis :

4. Je m'appelle mémoire permanente et je conserve toutes les données même si l'ordinateur est éteint.

Je suis :

5. Sans moi, on ne peut pas taper du texte.

Je suis :

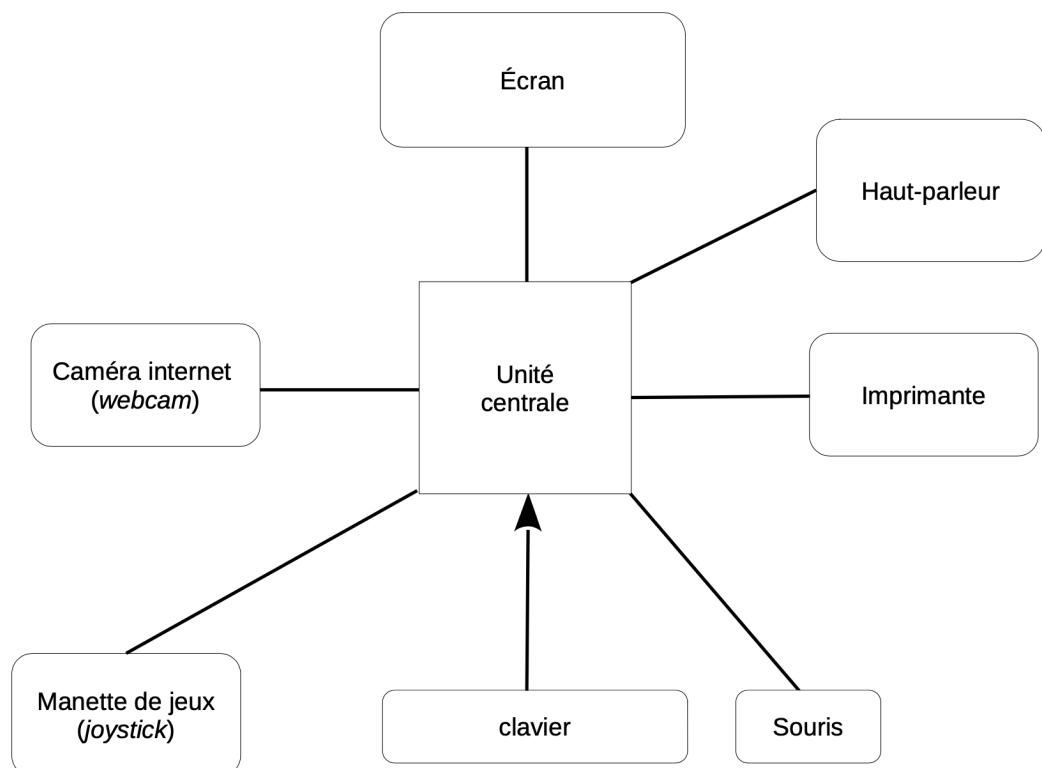
6. Je suis une mémoire plus rapide que la mémoire permanente et je ne conserve pas les données si l'ordinateur est éteint.

Je suis :

### 3 Complète le schéma.

Le schéma suivant représente les différents périphériques externes d'un ordinateur. Dessine le sens des flèches pour indiquer si le périphérique est en entrée ou en sortie.

*Par exemple : le clavier est un périphérique d'entrée, il y a donc une flèche qui part du clavier à l'unité centrale.*



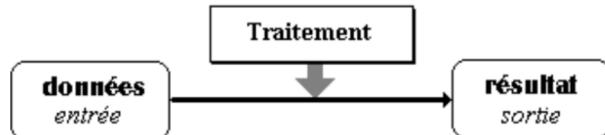


# PROGRAMMATION - INTRODUCTION

## 6.1 Traitement automatique des données

Un algorithme permet de résoudre un problème donné au moyen d'un nombre fini d'opérations élémentaires. L'application manuelle d'un algorithme est une tâche fastidieuse. C'est pourquoi l'ordinateur a été inventé dans le but d'automatiser l'implémentation d'algorithmes. On parle de l'informatique comme *la science du traitement automatique de l'information*.

Ce que l'on attend d'un programme c'est qu'il prenne les données d'entrée du problème, qu'il les transforme au moyen de l'algorithme, et qu'il redonne le résultat attendu.



Pour que le programme s'exécute sur un ordinateur, il faut indiquer à la machine comment s'y prendre. D'un côté l'ordinateur qui comprend seulement le langage binaire composé d'une suite de 0 et de 1 et de l'autre, le·la programmeur·euse qui parle un langage destiné à la communication entre humains comme le français, rempli d'ambiguïtés (plusieurs interprétations possibles de la même phrase). Il faut trouver un intermédiaire, c'est ce qu'on appelle un **langage de programmation**.

### Définition 1 :

Un **langage de programmation** est un langage formel utilisé pour expliquer en détail à un ordinateur comment réaliser une tâche.

Les langages de programmation sont caractérisés par une **syntaxe** très rigide (afin qu'aucune ambiguïté n'existe) et qui permet aux ordinateurs d'exécuter ce que les humains leur disent de faire. La syntaxe de codage sera constituée d'ensembles de mots et symboles spécifiques qui devront se combiner dans un ordre bien particulier (**règle de syntaxe**). Chaque fois que cet ordre ne sera pas satisfait, l'ordinateur générera une erreur (**erreur de syntaxe**).

### Définition 2 :

Les **règles de syntaxe** définissent l'ensemble des programmes qui sont valides et accep-

tables par l'interpréteur/compilateur. En français, les règles de syntaxe incluent l'orthographe et la grammaire, et déterminent quelles sont les phrases valides.

### *Langages naturels Vs Langages de programmation*



#### *Langages naturels*

- Communiquer entre humains
- Mots corrects (orthographe)
- Phrases correctes (grammaire)

#### *Langages de programmation*

- Indiquer à une machine comment résoudre un problème
- Identificateurs valides
- Syntaxe valide

## 6.2 Langages de programmation

Pour que le programme écrit dans un langage de programmation soit compréhensible par l'ordinateur il devra être traduit par un programme appelé interpréteur ou compilateur (en fonction du type de langage et de son application). À l'heure actuelle plusieurs centaines de langages de programmation sont utilisés en fonction du domaine, des applications et des préférences du·de la programmeur·euse.

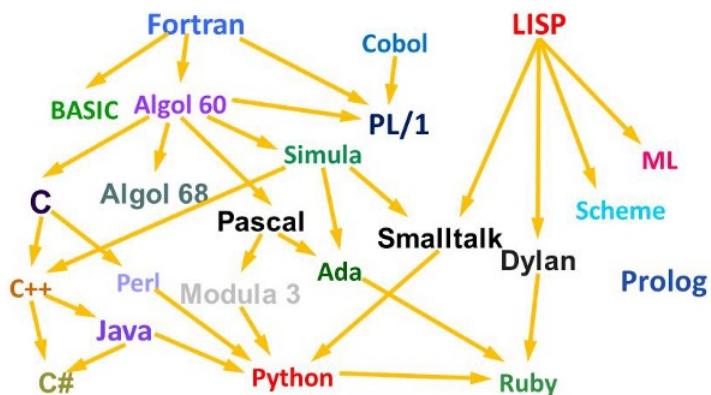


FIGURE 6.1 – Arbre généalogique des langages de programmation les plus utilisés.

Il existe plusieurs critères pour catégoriser tous ces langages de programmation. Nous allons en passer en revue quelques-uns.

