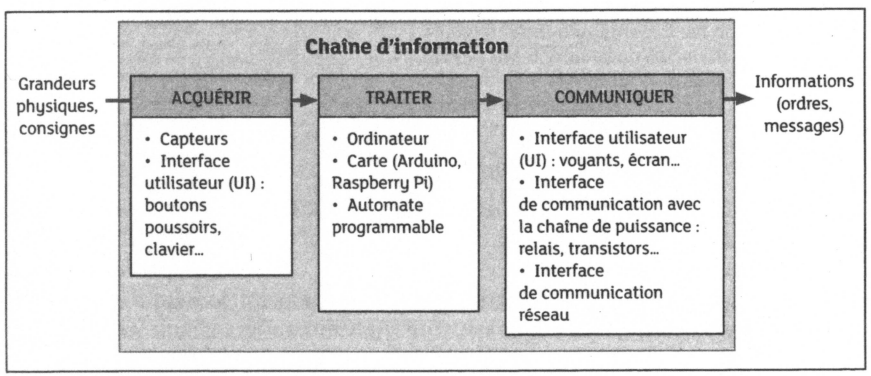
4 Acquisition et traitement de l'information

1 La chaîne d'information

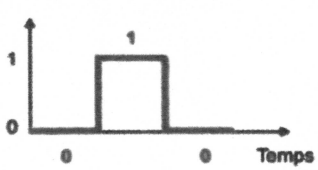
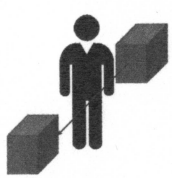
Pour réagir et s'adapter convenablement à son environnement, un système automatisé a besoin d'acquérir des informations grâce à des capteurs qui lui permettent, en fonction des besoins, de voir, d'entendre, de détecter les contacts, de savoir dans quelle position il se trouve...
La grandeur physique acquise par les capteurs est convertie en un signal qui peut ensuite être traité.

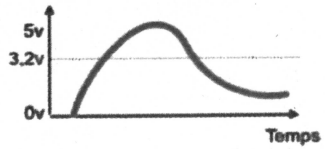
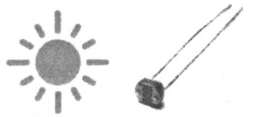
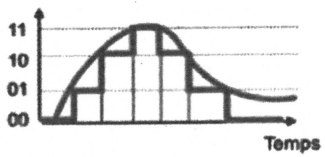


2 Les caractéristiques des signaux

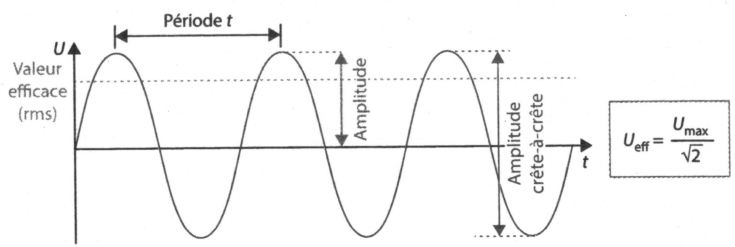
2.1. La nature des informations

Les informations peuvent circuler dans le système sous 3 formes différentes : logique, analogique, numérique.

Nature de l'information	Définition	Exemples
Information logique 	Un signal est dit logique si la grandeur de l'information ne peut prendre que deux valeurs. *Exemples* : « Tout ou Rien » (TOR) ; « 0 ou 1 » ; « high ou Low ».	Lorsque l'on utilise un capteur de présence, le fait de détecter, ou non, un objet se traduit par deux niveaux de tension différents. 

Nature de l'information	Définition	Exemples
Information analogique 	Un signal est analogique si la grandeur de l'information peut varier de façon continue dans le temps. *Exemple* : une variation de température, de luminosité, de vitesse.	Lorsque l'on utilise un capteur de luminosité, la variation de lumière va se traduire par une variation de tension à ses bornes. 
Information numérique 	Un signal numérique se compose d'une suite d'informations logiques, « 0 » et « 1 », qui représentent des nombres. Les signaux numériques ne peuvent prendre qu'un nombre fini de valeurs. Pour obtenir un signal numérique, il faut numériser une information.	Lorsque l'on utilise un codeur de position angulaire, chaque segment du disque en rotation contient des encoches qui laissent passer ou bloquent le faisceau lumineux. L'information reçue est donc une suite de 0 et de 1 qui traduisent la position angulaire du disque. 

2.2. La caractérisation d'un signal analogique périodique



• La période

La période représente la durée au bout de laquelle le signal se répète. Elle s'exprime en secondes (s).

• La fréquence

La fréquence correspond au nombre de répétitions d'un phénomène périodique par seconde. La fréquence est l'inverse de la période : $f = \frac{1}{t}$. Elle s'exprime en hertz (Hz).

• La valeur efficace

La tension efficace (U_{eff}) d'une tension alternative est la valeur de la tension continue qui produirait le même effet. On mesure une tension efficace avec un voltmètre.

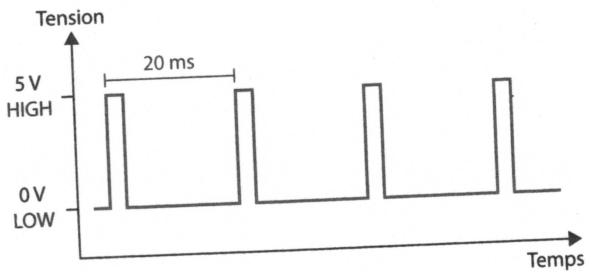
$$U_{eff} = \frac{U_{max}}{\sqrt{2}}.$$

2.3. La caractérisation des signaux numériques

On distingue deux niveaux :
– le niveau haut : H (*High*), NL1, 1 ;
– le niveau bas : L (*Low*), NL0, 0.

• Le chronogramme

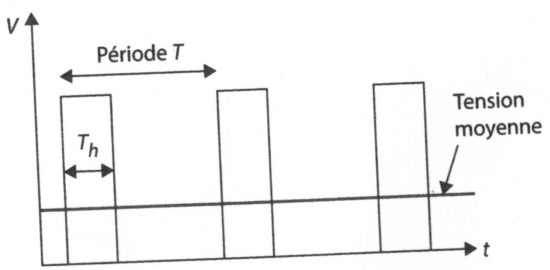
Le chronogramme est un diagramme utilisé pour représenter les signaux en fonction du temps.



