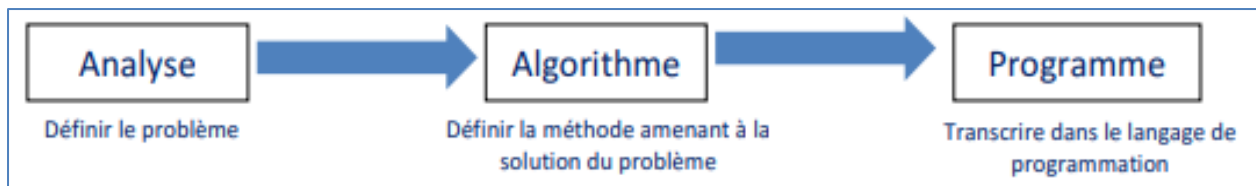


1. Algorithmes et programmes

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 écrites dans un langage compréhensible par une machine s'appelle un **programme informatique**.



1.1. 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.

Pseudo-code	Code
Début Allumer la DEL branchée sur la broche 2 Attendre 1 seconde Éteindre la DEL Attendre 1 seconde Recommencer Fin	<pre> void setup() { pinMode(2, OUTPUT); } void loop() { digitalWrite(2, HIGH); delay(1000); digitalWrite(2, LOW); delay(1000); } </pre>

En pseudo-code, l'instruction d'affectation se note avec le signe \leftarrow . Ainsi :

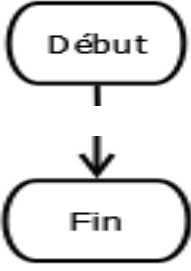
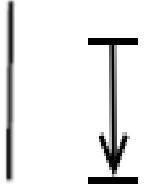
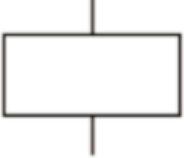
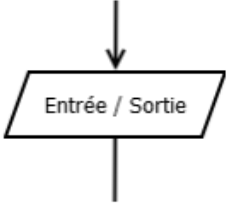

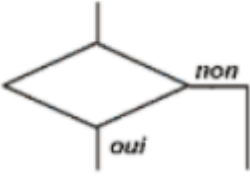
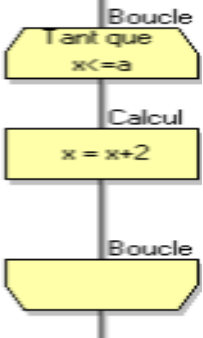

Var_1 \leftarrow 24

signifie : « Attribuer la valeur 24 à la variable appelée Var_1 ».

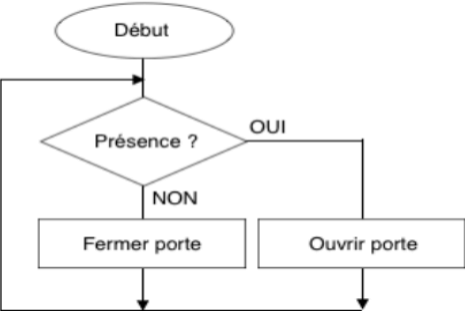
1.2. Algorigramme (ou logigramme)

Un algorithme peut être représenté sous forme graphique en utilisant un **algorigramme**, aussi appelé **logigramme**.

Pour construire un algorigramme, on utilise des symboles normalisés :

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

Exemple :

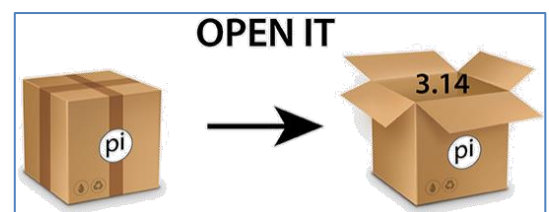
Pseudo-code	Algorithme
<p>Début Si Présence Alors ouvrir porte Sinon fermer porte Retour au début</p>	

2. 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 : $\pi = 3.1415926535$



3. Variables

Une variable est le nom donné à un espace de stockage pour contenir 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.

3.1. Types de variables

Le type des variables détermine les types de manipulations que l'on peut faire.

Exemples :

- 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 (« <i>integers</i> », en Anglais). Exemples : -12 ; 0 ; 7.
Les entiers naturels	unsignedint	Les entiers naturels (« <i>unsigned integers</i> », en Anglais) sont positifs ou égal à zéro. Exemples : 3 ; 0 ; 42.
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 « <i>True</i> » et « <i>False</i> » ou « <i>High</i> » et « <i>Low</i> ».
Les nombres décimaux	float	Les nombres décimaux (nombres à virgule). Exemples : -5,3 ; 0,007 ; 83,739. 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. Elles sont utilisées pour représenter du texte (des mots, des phrases, etc.) ;
Les tableaux	array	Les tableaux (« <i>array</i> », 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. <u>Exemple :</u> 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 (« <i>object</i> », 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).

3.2. 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 qu'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 (= utilisables) à travers l'ensemble du programme.