### 6.2.1 Paradigmes de programmation

Selon Wikipedia, un *paradigme de programmation* est une façon d'approcher la programmation informatique et de traiter les solutions aux problèmes et leur formulation dans un langage de programmation approprié. Les paradigmes les plus communs sont

1. **programmation impérative**
2. **programmation fonctionnelle**
3. **programmation orientée objet**

Certains langages de programmation permettent de programmer en utilisant plusieurs approches. C'est notamment le cas du langage Python que nous utiliserons dans ce cours, même si nous adopterons principalement l'approche impérative.

### 6.2.2 Proximité avec le langage binaire-machine

On différencie des langages de programmation en fonction de leur proximité avec le langage binaire-machine qui est considérée comme le langage de plus bas niveau. On distingue deux niveaux :

1. **bas niveau (proche machine)** : permet de contrôler tout ce qui se passe dans la machine pendant l'exécution du code.

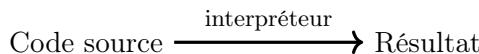
C'est un langage machine lisible par un humain qui permet simplement de manipuler explicitement des registres, des adresses mémoires, voire des instructions machines. Par exemple : assembleur, C....

2. **haut niveau (proche utilisateur)** : permet à l'utilisateur·trice d'utiliser des fonctions complexes sans se soucier de l'aspect machine (enfin un petit peu quand même). Pour son optimisation, il utilise parfois (souvent) des langages de bas niveau de façon transparente pour l'utilisateur·trice. Par exemple : FORTRAN, Python, PHP, SQLite. . .

### 6.2.3 Langage interprété et langage compilé

On peut aussi trier les langages entre langages de programmation interprétés et langages de programmation compilés :

1. **interprété** : le langage est lu avec un programme appelé interpréteur pour exécuter le code au fur et à mesure de la lecture du code.



(a) avantage : Test immédiat

(b) inconvénient : plus lent, le code est interprété à chaque lancement du **programme interprété**.

2. **compilé** : le langage est converti en langage machine (binaire) par l'intermédiaire d'un programme dédié (compilateur) qui produit un programme qui pourra dorénavant s'exécuter seul.



(a) avantage : Rapide

(b) inconvénient : nécessite une première étape de compilation.

### 6.2.4 Langages visuels

La rigidité et l'austérité des langages de programmation classique a poussé les programmeur·euse·s à développer des langages de programmation visuels qui simplifient l'apprentissage de la programmation. Dans un langage visuel, les éléments du langage de programmation sont accessibles sous la forme d'éléments graphiques (le plus souvent des blocs).

1. avantage : des symboles clairs et une absence de syntaxe -> les erreurs de saisie sont impossibles
2. inconvénient : Manque de modularité, il devient très difficile de structurer le programme pour des projets importants.



(a) Langage visuel

```

1 def on_button_pressed_a():
2     basic.show_leds("""
3         # # . # #
4         # # . # #
5         . . . .
6         . # # # .
7         # # # # #
8         """
9     )
10
11 def on_button_pressed_b():
12     basic.show_leds("""
13         . . . .
14         . # . # .
15         . . . .
16         # . . . #
17         . # # # .
18         """
19     )

```

(b) Langage textuel

FIGURE 6.2 – Le même programme écrit dans deux langages différents.

## 6.3 La langage Python et l'IDE Thonny

### 6.3.1 Python

Quel est le meilleur langage de programmation ? Cette seule question a permis de remplir des centaines de forums de programmeur·euse·s sur internet. Chaque langage ayant ses avantages, ses particularités et spécificités liées au matériel et type d'applications à créer. Cette année, nous utiliserons le langage Python. Voici un échantillon de raisons qui ont motivé notre choix<sup>1</sup> :

- Python est un langage interprété ce qui le rend le code portable sur la plupart des ordinateurs/systèmes d'exploitation.
- La syntaxe de Python est très simple.
- Code lisible et compact
- Python est gratuit.
- Python est très utilisé, en particulier pour l'apprentissage de la programmation.
- Python est en 2021 le langage le plus utilisé dans le monde selon la classification<sup>2</sup> IEEE<sup>3</sup> Spectrum
- etc...

1. Voir l'article sur <https://linux-center.org/articles/9812/python.html>

2. Voir la classification sur <https://spectrum.ieee.org/top-programming-languages-2021>

3. IEEE : Institute of Electrical and Electronics Engineers ou en français : l'Institut des ingénieur·euse·s électrique·n·e·s et électronicien·ne·s

### 6.3.2 Environnement de Développement Intégré Thonny

Python est un langage interprété, donc pour écrire des programmes Python nous aurons besoin d' :

- un éditeur de texte pour écrire le code ;
- un interpréteur Python pour tester le code ou des instructions ;
- éventuellement, un débogueur pour pouvoir exécuter les instructions du programme les unes après les autres afin d'identifier les erreurs de programmation.

Un logiciel comme **Thonny**, intègre ces différents aspects dans un seul logiciel, on appelle ce type de logiciel un Environnement de Développement Intégré (IDE en anglais pour *Integrated Development Environment*),

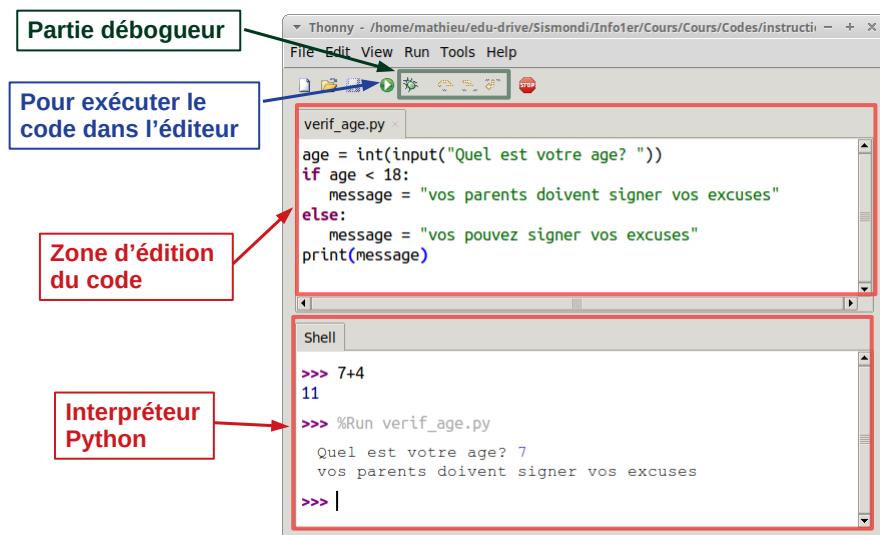


FIGURE 6.3 – IDE Thonny

#### Interpréteur Python

L'interpréteur Python se présente sous la forme d'une fenêtre avec des >>> après lesquels on peut entrer des instructions. En pressant la touche *Entrée* du clavier, Python interprète les instructions et affiche le résultat.

#### L'éditeur

Lorsque l'on souhaite taper une suite de commandes, on préfère créer un script (programme) avec l'éditeur. Ainsi on peut sauvegarder son travail dans un fichier texte. Pour exécuter le code tapé dans l'éditeur, il suffit de cliquer sur le bouton vert avec la pointe de flèche.