• La modulation par largeur d'impulsion (MLI) ou *Pulse Width Modulation* (PWM)

L'information utile est le temps au niveau haut par rapport à la période.

• Le rapport cyclique



Le rapport cyclique (en %) correspond au temps au niveau haut du signal, divisé par la période :

$$\alpha = \frac{T_h}{T}.$$

Pour exprimer le rapport cyclique en pourcents (%), il faut multiplier la valeur de celui-ci par 100 :

$$\alpha = \frac{T_h}{T} \times 100.$$

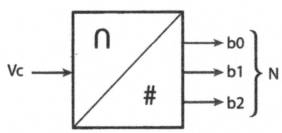
Calcul de la tension moyenne : $U_{moyenne} = U_{max} \times \frac{T_h}{T}$.

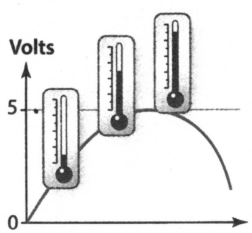
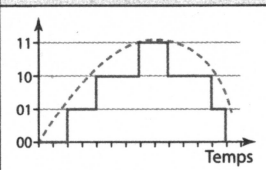
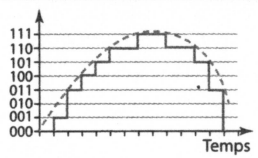
Exemples d'utilisation ➤
• Pilotage des servomoteurs analogiques.
• Pilotage de la vitesse des moteurs à courant continu.

3 La conversion analogique-numérique (CAN)

Lorsque le signal délivré par un capteur est un signal analogique variable au cours du temps (*exemple* : une tension), ce signal doit être converti en un signal numérique pour pouvoir être traité par la carte électronique.

Un **convertisseur analogique-numérique (CAN)** (ou DAC pour *Digital Analog Converter*) est un dispositif électronique permettant la conversion d'un signal analogique en un signal numérique.



Signal analogique	Signal numérique codé sur 2 bits	Signal numérique codé sur 3 bits
	 **Soit 4 valeurs possibles : de 0 à 3**	 **Soit 8 valeurs possible : de 0 à 7** **Exemple** : 100 en binaire correspond à 4 en decimal.

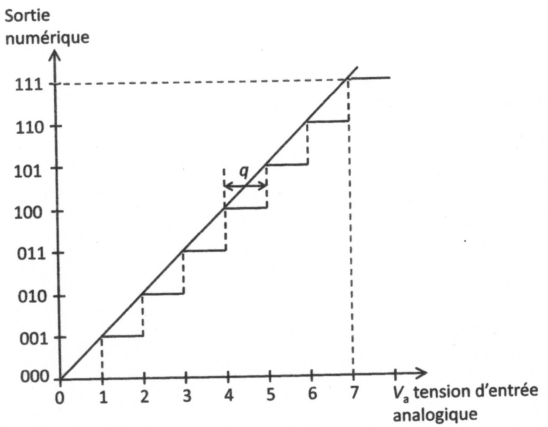
Puissance de 2	2^1	2^0
Décimal	2	1
0	0	0
1	0	1
2	1	0
3	1	1

Puissance de 2	2^2	2^1	2^0
Décimal	4	2	1
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Le **quantum** (q) du convertisseur est la variation minimale du signal d'entrée qui provoque la variation d'une unité de la donnée numérique de sortie.
Le quantum s'exprime dans l'unité de la grandeur analogique d'entrée. Il dépend de l'amplitude du signal d'entrée et du nombre de bits du convertisseur.

$$q = \frac{\text{Amplitude du signal d'entrée}}{2^n}$$

avec n le nombre de bits du convertisseur.
Plus le quantum est petit et plus le signal numérisé est fidèle au signal analogique de départ.



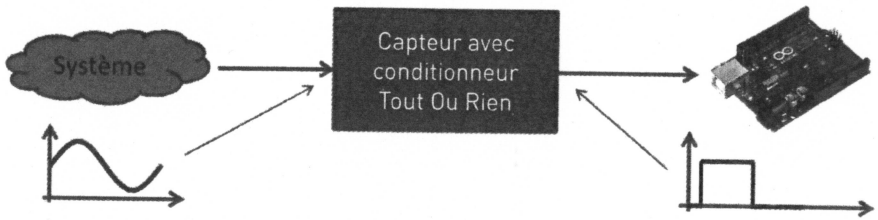
4 Les capteurs

4.1. Les caractéristiques des capteurs

Pour choisir un capteur, différents critères sont à prendre en compte :
– **l'étendue de la mesure** : valeurs extrêmes pouvant être mesurées par le capteur ;
– **la résolution** : plus petite variation de grandeur mesurable par le capteur ;
– **la sensibilité** : variation du signal de sortie par rapport à la variation du signal d'entrée (exemple : un capteur de température de sensibilité 10 mV/°C) ;
– **la précision** : aptitude du capteur à donner une mesure proche de la valeur vraie.

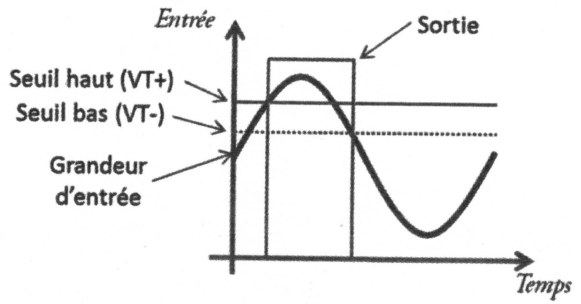
4.2. Les capteurs « tout ou rien » (TOR)

Les **capteurs « tout ou rien » (TOR)** délivrent une information binaire : « 0 » ou « 1 ».
Une valeur de seuil est choisie. Lorsque la grandeur d'entrée est inférieure au seuil, la sortie du capteur est à 0, lorsque la grandeur d'entrée est supérieure au seuil, la sortie du capteur est à 1.