On utilisera l'extension .py pour enregistrer les fichiers qui contiennent du code Python. De plus, on veillera à utiliser un nom de fichier cours mais explicatif sans utiliser d'accent et d'espace (les espaces peuvent être remplacer par le caractère `_ underscore`)

### 6.3.3 Première rencontre avec un programme Python

Dans un langage impératif, un programme est constitué d'instructions qui vont de la première à la dernière lignes. Comme nous le verrons pas la suite, l'ordre d'exécution entre la première et la dernière instruction pourra varier en fonction d'instructions de contrôle de flux. Prenons le programme suivant, écrit en Python :

**Programme 6.1 :**

Listing 6.1 – Premier programme Python

---

```

1 print("A vous de jouer")
2 x = int(input("position x : "))
3 y = int(input("position y : "))
4 x_bateau = 7
5 y_bateau = 4
6
7 if x_bateau == x and y_bateau == y:
8     message = "Coulé"
9 else:
10    if x_bateau == x or y_bateau == y:
11        message = "En vue"
12    else:
13        message = "Dans l'eau"
14
15 print(message)

```

---

#### Question :

Que fait ce programme ?

*Le but d'un programme est de traiter de l'information en entrée afin de produire le résultat attendu en sortie. Par conséquent, on retrouvera souvent la structure suivante*

- Une partie entrée où les données du problème sont obtenues (lignes 2 à 5 dans le programme 6.1).
- Le corps du programme où un algorithme est exécuté instruction après instruction en fonction des données lues (lignes 7 à 13 dans le programme 6.1) ;
- Une partie sortie qui affiche ou stocke le résultat obtenu (ligne 15 dans le programme 6.1).

*Nous verrons par la suite qu'un programme est essentiellement constitué d'expressions ou structures de données et d'instructions ou structures de contrôles. Il contiendra toujours les 4 ingrédients de base suivants*

- **des structures de données** : les variables (qui représenteront l'Information).
- **3 types de structures de contrôle du flux d'instructions** (qui permettront de traiter cette Information) :
  - les séquences d'instructions.
  - les tests conditionnels ou alternatives.
  - les boucles ou répétitions.



## CHAPITRE 3

# NOTION DE FICHIERS

### 3.1 Fichiers

#### 3.1.1 Introduction

Un fichier informatique est un ensemble de données numériques que l'on souhaite faire persister en le stockant sur un support permanent. Un fichier peut être : une image, un film, du texte, un programme source, un programme exécutable, un fichier compressé, etc. Quant au support permanent (cf. chapitre Ordinateur) il s'agit d'un disque dur, un DVD, une clé USB, etc.

Un disque stocke simplement une suite de bits, une suite de 0 et de 1. Le nombre de bits qu'un disque peut stocker est appelé sa capacité : par exemple un disque d'un téraoctet (binaire) peut stocker 240 mots de 8 bits, soit un peu plus de huit mille milliards de bits. On peut donc facilement stocker un texte, une image, un son ou un programme sur un tel disque. Cependant, comme on souhaite souvent stocker sur un disque plusieurs images, textes, etc., il faut diviser les huit mille milliards de bits dont le disque est constitué en plusieurs espaces plus petits, que l'on appelle des fichiers. Un fichier est simplement une suite de 0 et de 1, à laquelle on associe un nom. Par exemple, le texte "Je pense, donc je suis." se représente en ASCII étendu comme la suite de 184 bits suivante :

```
latex 0100101001100101001000000111000001100101011011 1001110011011001010010110000100000011001  
1111011011100110001100100000011010100110010100 1000000111001101110101011010010111001100101110
```

Il est possible de stocker cette suite de bits sur un disque en lui donnant le nom *cogito.txt*, l'extension *.txt* indiquant que cette suite de bits exprime un texte en ASCII. L'extension détermine le type d'information exprimé (texte, image, son, etc.) et le format utilisé pour l'exprimer.

#### 3.1.2 Nom de fichier

Un fichier comporte donc un *nom\_de\_fichier* qui permet de l'identifier et d'y accéder. Ce nom est généralement constitué de deux parties séparées par un point. Par exemple, le fichier nommé *Recette\_tarte\_aux\_pommes.odt* se décompose :

Nom du fichier	.	Extension
<i>Recette_tarte_aux_pommes</i>	.	<i>odt</i>

Comme le nom du fichier identifie son contenu, il est important de choisir un nom explicite. En effet, si vous nommez votre rédaction de français avec le nom suivant :

→ *Rédaction\_le\_petit\_prince.odt* : Ce fichier désignera probablement une rédaction avec comme sujet le petit prince.

Par contre, si vous choisissez le nom :

→ *mon\_fichier.odt* : Ce nom ne donne aucune information sur son contenu. On n'aura strictement aucune idée de quoi il s'agit sauf qu'il s'agit d'un fichier !

### **Remarques 1 :**

1. Pour l'utilisateur·trice, il est donc utile et important de bien choisir un nom qui identifie le contenu de son fichier.
2. Vous remarquez également que des caractères soulignés ( \_ ) ont été utilisés dans la nomenclature. En effet, pendant longtemps, les espaces n'étaient pas autorisés pour nommer des fichiers, car les espaces étaient utilisés comme séparateurs de nom de fichier ! Actuellement, l'espace est accepté sur la plupart des systèmes, mais il existe toujours de vieux programmes qui refusent un espace dans le nom du fichier ou génère une erreur.
3. Les caractères accentués (é ê è ù à ä ï ô û) étaient également interdits. C'est pour cette raison que l'on trouve souvent des fichiers qui n'ont pas d'accent (merci l'orthographe me direz-vous). À noter qu'il subsiste également d'autres caractères qui sont interdits ou déconseillés :
  - Microsoft Windows (interdits) < > : " \ / | . ? \*
  - Gnu/Linux et Mac OS (déconseillés) : / \*
4. Les noms ont aussi une limite en longueur, jusqu'à 256 caractères (voir plus pour les tout nouveaux systèmes), mais dans les années 80, le système d'exploitation MS-Dos n'autorisait que 8 caractères plus le point et l'extension de 3 caractères !
5. Certains systèmes de fichier sont sensibles à la casse, mais pas Microsoft Windows. Exemple avec ces trois noms :
  - a) *recette\_tarte\_aux\_pommes.odt*
  - b) *Recette\_tarte\_aux\_pommes.odt*
  - c) *RECETTE\_TARTE\_AUX\_POMMES.odt*
 Ces trois fichiers indiqueront le même document pour Microsoft Windows, mais bien trois fichiers distincts pour GNU/Linux ou macOS.

## **3.2 Extension**

### **3.2.1 Type d'extension**

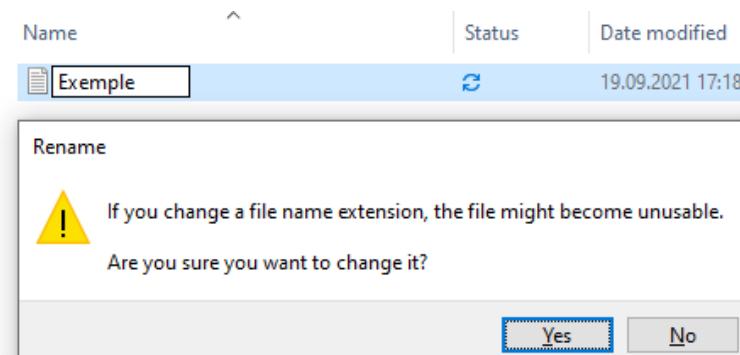
Comme cité précédemment, l'extension est formée en général de 3 lettres, mais elle peut en contenir plus (ex. : .html), voir moins ou carrément aucune. L'extension, par convention, indique la nature du fichier. Voici quelques exemples :

Nature du contenu	Extensions
Document texte libre office writer	odt
Document tableur libre office calc	ods
Document texte Microsoft Word	docx
Document tableur Microsoft Excel	xlsx
Document texte	txt
Exemples de formats de fichiers compressés	zip, 7z, rar, lzw
Exemples de formats de fichiers audio	mp3, wav, ogg, flac
Images, photos	png, jpeg, gif
Video, film	avi, mpg, mov, flv
Page web	htm, html
Exemples de formats de fichiers de code source	java, c, py

### 3.2.2 Type MIME

Les systèmes d'exploitation gnu/Linux et MacOS ne demandent pas d'extension aux différents type de fichiers. En effet, ces deux systèmes se basent sur des informations rattachées aux fichiers, les types MIME. Les types MIME sont désormais appelés "types de média Internet". Les types MIME ont été créés à l'origine pour le courrier électronique. "MIME" (Multipurpose Internet Mail Extensions) est un standard qui a été proposé par les laboratoires Bell Communications en 1991 afin d'étendre les possibilités du courrier électronique (courriel), c'est-à-dire de permettre d'insérer des documents (images, sons, texte, ...) lors d'un envoi de message électronique, mais ils se sont étendus à d'autres utilisations.

MS Windows ignore les types MIME et se base uniquement sur les extensions de fichier. Par exemple, vous pouvez avoir un fichier texte nommé "Exemple.txt". MS Windows comprend qu'il s'agit d'un fichier texte en raison de l'extension de fichier .txt. Supprimez le .txt. extension de fichier et renommer le fichier en "Exemple" sans extension de fichier. MS Windows ne saura pas quoi faire avec le fichier résultant. C'est pourquoi MS Windows vous avertit lors de la suppression de l'extension de fichier, en disant "Si vous modifiez une extension de nom de fichier, le fichier peut devenir inutilisable. Êtes-vous sûr de vouloir le changer ?".



Ce fichier ne deviendra pas inutilisable (pour toujours). Vous pouvez le rendre à nouveau "utilisable" en ajoutant l'extension du fichier d'origine. C'est une des raisons principales de MS Windows de masquer les extensions de fichiers par défaut dans son explorateur de fichier. Cette option évite que les utilisateurs suppriment accidentellement l'extension d'un fichier. Des attaquants (pirates) peuvent abuser de ce comportement pour cacher du code malveillant et des virus dans des fichiers en choisissant de fausses extensions de fichiers.

Les informations sur la nature du fichier, le type MIME, sont intégrées au début du fichier lui-même. Ainsi, lorsque vous ouvrez un fichier sans extension de fichier, gnu/Linux et MacOS examineront le type MIME du fichier pour déterminer de quel type de fichier il s'agit.

### 3.3 Organisation des fichiers

Lorsque vous prenez une photo avec votre smartphone où se trouve-t-elle ? Quel nom de fichier identifie la photo ? Comment vous y prenez-vous pour classer vos photos ? Ne vous est-il pas arrivé de défiler les photos sur votre écran à la recherche d'une en particulier ? Comment faudrait-il procéder pour retrouver plus rapidement une photo ? Si vos photos sont classées par date et par album n'est-il pas plus simple de retrouver une photo ? Le problème dans notre cas de figure c'est que toutes les photographies se retrouvent au même endroit, sans classement, dans la mémoire permanente du téléphone mobile.

En informatique, pour organiser des fichiers, il est nécessaire de créer des dossiers (ou des répertoires). Dans un dossier, nous pouvons trouver d'autres dossiers et fichiers. Cette structure forme une hiérarchie cohérente. Organiser ses fichiers c'est comme ranger ses feuilles de cours dans plusieurs classeurs avec des pochettes et des séparateurs. Prenons l'exemple suivant pour illustrer le classement de ses feuilles de cours papier. Il s'agit donc de classer différents documents de cours dans leur classeur respectif. Dans notre illustration, nous avons choisi de créer un classeur par discipline ce qui permet, par exemple, de ranger la feuille de géographie Suisse dans la pochette Europe qui se trouve dans le classeur Géographie. Ce classeur sera rangé dans l'armoire des classeurs.

Cette structure se retrouve dans les systèmes de fichiers de manière analogue. Sous GNU/Linux ou macOS le niveau 0 se nomme la racine et le dernier niveau les feuilles (par analogie avec un arbre inversé). Le passage d'un niveau à l'autre, les branches, est décrit par un / (slash). Le chemin pour parvenir au fichier depuis la racine (niveau 0) se nomme le chemin d'accès. Il décrit précisément l'emplacement du fichier (l'itinéraire complet). On appelle une telle organisation des fichiers une organisation arborescente, car on peut la visualiser sous la forme d'un arbre.