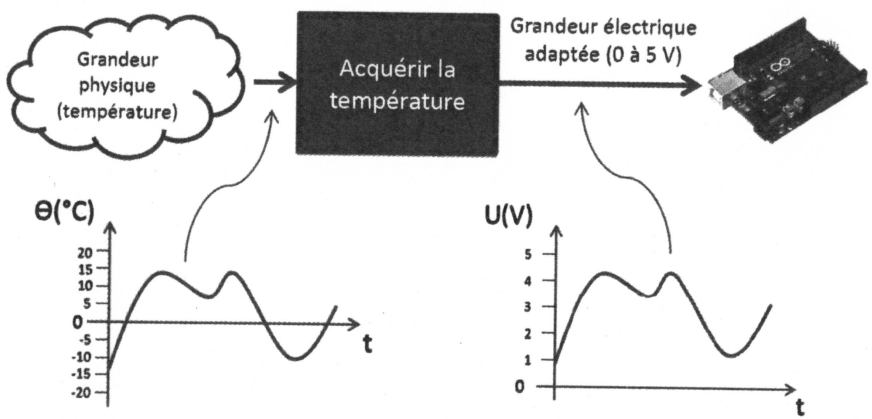
Dans la pratique, le capteur TOR possède deux seuils distincts afin d'éviter que la sortie ne devienne instable lorsque l'entrée est très proche du seuil.

- Pour que la sortie du capteur passe à l'état haut, il faut que la grandeur d'entrée passe au-dessus du seuil haut.
- Pour que la sortie du capteur passe à l'état bas, il faut que la grandeur d'entrée passe en dessous du seuil bas.

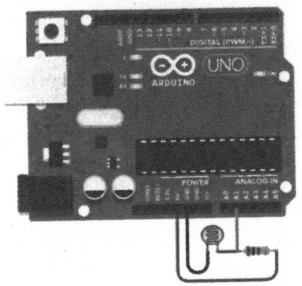


4.3. Les capteurs proportionnels analogiques

Les **capteurs proportionnels analogiques** fournissent un signal proportionnel à la grandeur mesurée.



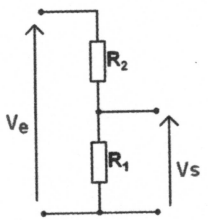
Exemple ➤ **La photorésistance**



Une photorésistance est un capteur dont la résistance (en ohm) varie en fonction de l'évolution de la luminosité.
Ce type de capteur est dit passif car il a besoin d'être inséré dans un montage électrique. L'information récupérée est la variation de la tension U (en volt) aux bornes du capteur.
Un montage fréquemment utilisé est le pont diviseur de tension.
Sur le montage, nous pouvons observer que :
– la photorésistance et la résistance sont alimentées par la carte Arduino, en 5 V ;
– l'information de la tension (image de la luminosité) arrive sur une broche analogique (A1) de la carte Arduino.

Pour calculer la tension aux bornes de la résistance R_1 qui, dans notre exemple est la photorésistance, il faut connaître la tension d'alimentation V_e et la résistivité des résistances R_1 et R_2.

On applique la loi d'Ohm :
$V_e = (R_1 + R_2) \times i$ donc $i = \dfrac{V_e}{R_1 + R_2}$.

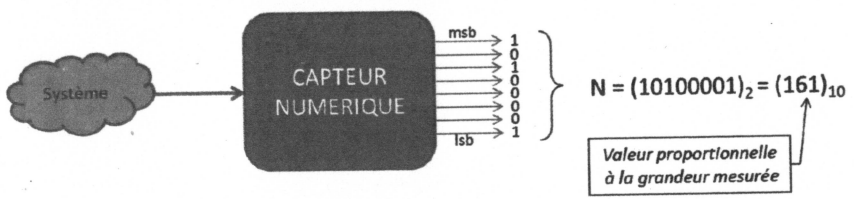


On applique une deuxième fois la loi d'Ohm :

$V_s = R_1 \times i$ donc $V_s = R_1 \times \dfrac{V_e}{R_1 + R_2}$.

4.4. Les capteurs proportionnels numériques

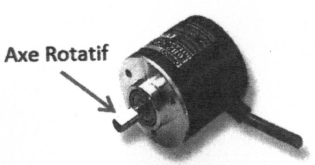
Les **capteurs proportionnels numériques** fournissent des signaux numériques proportionnels à la grandeur à mesurer.



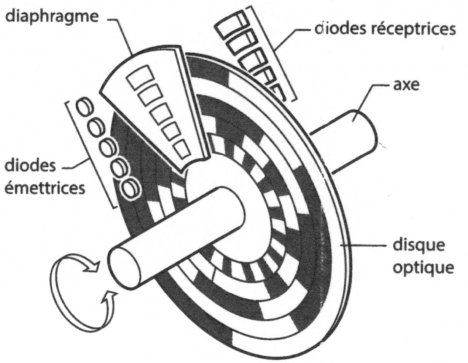
Exemples ➤

• **Le codeur absolu**

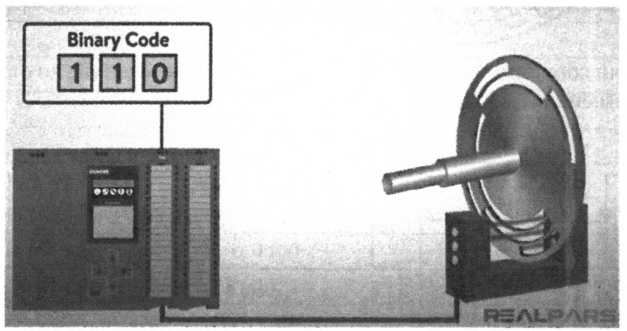
Un codeur absolu délivre la position angulaire d'un axe en rotation.
Chaque segment angulaire possède une valeur de position unique, ou mot de données, qui est codée sur un disque qui tourne avec l'axe. Le nombre de codes uniques sur le disque détermine la précision avec laquelle la position peut être exprimée. Le codeur lit le code à l'aide d'un capteur optique, capacitif ou magnétique.



Dans le cas d'un capteur optique, le disque, solidaire de l'axe, comporte des motifs : segments opaques et segments transparents.
D'un côté du disque, des diodes émettent de la lumière, celle-ci passe à travers les segments transparents et est bloquée par les segments opaques.
De l'autre côté du disque, les diodes réceptrices reçoivent ou non la lumière.

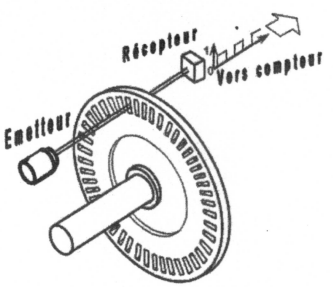


Ces signaux constituent le mot de données.