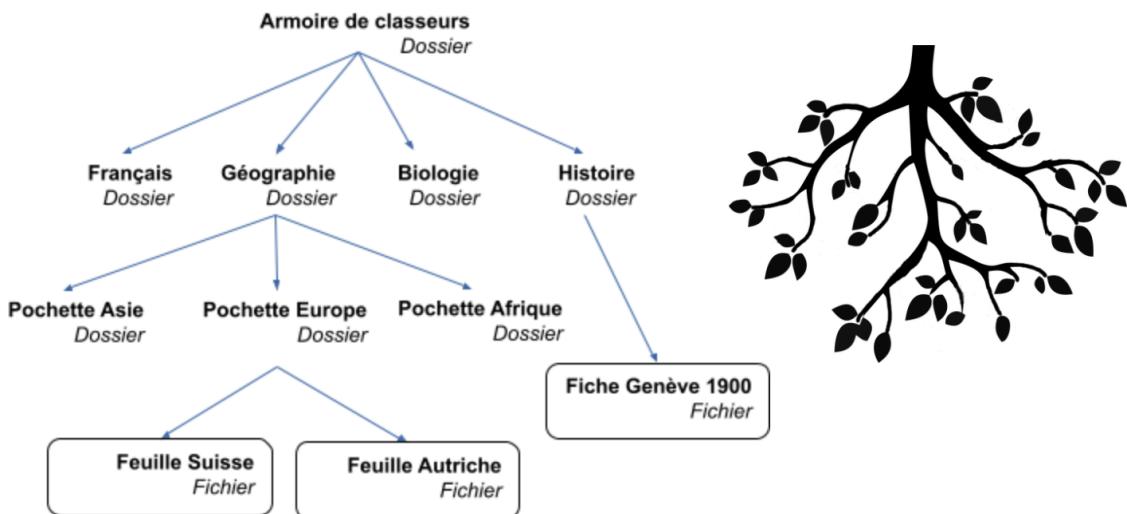
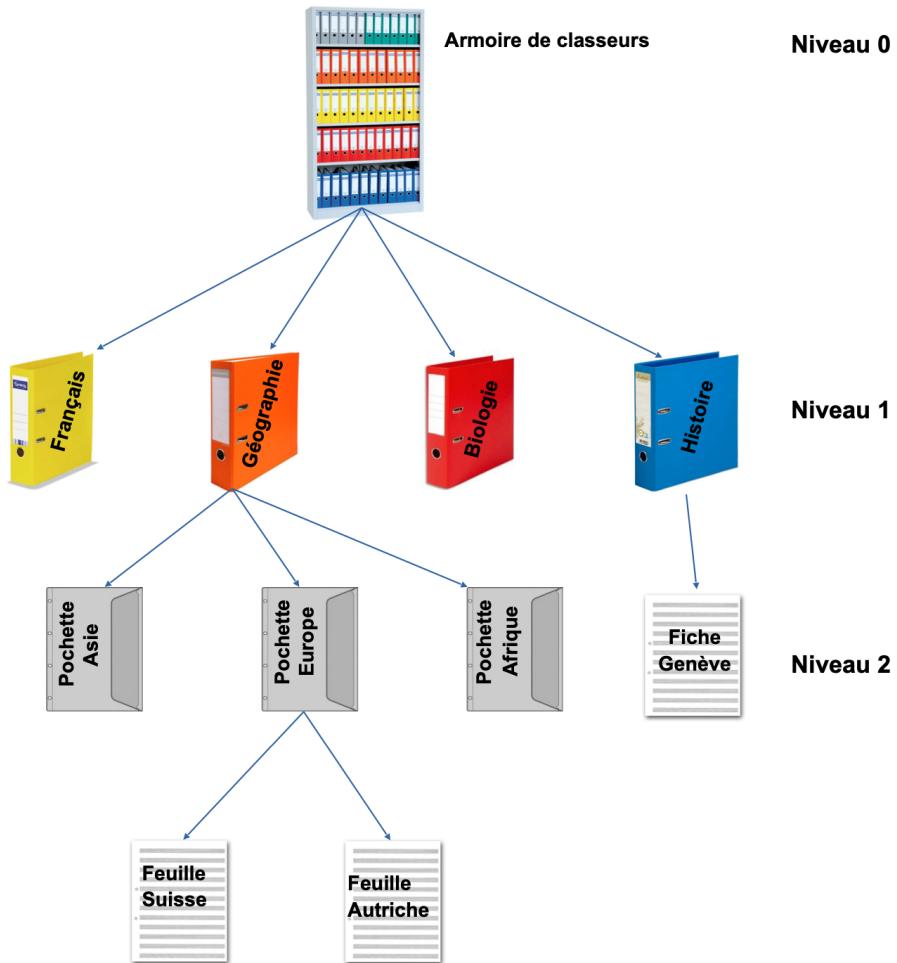
Pour résumer :

- a) Un **dossier** peut comporter d'autres dossiers et des fichiers.
- b) Un **fichier** est un ensemble organisé d'informations, désigné par un nom précis, que le système d'exploitation d'un ordinateur manipule comme une simple entité, dans sa mémoire ou sur un support de stockage.

L'organisation arborescente des fichiers n'est pas le seul moyen de structurer l'information : elle est en concurrence avec d'autres méthodes, parmi lesquelles l'utilisation de liens hypertextes, notion qui n'a pas été inventée pour structurer l'information, mais pour simplifier le mécanisme de référence dans une page web.

### 3.4 Explorateur de fichiers sous différents système d'exploitation

Pour nous aider à parcourir l'arborescence des fichiers et des dossiers, chaque système d'exploitation offre un navigateur graphique dans son environnement graphique ou la possibilité de

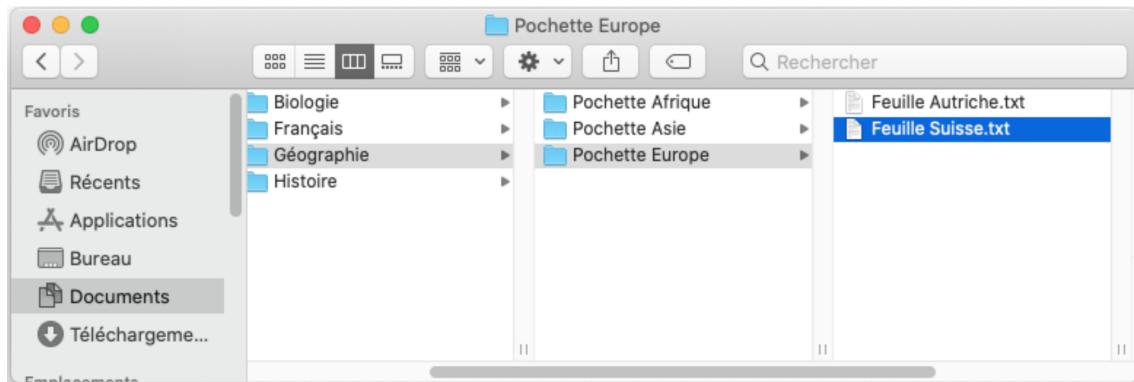


parcourir l'arborescence textuellement dans une console à l'aide de commande.

### 3.4.1 MacOS et GNU/Linux

Sous GNU/Linux et macOS, pour chercher la « Feuille Suisse.txt », on écrirait le parcours depuis la racine de manière suivante :

### /Armoire de classeurs/Géographie/Pochette Europe/

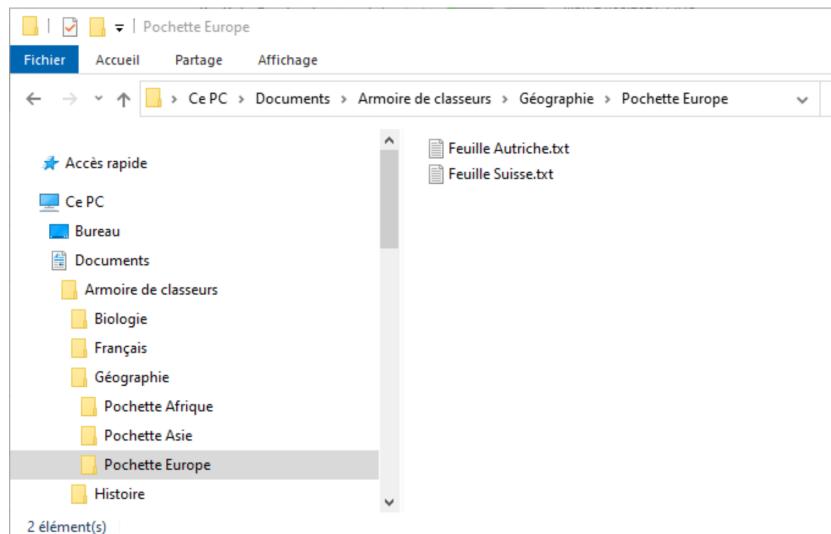


En mode console (textuel), la commande `ls` permet de visualiser le contenu d'un répertoire et la commande `cd nom_du_dossier` permet de se déplacer dans le dossier nommé `nom_du_dossier`.

#### 3.4.2 Microsoft Windows

Sous Microsoft Windows, le chemin serait :

C :\Armoire de classeurs\Géographie\Pochette Europe



Noter la différence entre les deux systèmes. Le caractère / qui délimite les niveaux pour GNU/Linux et macOS et le caractère \ pour Microsoft Windows. Il y a aussi la lettre qui détermine la partition ou un périphérique (clé USB, lecteur DVD, etc.) alors que sous GNU/Linux et macOS, une partition est invisible à l'utilisateur·trice en étant considéré comme une branche de l'arbre. L'utilisation de lettre est une spécificité unique du système d'exploitation Microsoft Windows.