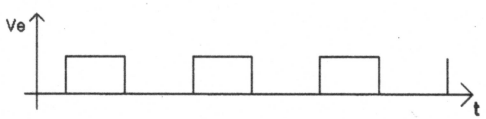


• **Le codeur rotatif incrémental (ou fourche optique)**

Le codeur rotatif incrémental ajoute ou soustrait (selon le sens de rotation) une unité à un compteur à chaque rotation supérieure à la résolution du capteur.
Le codeur optique dispose d'un disque avec de nombreuses fenêtres (le nom technique est « tic »). Ces fenêtres laissent passer la lumière. Une LED dans le codeur émet un faisceau lumineux qui tente de passer au travers du disque, lorsqu'une fenêtre se présente, la lumière passe et est captée par un récepteur.



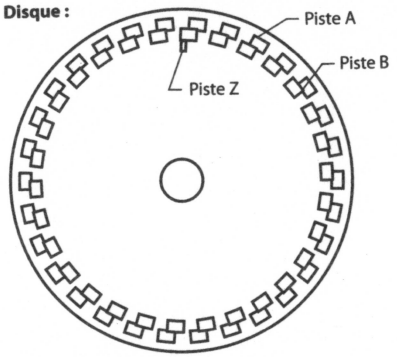
Quand le récepteur, qui est un phototransistor, capte de la lumière, il délivre un signal logique. Le niveau haut correspond au fait que la lumière passe et le niveau bas au fait que la lumière ne passe pas.
Ce type de signal permet, connaissant la résolution du codeur, de calculer l'angle de rotation et/ou la vitesse de rotation du disque.



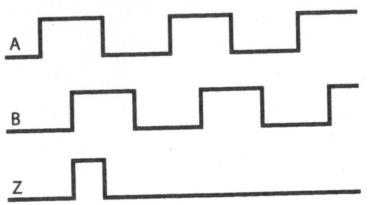
Détermination du sens de rotation

Pour pouvoir déterminer le sens de rotation du disque, on met une seconde piste de tics en dessous de la première mais décalée de 25 % par rapport à la première.

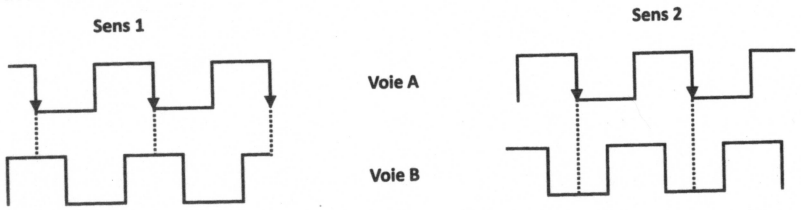
Disque :



Signaux délivrés :



Pour connaître le sens de rotation, on analyse, par exemple au moment d'un front descendant sur la voie A, si la voie B est à 1 ou à 0.

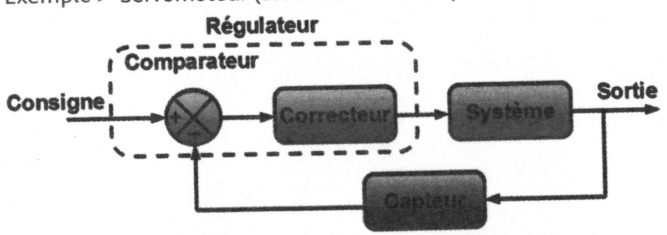


5 Les systèmes asservis

Un **système asservi ou régulé** (*exemple* : régulateur de vitesse sur une voiture), compare en permanence ce qu'il fait à ce qu'il doit faire, grâce à un capteur. Si un écart est détecté, le système tente de le corriger.

Le schéma bloc d'un système asservi possède forcément une boucle de retour avec un capteur.

Exemple ➤ Servomoteur (asservissement en position).



6 Les bases de numération

En fonction des besoins, on utilise 3 bases :

– la base 10 que nous utilisons dans la vie de tous les jours ;

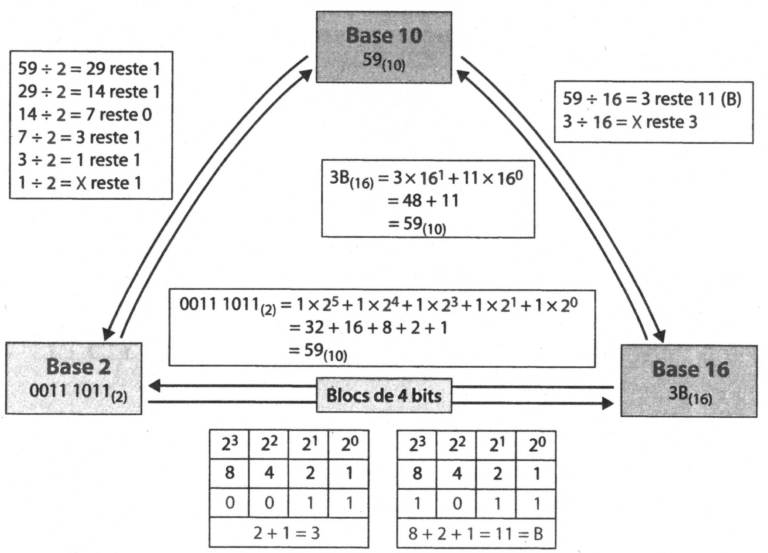
– la base 2 utilisée par les ordinateurs ;

– la base 16 qui permet de compacter les mots écrits en base 2 pour les rendre plus lisibles par les humains.

Le tableau ci-dessous permet de lire directement comment sont écrits les 16 premiers nombres dans les 3 bases.

$(Décimal)_{10}$	$(Binaire)_2$	$(Hexadécimal)_{16}$
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Voici, sur un exemple, les méthodes pour passer d'une base à l'autre.



• **Notation**

Lorsqu'on écrit un nombre, il faut spécifier dans quelle base il est écrit, sinon il y a des risques de confusion.

Par convention, on indique la base en indice :

$3A9_{(16)} = (3A9)_{16}$ → base 16 : hexadécimal

$238_{(10)} = (238)_{10}$ → base 10 : décimal

$0100\ 1101_{(2)} = (0100\ 1101)_2$ → base 2 : binaire

En programmation, la notation permettant de voir que l'on est en hexadécimal dépend du langage.

Exemple ➤ avec $(AE4F)_{16}$:

Langage	Préfixe	Exemple
C, C++, Java	0x	0xAE4F
Pascal	$	$AE4F
Basic	&H	&HAE4F
HTML	#	#AE4F

7 Les codages

7.1. Le code ASCII

Le code ASCII (*American Standard Code for Information Interchange*) est principalement utilisé pour la transmission en série des caractères alphanumériques (chiffres, lettres, caractères spéciaux) dans les systèmes informatiques.

Chaque caractère en code ASCII est codé sur 8 bits, donc sur 2 caractères exprimés en hexadécimal.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STH	ETH	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	CD2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL
8	€	□	‚	ƒ	„	…	†	‡	ˆ	‰	Š	‹	Œ	□	Ž	□
9	□	'	'	"	"	•	–	—	˜	™	š	›	œ	□	ž	Ÿ
A		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
B	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Exemple ➤ Le caractère « A » est codé en hexadécimal par 41.

7.2. L'Unicode

Pour coder de manière universelle l'ensemble des symboles utilisés, quelle que soit la langue (français, anglais, grec, chinois...), il faut attribuer à tout caractère ou symbole un nom et un identifiant numérique, et ce de manière unifiée. C'est ce que propose la norme Unicode.

Pour stocker sur un support informatique un texte constitué de caractères Unicode, il faut utiliser un procédé appelé **processus d'encodage**.

Actuellement un des systèmes d'encodage couramment utilisé est UTF-8.

Caractère	Encodage	Bits
A	UTF-8	01000001
A	UTF-16	00000000 01000001
A	UTF-32	00000000 00000000 00000000 01000001

8 Les algorithmes et les programmes

Lorsque l'on a un problème à résoudre, on commence par identifier les étapes successives qui vont nous amener à sa résolution. Le fait de noter ces étapes successives constitue un **algorithme**.

Lorsque ce même problème doit ensuite être résolu par un robot, un ordinateur ou une carte électronique, cela nécessite d'écrire les étapes dans un langage spécifique compréhensible par la machine. Cette succession d'étapes s'appelle un **programme informatique**.

Analyse → **Algorithme** → **Programme**

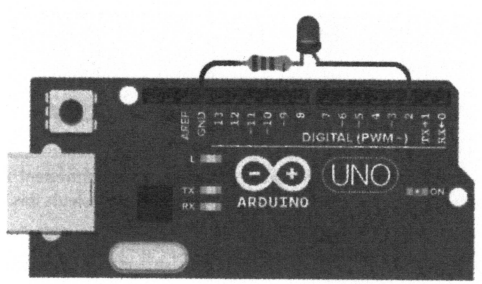
Définir le problème — Définir la méthode amenant à la solution du problème — Transcrire dans le langage de programmation

8.1. Le pseudo-code

En programmation, le **pseudo-code**, également appelé LDA (langage de description d'algorithmes) est une façon de décrire un algorithme en langage presque naturel, sans référence à un langage de programmation en particulier.

Exemple ➤ Faire clignoter une DEL

Lorsque l'on souhaite faire clignoter une DEL (diode électroluminescente), l'écriture du pseudo-code peut, dans un premier temps, faciliter l'identification de la succession des étapes à programmer. Le code, dans la colonne de droite, est celui qu'il faudrait écrire sur Arduino pour faire clignoter la diode, branchée sur la broche 2.

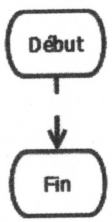
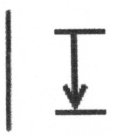
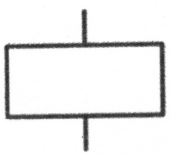
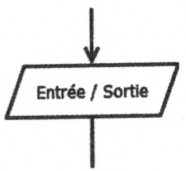
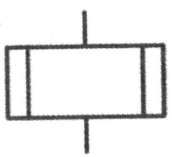
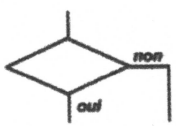
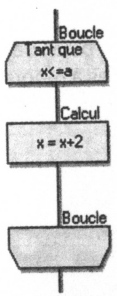
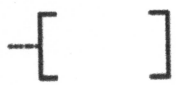


Pseudo-code	Code

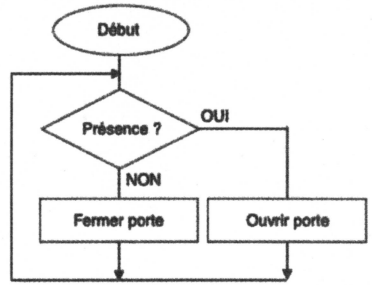
Début
Allumer la DEL branchée sur la broche 2
Attendre 1 seconde
Éteindre la DEL
Attendre 1 seconde
Recommencer
Fin
	`void setup() {`
	`  pinMode(2, OUTPUT);`
	`}`
	`void loop() {`
	`  digitalWrite(2, HIGH);`
	`  delay(1000);`
	`  digitalWrite(2, LOW);`
	`  delay(1000);`
	`  }`

8.2. L'algorigramme (ou logigramme)

Un algorithme peut être représenté sous forme graphique en utilisant un **algorigramme**, aussi appelé **logigramme**.
Pour construire un algorigramme, on utilise des symboles normalisés.

			
Début / Fin Il n'y a qu'un seul « Début » mais il peut y avoir plusieurs « Fin ».	Les liaisons, avec ou sans flèche, permettent de guider la lecture de l'algorigramme.	Les instructions permettent de réaliser les traitements (calculs, incrémentation de valeurs, ordre de fonctionnement...).	Lecture ou écriture d'une donnée.
			
Les sous-programmes permettent d'améliorer la lisibilité du programme principal.	Les tests consistent à poser une question à laquelle la réponse est soit « oui », soit « non ».	Les boucles permettent d'exécuter plusieurs fois une portion du programme.	Les commentaires permettent d'aider à la compréhension du programme.

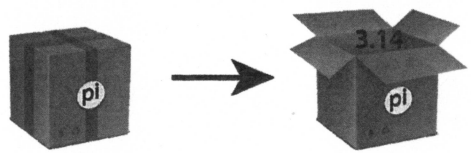
Exemple ➤

Pseudo-code	Algorigramme
Début **Si** Présence **Alors** ouvrir porte **Sinon** fermer porte **Retour** au début	

9 Les constantes et les variables

9.1. Les constantes

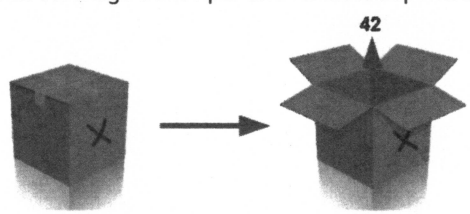
Une **constante** est une donnée qui ne change pas au cours du programme.
On utilise des constantes pour augmenter la lisibilité des programmes et pour en faciliter l'écriture.