## 3.5 Exercices

1

Se connecter sur [www.eduge.ch](http://www.eduge.ch) puis aller sur le drive. Organiser le drive en sous fichier : Il devra y avoir un dossier principal nommé : Informatique et au moins 3 sous-dossiers appelés : Microbit, Python et Projet.

Le prochain petit programme fait en Microbit devra être sauvegardé sur le drive dans le dossier Microbit.

**2** Dans moodle ou classroom, récupérez le fichier Exercice\_1\_fichier.zip et décompressez-le sur votre ordinateur. Ouvrez le dossier "A\_Trier\_A\_Renommer", observez chaque fichier qu'il contient, renommez-les et déplacez-les dans le bon dossier. Renommez également les dossiers avec des noms corrects.

1. Combien y a-t'il de dossiers dans le dossier Nourriture ?
2. Combien y a-t'il de fichiers dans le dossier Legumes ?
3. Quelle est la taille de l'image avec des carottes ?
4. Quelle est l'extension de l'image avec des carottes ?
5. Quelle est la taille du dossier Fruits ?

**3** \* Choisissez une photo prise avec votre smartphone. Donner quelques informations à son sujet en exploitant les métadonnées qui lui sont associées. Quand et où cette photo a-t-elle été prise ? Sous quel nom et quel format est-elle stockée dans le smartphone ? Quel est son poids ? À quelles informations supplémentaires avons-nous accès ? *Site pour lire les métadonnées : <https://www.verexif.com/fr/>*



# PROGRAMMATION - TYPES ET VARIABLES

## 7.1 Quelques notions de base

Tous les langages de programmation ont été créés dans le même but **traiter de l'information**. Dans un langage impératif, on retrouve toujours les concepts suivants :

- **La notion de valeur** qui représente l'information,
- **La notion d'expression** qui produit une valeur,
- **La notion d'instruction** qui est une commande à exécuter par la machine,
- **La notion de structure de contrôle** qui exécute des instructions d'une manière bien spécifique en fonction des valeurs.

### 7.1.1 La notion de valeur

Une valeur est une représentation de l'information. Par exemple le nombre entier 3 représente une valeur, de même que la chaîne de caractères "Hello world" ou le tableau de nombres à virgule flottante [7.31, 1.74, 3.14, 10.0] (similaire aux nombres décimaux). Une valeur est toujours associée à un type (nombre entier/virgule flottante, chaîne de caractères).

### 7.1.2 La notion d'expression

Des valeurs, des opérateurs mathématiques et des parenthèses peuvent former une expression. Une expression est évaluée et le résultat de cette évaluation donne une valeur.

#### Définition 7.2

Une expression est le résultat d'un calcul effectué par le programme. Elle fournit une valeur associé à un type.

On peut utiliser l'interpréteur python pour évaluer des expressions

Exemple 7.1

```
>>>(1 + 2) * (3 + 4)
21
```

Ici l'expression produit la valeur 21 qui est de type entier.

**7.1.3 La notion d'instruction**

Une instruction peut être vue comme une commande ou un ordre que l'on donne à la machine. En Python, il y a deux types d'instruction :

- affectation : qui effectue un calcul et qui le stocke en mémoire dans une variable.
- fonction : qui est une "commande" prédéfinie.

Exemple 7.2

```
>>> x = 1 + 2      # c'est une affectation
>>> print(x)      # c'est un appel à une fonction
3
```



Tout texte qui suit le caractère dièse "#" est ignoré par Python, on appelle ceci des commentaires. Leur but est d'expliquer le fonctionnement du programme. C'est une bonne habitude de commenter abondamment le code.

Pour utiliser une instruction, par exemple la **fonction print**, il faut lui fournir une valeur (l'information à afficher). Dans l'exemple

```
>>> print("I love computer science")
I love computer science
```

la valeur à afficher est le message "I love computer science". En revanche, la commande

```
>>> print(3 * 5 + 2)
17
```

ne produit pas le même résultat que

```
>>> print("3 * 5 + 2")
3 * 5 + 2
```

car dans le premier cas la valeur est un nombre qui est le résultat de l'évaluation de l'expression  $3 \cdot 5 + 2$  (un nombre entier), et dans le second la valeur est le message "3 \* 5 + 2" qui est une chaîne de caractère (un texte).



Les fonctions sont des blocs d'instructions déjà définis qui font faire quelque chose au programme. La plupart des langages de programmation, par exemple Python, possède toute une série de fonctions prédéfinies qui permettent de réaliser des tâches standard, comme la fonction **print** qui permet d'afficher des messages dans l'interpréteur. L'appel d'une fonction s'effectue en indiquant le nom de la fonction, suivi d'une paire de parenthèses. Ces parenthèses contiennent les éventuels arguments de la fonction, c'est à dire les valeurs nécessaires pour que la fonction puisse être exécutée, séparés par des virgules (exemple : **print ("J'ai ", 16, " ans")**).

### 7.1.4 La notion de structure de contrôle

Les **structures de contrôle** permettent de gérer le flux d'exécution d'un programme, c'est-à-dire l'ordre dans lequel les **instructions** seront exécutées.

## 7.2 La notion de type

Le type d'information qu'une valeur peut encoder varie d'un langage à l'autre. En général, plus le langage est de haut niveau plus il va offrir des types élaborés. La plupart des langages proposent un certain nombre de **types de base**<sup>1</sup>. Ils correspondent aux données qui peuvent être traitées directement par le langage. En Python, les quatre types de base incontournables sont

1. **Les entiers** (`int`) : -4, 0, 99.

→ Ils servent à représenter des nombres de manières exactes.

2. **Les nombres à virgules flottantes** (`float`) : 20.5 , 10. ou 0.001 .

→ Ils servent à représenter des nombres (éventuellement) approximativement, c'est un genre de notation scientifique en puissance de 2 avec un certains nombres de chiffres significatifs (rappel : en notation scientifique en puissance de 10, on écrit  $7000000001 \approx 7,00 \cdot 10^9$  ).

**Attention** : On utilise un point pour séparer la partie entière de la partie décimale (ex : 2.68) et non une virgule !

3. **Les chaînes de caractères** (`str`) : " Hello , World ", ' Oui '.

→ elles servent à représenter du texte, par exemple lorsque l'on veut afficher une information dans l'interpréteur. Elles sont constituées d'une suite de caractères (lettre, chiffre, signe de ponctuation, espace, ...) placées entre guillemets ou (de manière équivalente) entre apostrophes.

4. **Les booléens** (`bool`) : Seulement deux valeurs possibles True (vrai) ou False (faux).

→ Ils servent à représenter *les valeurs de vérité*, par exemple lorsque l'on cherche à évaluer une condition. L'évaluation de l'expression  $7 < 18$  donne la valeur True tandis que le résultat de  $0 > 1$  donne False

Les 4 types présentés ci-dessus sont ceux qu'on retrouve dans quasiment tous les langages de programmation. Python, étant un langage de haut niveau, propose une grande quantité de types de bases (voir la liste complète à [https://fr.wikiversity.org/wiki/Python/Les\\_types\\_de\\_base](https://fr.wikiversity.org/wiki/Python/Les_types_de_base)).