Exemple ➤ Pi = 3,1415926535

9.2. Les variables

Une **variable** est le nom donné à un espace de stockage d'une donnée qui va être amenée à changer au cours du programme.
La taille de l'espace de stockage utilisé par une variable dépend du type de variable.



On peut faire une analogie avec une boîte de rangement :
– lorsque l'on crée une variable, on lui donne un nom, c'est le nom de la boîte ;
– on définit son type : c'est la taille de la boîte ;
– on associe une valeur à la variable : c'est comme si on rangeait la valeur dans la boîte ;
– on peut ensuite lire, modifier et utiliser la valeur associée à la variable, en y faisant appel par l'intermédiaire du nom de la variable.

9.3. Les types de variables

Le type de variables détermine le type de manipulations que l'on peut faire.

Exemple ➤
- Avec les nombres entiers, on peut effectuer des opérations mathématiques.
- Avec une suite de caractères, on peut transformer le texte en remplaçant certains mots ou en mettant des caractères en majuscules.
- Avec un tableau (*array*), on peut ajouter ou enlever des éléments.

Présentation de quelques types de variables fréquemment utilisées :

Les entiers relatifs	*int*	Les entiers relatifs (« *integers* », en anglais). *Exemples* : –323 ; 0 ; 7 ; 42.
Les entiers naturels	*unsignedint*	Les entiers naturels (« *unsigned integers* », en anglais) sont positifs ou égaux à zéro. *Exemples* : 0 ; 3 ; 15 000.
Les booléens	*boolean*	Les booléens sont des variables logiques, codées sur 1 bit, qui ne peuvent prendre que deux valeurs : « 0 » ou « 1 ». Ils sont parfois notés « *True* » et « *False* » ou « *High* » et « *Low* ».
Les nombres décimaux	*float*	Les nombres décimaux (nombres à virgule). *Exemples* : –5,3 ; 0,007 ; 880,559. Attention ! Dans la plupart des langages, il faudra utiliser « . » au lieu de « , » pour séparer la partie entière de la partie décimale (norme anglosaxonne).
Les chaînes de caractères	*string*	Les chaînes de caractères sont utilisées pour représenter du texte (des mots, des phrases, etc.).
Les tableaux	*array*	Les **tableaux** (« *array* », en anglais) sont utilisés pour créer des listes avec des indices qui permettent d'identifier les différents éléments de la liste. En général, la première valeur a l'indice « 0 ». *Exemple* : monTableau = ['bleu', 'rouge', 'vert'] ; Dans le tableau, appelé « monTableau », l'élément d'indice 1 est la chaîne de caractères 'rouge'.
Les objets	*object*	Les objets (« *object* », en anglais) sont des conteneurs qui peuvent inclure tous types de données, y compris des sous-objets, des variables (les propriétés des objets), ou des fonctions (les méthodes associées aux objets).

9.4. La portée des variables

La **portée des variables** sert à désigner les différents espaces dans le programme dans lesquels une variable est accessible, c'est-à-dire utilisable.

• Variables locales

Les variables définies dans une fonction sont appelées variables locales. Elles ne peuvent être utilisées que localement, c'est-à-dire uniquement à l'intérieur de la fonction qui les a définies.

• Les variables globales

Les variables définies dans l'espace global du script, c'est-à-dire en dehors de toute fonction, sont appelées des variables globales. Ces variables sont accessibles (ce qui signifie utilisables) à travers l'ensemble du programme.

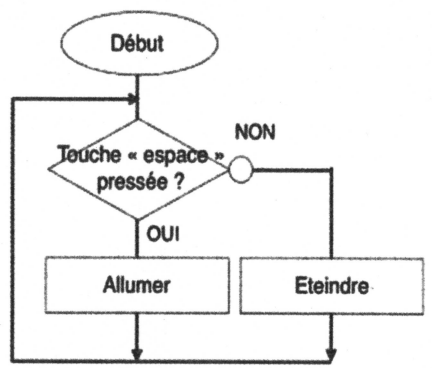
10 Les opérateurs

Les opérateurs permettent de manipuler ou comparer des valeurs, notamment des variables. Parmi les opérateurs utilisés fréquemment, on identifie :
– les opérateurs **arithmétiques** : addition, soustraction, multiplication, division ;
– les opérateurs de **comparaison** : égal à, différent de, supérieur à, inférieur à ;
– les opérateurs **logiques (booléens) : ET** (il faut que toutes les entrées soient à 1 pour que la sortie le soit), **OU** (il faut au moins une entrée à 1 pour que la sortie soit à 1), **NON** (si l'entrée est à 1, la sortie est à 0 et inversement).

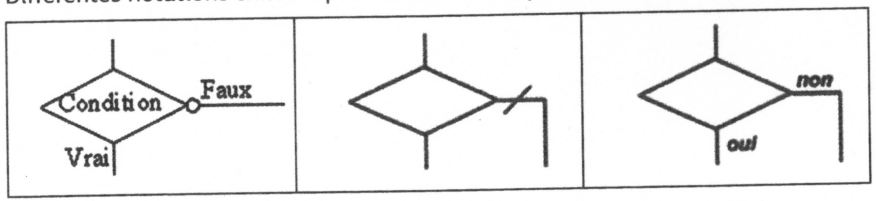
Opérateurs arithmétiques	Opérateurs de comparaison
• = (égalité) • + (addition) • – (soustraction) • * (multiplication) • / (division) • % (modulo)	• == (égal à) • != (différent de) • < (inférieur à) • > (supérieur à) • <= (inférieur ou égal à) • >= (supérieur ou égal à)
Opérateurs booléens	**Opérateurs composés**
• && (ET booléen) • II (OU booléen) • ! (NON booléen)	• ++ (incrémentation) • -- (décrémentation)

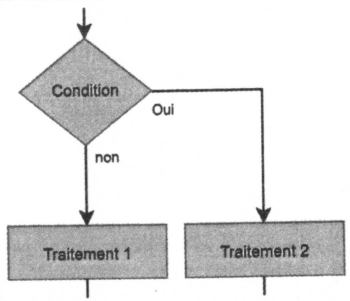
11 Les structures de contrôle (Si... Alors... Sinon)

Les structures de contrôle permettent d'exécuter seulement certaines instructions d'un programme en fonction d'une condition.

Pseudo-code	Algorigramme
Début **Si** touche « espace » pressée **Alors** allumer **Sinon** éteindre **Fin Si** **Retour début**	

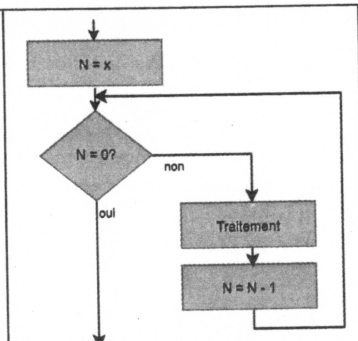
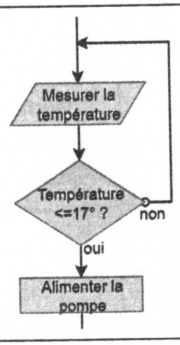
Différentes notations existent pour identifier la réponse au test :



Structure alternative ou conditionnelle Une structure alternative n'offre que deux issues possibles s'excluant mutuellement. Une condition est testée, et, en fonction du résultat du test, soit le traitement 1, soit le traitement 2 est réalisé.		**SI « Condition » vraie** **Alors** **Faire « Traitement 2 »** ... **Sinon** **Faire « Traitement 1 »** ...

12 Les boucles (Tant que... Faire...)

Une **boucle**, aussi appelée itération, permet de faire de multiples fois quelque chose jusqu'à ce qu'une condition soit vraie.

Boucle avec comptage On initialise la variable N avec une valeur x. On teste si N est égal à 0, si ce n'est pas le cas, on exécute le traitement et on décrémente la variable N puis on teste à nouveau la variable N, et ainsi de suite jusqu'à ce que N = 0.		**Pour N = x jusqu'à ce que N = 0** **Répéter** **« Traitement »** **Fin Pour**
Boucle conditionnelle Tant que la condition testée est fausse (petit cercle sur le bord du losange), il faut réaliser l'action « Mesurer la température ». Lorsque la condition devient vraie (la température est inférieure ou égale à 17 °C), il faut alimenter la pompe.		**Tant que Température n'est pas <=17** **Faire « Mesurer température »** **Fin tant que**

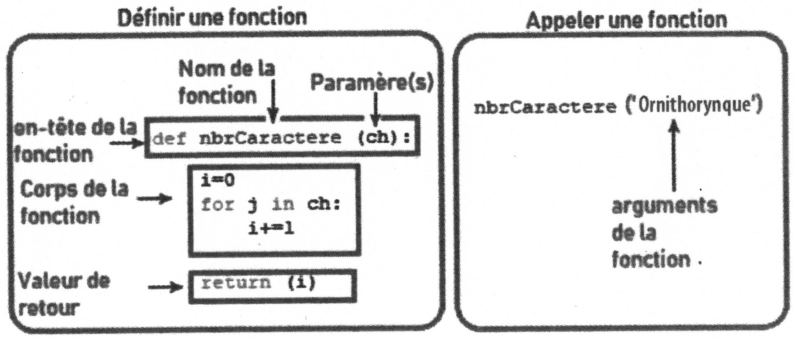
Un morceau de code peut être répété un nombre de fois choisi (*exemple* : pour faire clignoter 3 fois une diode, il faudra l'allumer puis l'éteindre 3 fois). Cela nécessite de mettre en place un compteur. La valeur de ce compteur est incrémentée à chaque fois que le programme exécute la boucle et est comparée à la valeur voulue.
Un morceau de code peut être répété jusqu'à ce qu'une condition soit vraie.

13 Les fonctions

Les fonctions représentent une sorte de « programme dans le programme ». On utilise des fonctions pour regrouper des instructions et les appeler sur demande. Cela permet de simplifier le programme principal.

L'utilisation de fonctions se compose de deux phases :
– la **définition** de la fonction, c'est-à-dire sa création. On choisit un nom pour la fonction et on indique la liste des paramètres éventuels de la fonction. On écrit ce que fait la fonction et on précise le type de valeur retournée en sortie par la fonction ;
– l'**appel** de la fonction en précisant les arguments de la fonction.

Exemple ➤ On peut créer une fonction qui compte le nombre de caractères dans un mot. En Python, cela s'écrira de la manière suivante :

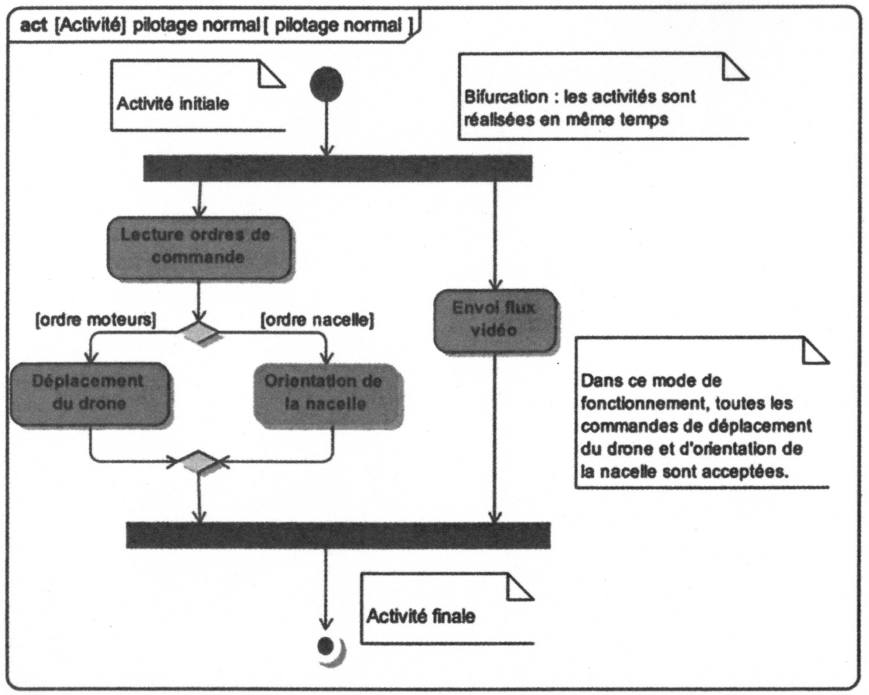


• Paramètres et arguments d'une fonction

Les paramètres sont des variables utilisées dans la définition de la fonction, tandis que les arguments sont les valeurs que nous fournissons à la fonction lors de son appel.

14 Le diagramme d'activités

En SysML, on utilise le diagramme d'activités (*Activity Diagram*) pour faire des algorigrammes.
Le diagramme d'activités permet de décrire la transformation des flux d'entrées en flux de sorties (matières, énergies, informations) par le biais de séquences d'actions.
Lorsqu'une tâche est terminée, la suivante commence : il n'y a pas d'événement associé aux transitions (contrairement au diagramme d'états).