*Pour connaître le type d'une donnée, il suffit de recourir à la fonction `type`*



```
>>> type(10)
<class 'int'>
>>> type(10.0)      # le type float est caractérisé par un point décimal
<class 'float'>
>>> type("10")      # le type string est caractérisé par les guillemets
<class 'str'>
```

1. on dit aussi **types fondamentaux** ou encore **types primitifs**

### 7.3 Les opérations

L'essentiel du travail effectué par un programme consiste à manipuler des données. Peu importe la donnée et son type, ils se ramènent toujours en définitive à une suite finie de bits. Mais alors, pourquoi se compliquer la vie ? Deux raisons,

1. La taille d'une donnée, c'est-à-dire le nombre de bits nécessaire pour la représenter varie en fonction du type de donnée.

Exemple : la chaîne de caractère "c est trop long" nécessite plus de bits que le nombre 8.

2. Il existe des opérations associées à la plupart des types. Ces opérations vont permettre de transformer l'information en opérant sur les valeurs d'entrées du programme.

#### 7.3.1 Opération sur les nombres ( `int` et `float` )

Les opérations sur les nombres sont les suivantes :

opérateur	+	-	*	/	**	Seulement pour les <code>int</code>	
						//	%
signification	addition	soustraction	multiplication	division	exponentiation	Division entière	Modulo

On note que la division entière // et le modulo (le reste de la division entière) sont des opérations seulement définies sur les entiers ( `int` ).

#### Priorité des opérations

Comme en mathématique, les opérateurs ont un ordre de priorité. Ainsi si le calcul  $4+2*3$  correspond à  $4+(2*3)$  parce que la multiplication a priorité sur l'addition. L'ordre de priorité est le suivant :

1. Exponentiation (Puissance)
2. Modulo
3. Multiplication et division entières
4. Addition et soustraction

Sur les opérateurs de même priorité, c'est celui qui est le plus à gauche qui est évalué en premier. Les parenthèses permettent de changer ces priorités. Leurs effet sera de forcer une opération avant une autre. Par exemple, Si on veut d'abord effectuer  $4+2$  et multiplier le résultat par 3, alors il faut l'indiquer avec des parenthèses :  $(4+2)*3$

#### 7.3.2 Opérations sur les booléens et les chaînes de caractère ( `bool` et `str` )

Les opérations avec les booléens et les chaînes de caractère seront traitées dans des chapitres dédiés.

### 7.4 Variables

#### 7.4.1 Le concept de variable

## 7.4. VARIABLES

Le but d'un programme est de transformer de l'information. En général, un programme va commencer par stocker en mémoire les données d'entrée qui seront utilisées lors des étapes de traitement. Pour accéder à ces espaces de stockage et savoir quelle genre d'information ils contiennent, il va falloir leur attribuer un **nom** (ou **identifiant**) ainsi qu'un **type**. Pour finir, leur valeur va varier au fil de l'exécution du programme, c'est pourquoi on les appelle des **variables**. Chaque fois qu'on modifie la valeur d'une variable, on dit qu'on lui **affecte** une nouvelle valeur.



35

### Définitions 7.3

- Une **variable** est un nom associé à un emplacement de la mémoire. C'est comme une boîte que l'on identifie par une étiquette. On dit aussi que le **nom de la variable est son identifiant**.
- Une **affectation** est l'attribution d'une valeur (d'un contenu) à une variable

Pour résumer, une variable est décrite par

- Un **identifiant** unique qui la désigne.
- Un **type** qui définit de quel « genre » est l'information associée.
- Une **valeur** qui doit respecter le type.

### Activité 1

Indiquez le type des variables permettant de stocker (sur votre smartphone) les informations suivantes :

1. le nom d'un contact
2. l'heure du réveil
3. un SMS
4. le code de votre de votre compte eduge
5. le pourcentage affiché de batterie restante
6. la note de votre dernière évaluation de Mathématiques
7. l'énoncé d'un devoir d'Histoire
8. le fait que vous ayez déjà fini (ou non) ce devoir

Il faut imaginer la mémoire de l'ordinateur comme une grosse armoire avec plein de petites boîtes. Certaines de ces boîtes ont un nom (une étiquette) : ce sont des variables, et elles peuvent contenir une valeur. Chaque fois que l'on a besoin de stocker une donnée on va créer une nouvelle variable, en programmation on dira que l'on **déclare** une variable.

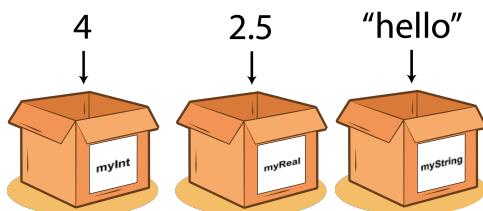
En Python la déclaration d'une variable (et son typage) se fait dynamiquement à l'exécution du programme dès que cette variable apparaît dans une ligne de code. En fait, il suffit d'une **affectation** en utilisant le symbole "`=`". Le code

```
myInt = 4
myString = " hello "
myReal = 2.5
```

crée 3 variables avec comme identifiant `myInt`, `myString` et `myFloat` qui sont de type `int`, `str` et `float` et qui contiennent les données 4, " hello " et 2.5 .

nom		n
"John"		2
	trouve	
	False	
x_pos		k
	10.5	-30

FIGURE 7.1 – Représentation d'une mémoire d'ordinateur comme des boîtes. Lorsqu'une variable est créée la boîte obtient une *étiquette* (en rouge) et une valeur (en bleu).



En Python, les noms de variables doivent en outre obéir à quelques règles simples :

- Il est exclusivement composé de lettres (majuscules et/ou minuscules) et de chiffres mais doit toujours commencer par une lettre.
- On ne peut pas utiliser l'un des 33 mots réservés du langage Python

! and      continueexcept    global    lambda    pass      while  
 as        def        False      if        None      raise    with  
 assert    del        finally     import    nonlocalreturn    yield  
 break     elif      for        in        not        True  
 class    else      from      is        or        try

- Le Seul symbole autorisé est le tiret du bas "`_`" (underscore en anglais). Tous les autres caractères spéciaux sont interdits, en particulier l'espace blanc " " !
- Il est sensible à la casse : les minuscules sont différentes des MAJUSCULES. Exemple : les identifiant `Age` et `age` vont désigner des variables différentes.



*Il est très important de donner un nom clair et précis aux variables. En lisant le nom de la variable, il faudrait avoir une idée de ce qu'elle représente. Par exemple, si je lis `note_eleve_JohanS`, je me dis que cette variable représente la note de l'élève Johan S. et donc c'est probablement un `float` entre 0.0 et 6.0. Pour améliorer la lecture, on utilise le seul symbole autorisé l'underscore `_`*

### 7.4.2 L'affectation

Chaque variable (déclarée) possède une valeur qui est l'information qu'elle porte. Par exemple, la valeur de la variable `nombre_eleves_GR103` pourrait être 23, celle de la variable `note` pourrait être 4.5, celle de la variable `prenom` pourrait être "Anouk". En Python, pour définir ou modifier la valeur d'une variable, il suffit de lui affecter une valeur en utilisant le symbole égal "`=`".

#### Définition 7.4

**L'affectation** d'une valeur à une variable est l'instruction basique qui permet d'attribuer une valeur à une variable. Par abus de langage, on dit souvent «affectation de variable» ou «assignation de variable».

En Python, l'affectation s'effectue avec l'opérateur `=`. L'instruction d'affectation est composée d'une variable, suivie du symbole `=` et d'une expression à évaluer. Après avoir exécutée une affectation la variable prendra la valeur de l'expression évaluée.