15 Le diagramme d'états-transitions

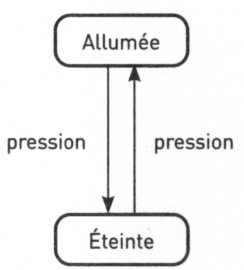
15.1. L'objectif du diagramme

Le diagramme états-transitions modélise l'évolution de l'état d'une machine en fonction des événements qui peuvent se produire. Dans le langage SysML, il est appelé « *State Machine Diagram* » (stm).

Un **diagramme d'états-transitions** est un graphe qui représente une machine dont le comportement des sorties ne dépend pas seulement de l'état de ses entrées, mais aussi d'un historique des sollicitations passées.

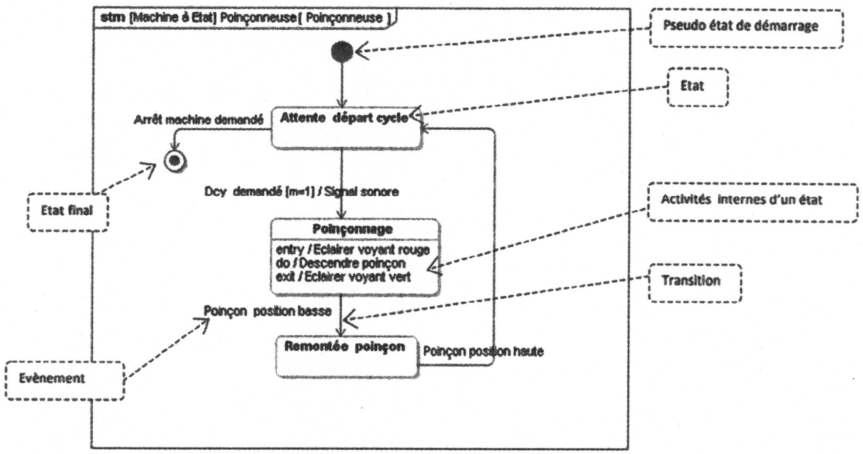
Une machine à états est représentée par un graphe comportant des états, matérialisés par des rectangles aux coins arrondis, et des transitions, matérialisées par des arcs orientés liant les états entre eux.

Exemple ➤ Une ampoule électrique possède deux états : elle peut être allumée ou éteinte.
Le passage d'un état à l'autre se fait à partir d'une même action : l'appui sur un bouton poussoir.
Lorsque l'on appuie sur un bouton d'éclairage, la réaction de l'éclairage associé dépend de son état courant (de son historique) : si la lumière est allumée, elle s'éteindra, si elle est éteinte, elle s'allumera.



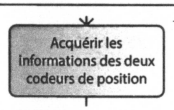
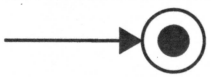
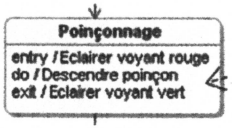
15.2. Les éléments constitutifs du diagramme

Sur un diagramme états-transitions, on trouve les éléments suivants.



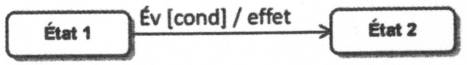
• Les états

L'état d'un objet est une situation stable dans la vie de l'objet. Il peut être en train d'effectuer une activité ou d'attendre un événement.

	**État** Un état représente une période dans la vie d'un objet pendant laquelle ce dernier attend un événement ou accomplit une activité
	**État initial** Il est activé au démarrage de la machine.
	**État final** Il peut y en avoir plusieurs car différents scénarii peuvent être possibles pour mettre fin à un comportement.
	**Un état englobant (ou super-état)** On l'identifie grâce au symbole en bas à droite du rectangle. Il peut être décomposé en sous-états. Cela permet d'occulter certains détails pour une meilleure lisibilité du diagramme.
	On peut, dans l'état, décrire l'enchaînement de différentes activités : – activités exécutées en entrant dans l'état (repérées par le terme « *entry* ») ; – celles exécutées tant que l'on reste dans l'état (repérées par le terme « *do* ») ; – celles exécutées en sortant de l'état (repérées par le terme « *exit* »).

• Les transitions

Les transitions permettent à une machine de passer d'un état à l'autre en fonction d'événements et sous certaines conditions :



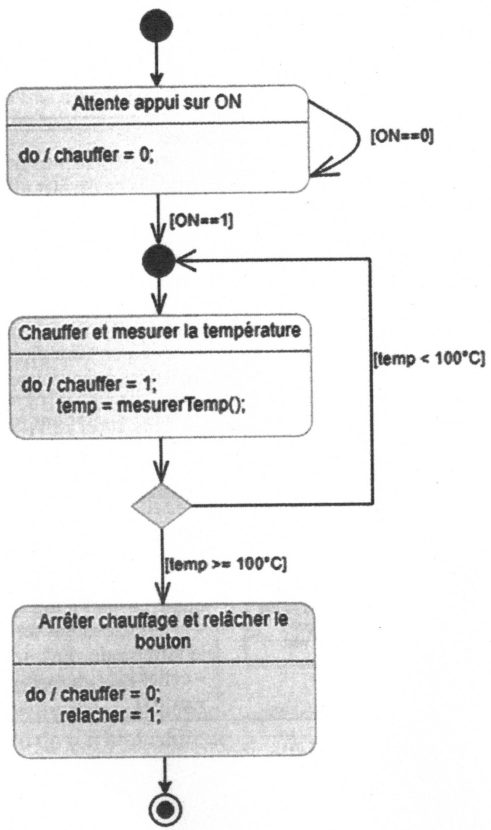
Une transition possède, au maximum :
– un événement déclencheur ;
– une condition appelée « condition de garde » ou « garde ». C'est une condition booléenne. *Exemple* : vérification de l'état de variables associées à des capteurs (« m = 1 » pour vérifier que la variable « m » est à « 1 ») ;
– un effet associé.

- **Les autres éléments du diagramme**

Fin du cycle de traitement	Commentaire
	Test (ou pseudo-état de choix)

15.3. Un exemple d'utilisation d'un test

Le diagramme suivant décrit une bouilloire.
L'utilisateur enfonce le bouton « ON », la bouilloire se met à chauffer l'eau et à tester la température de celle-ci.
Dès que la température atteint 100 °C, l'action de chauffer l'eau s'arrête et le bouton est relâché pour qu'il revienne sur la position OFF.



MES NOTES POUR RÉVISER