#### Exemple 7.3

```
>>> x = (1 + 2) * (3 + 4)
>>> print(x)
21
```



- Il ne faut pas confondre l'opérateur `=` (l'affectation) et l'opérateur `==` qui sert à tester l'égalité de deux valeurs. L'opérateur `==` est beaucoup plus proche du concept d'égalité utilisée en mathématiques.
- Il est important de distinguer les expressions, comme `x + 2`, des instructions, comme `y = x + 2`; . Une expression se calcule, une instruction s'exécute.

### 7.4.3 Différents types d'affectations

#### Exemples 7.4

##### 1. Affectation d'une quantité

---

```
note = 37 * 50 + 1
```

---

**Explications :** La valeur  $37 * 50 + 1$  (c'est-à-dire  $47$ ) est injectée dans la case mémoire nommée note .

##### 2. Affectation d'un texte

---

```
prenom = "Robert"
nom_prenom = "Dupont"+'Robert'
```

---

**Remarque :** Un Chaîne de caractères peut être écrite soit entre "guillemets", soit entre 'apostrophes' droits.

##### 3. Affectations parallèles

---

```
n, x = 1, 7.3
```

---

**Explications :** C'est équivalent à effectuer les affectations  $n = 1$  et  $x = 7.3$  simultanément.

##### 4. Auto-affectation

---

```
k = k ** 2
t = 3 * t - 2
```

---

**Explications :** Une expression avec la valeur de la variable est calculée puis réaffectée à la variable elle-même.

Par exemple, si t contient la valeur 5 alors après avoir exécuté  $t = 3 * t - 2$  t vaudra  $3 * 5 - 2$  (c'est-à-dire 13).

##### 5. Incrémentation ou décrémentation

---

```
k = k + 1
compteur = compteur - 2
```

---

**Remarque :** Auto-affectation additionnée ou soustraite d'un certain pas (d'un certain nombre) entier.

# PROGRAMMATION - CHAÎNES DE CARACTÈRES

## 9.1 Introduction

Les chaînes de caractères (type `str`) en Python sont des séquences de caractères qui peuvent être utilisées pour stocker des textes. Les chaînes de caractères sont définies entre apostrophes ('), guillemets simples (""), ou guillemets triples ("'''").

Listing 9.1 – Chaînes de caractères

---

```

1 str1 = "Ceci me permet d'écrire l'apostrophe."
2 str2 = 'Ceci me permet de "placer" le guillemet.'
3 str3 = """ Ceci me permet d'écrire
4 sur plusieurs lignes """

```

---

## 9.2 Boite à outils

### 9.2.1 Opérateurs

L'opérateur `+` permet de concaténer des chaînes de caractères. L'opérateur `*` permet de répéter une chaîne de caractères plusieurs fois.

Listing 9.2 – Opérateurs

---

```

1 chaine1 = "Bonjour"
2 chaine2 = " toi !"
3 chaine3 = chaine1 + chaine2
4 print(chaine3)                      # affiche "Bonjour toi !"
5
6 chaine4 = "Coucou"
7 chaine5 = chaine4 * 3
8 print(chaine5)                      # affiche "Coucou Coucou Coucou"

```

---

### 9.2.2 Fonction `len()`

En Python, la `len()` fonction intégrée peut être utilisée pour déterminer la longueur d'un objet. Il peut être utilisé pour calculer la longueur de chaînes, de listes, d'ensembles et d'autres objets dénombrables.

Listing 9.3 – Fonction len()

---

```

1 longueur = len("Hello")
2 print("La longueur : ", longueur) # affiche 5

```

---

### 9.2.3 in

La syntaxe `in` est utilisée pour déterminer si une lettre ou une sous-chaîne existe dans une chaîne.

Elle renvoie `True` si une correspondance est trouvée, sinon `False` est renvoyée.

Listing 9.4 – in

---

```

1 jeu = "Popular Nintendo Game: Mario Kart"
2
3 if "l" in jeu:
4     print("l est dans la chaîne jeu .")
5 else:
6     print("l n'est pas dans la chaîne jeu .")

```

---

### 9.2.4 Indexation et découpage des chaînes

Un seul caractère peut être accédé avec la notation entre crochets `[index]`, ou une sous-chaîne peut être accédée en utilisant le découpage `[start:end]`.

L'indexation avec des nombres négatifs compte à partir de la fin de la chaîne.

Listing 9.5 – indexation

---

```

1 mot = 'orange'
2 #      0 1 2 3 4 5
3
4 print(mot[0])      # => 'o'
5 print(mot[1])      # => 'r'
6 print(mot[4:6])    # => 'ge'
7 print(mot[:4])     # => 'oran'
8 print(mot[-1])     # => 'e'

```

---

### 9.2.5 Itérer la chaîne

Pour parcourir une chaîne en Python, la notation `for ... in` est utilisée.

Listing 9.6 – iteration

---

```

1 mot = "hello"
2 for c in mot:
3     print(c)

```

---

affiche chaque lettre du mot `"hello"` les unes après les autres.

### 9.2.6 Autres fonctions

`.lower()` renvoie une chaîne avec tous les caractères majuscules convertis en minuscules.

`.upper()` renvoie la chaîne avec tous les caractères minuscules convertis en majuscules.

Listing 9.7 – lower / upper

---

```

1 salutation = "Bienvenue chez Chili's"
2 print(salutation.lower()) # affiche bienvenue chez chili's

```

---

.`isalpha()` renvoie `True` si tous les caractères de la chaîne sont alphabétiques et qu'elle contient au moins un caractère, sinon `False`.

.`isdigit()` renvoie `True` si tous les caractères de la chaîne sont des chiffres et qu'elle contient au moins un caractère, sinon `False`.

Listing 9.8 – `isalpha / isdigit`

---

```

1  texte = "LeGrandParc"
2  if texte.isalpha():
3      print("Contient que des lettres")
4  else:
5      print("Contient d'autres caractères")
6 # affiche Contient que des lettres

```

---

### 9.3 Exercices

- 1** Écrire un programme qui demande à l'utilisateur-trice de saisir une phrase et qui affiche la longueur de cette phrase.
- 2** Écrire un programme qui demande à l'utilisateur-trice de saisir deux mots et qui dit si les deux mots sont les mêmes.
- 3**
  - a) Écrire un programme qui demande à l'utilisateur-trice de saisir une phrase et qui affiche les 10 premiers caractères de cette phrase.
  - b) Modifier le code pour demander également à l'utilisateur-trice le nombre de caractères à afficher.
- 4** Afficher le menu d'un programme jusqu'à ce que l'utilisateur-trice saisisse "q" pour quitter.  

```

-- Menu --
1. Option 1
2. Option 2
q. Quitter
Choisissez une option :

```
- 5**
  - a) Écrire un programme qui demande à l'utilisateur-trice de saisir une phrase et dit si elle contient la lettre "a".
  - b) Écrire un programme qui demande à l'utilisateur-trice de saisir une phrase et qui compte le nombre de lettres "a".
  - c) Écrire un programme qui demande à l'utilisateur-trice de saisir une phrase et qui compte le nombre de voyelles.
- 6** Écrire un programme qui demande à l'utilisateur-trice de saisir une phrase et qui affiche cette phrase sans les espaces.
- 7** Écrire un programme qui demande à l'utilisateur-trice de saisir un mot et qui affiche ce mot à l'envers.