4. Les opérateurs

Les opérateurs permettent de manipuler ou comparer des valeurs, notamment des variables. Parmi les opérateurs utilisés fréquemment, on identifie :

- les opérateurs **mathématiques** : addition, soustraction, multiplication, division, etc. ;
- les opérateurs de **comparaison** (égal à, différent de, supérieur à, inférieur à) ;
- les opérateurs **logiques** : **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

- = (égalité)
- + (addition)
- - (soustraction)
- * (multiplication)
- / (division)
- % (modulo)

OPÉRATEURS DE COMPARAISON

- == (égal à)
- != (différent de)
- < (inférieur à)
- > (supérieur à)
- <= (inférieur ou égal à)
- >= (supérieur ou égal à)

OPÉRATEURS BOOLÉENS

- && (ET booléen)
- || (OU booléen)
- ! (NON booléen)

OPÉRATEURS COMPOSÉS

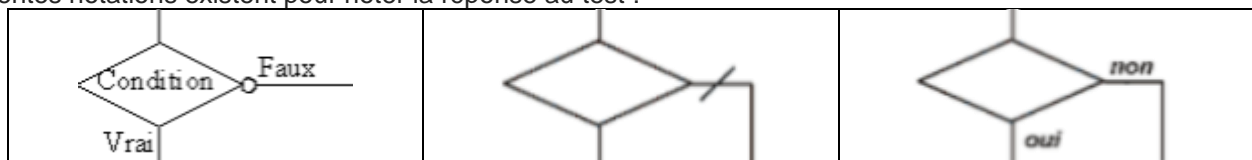
- ++ (incrémentatation)
- -- (décrémentatation)

5. 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.

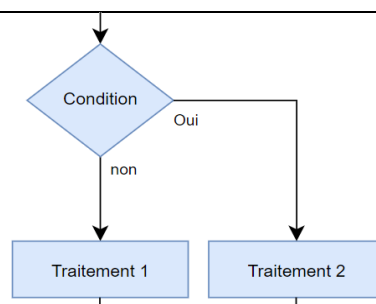
Pseudo-code	Algorithme
Début Si touche « espace » pressée Alors allumer Sinon éteindre Fin Si Retour début	

Différentes notations existent pour noter 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 »
 ...

6. 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.

- 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.

<p>Boucle avec comptage</p> <p>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.</p>	<pre> graph TD Start(()) --> N_x[N = x] N_x --> N_0{N = 0?} N_0 -- non --> Traitement[Traitement] Traitement --> N_minus_1[N = N - 1] N_minus_1 --> N_0 N_0 -- oui --> End(()) </pre>	<p>POUR N = x A 0 REPETER</p> <p>« Traitement »</p> <p>FIN POUR</p>
<p>Boucle conditionnelle</p> <p>Tant que la condition testée est fausse (petit cercle sur le bord du losange), il faut réaliser l'action « Mesurer la température ».</p> <p>Lorsque la condition devient vraie (la température est $\leq 17^\circ\text{C}$), il faut alimenter la pompe.</p> <p>On peut utiliser cette structure pour répéter à l'infini (tant que le système est sous tension) le programme.</p>	<pre> graph TD Start(()) --> Mesurer[Mesurer température] Mesurer --> Temp{Température <= 17°C} Temp --> ALIMENTER[ALIMENTER RC] Temp --> Mesurer </pre>	<p>Tant que Température n'est pas ≤ 17</p> <p>FAIRE « Mesurer température »</p> <p>Fin tant que</p>

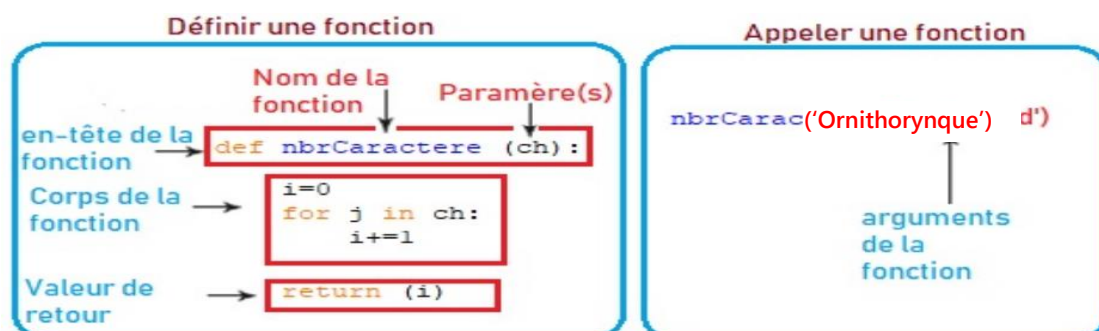
7. Les fonctions

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 précise le type de valeur retournée en sortie, par la fonction.
- L'**appel** de la fonction en précisant les arguments de la fonction

En Python, cela s'écrit de la manière suivante :



7.1. 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.

Lors de l'appel de la fonction, les arguments utilisés doivent être fournis dans le même ordre que celui des paramètres correspondants (en les séparant eux aussi à l'aide de virgules). Le premier argument sera affecté au premier paramètre, le second argument sera affecté au second paramètre, et ainsi de suite.

8. Diagramme d'états-transitions

8.1. Objectif du diagramme

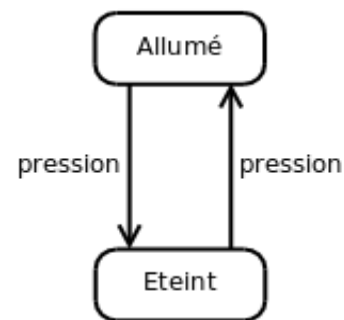
Le diagramme états-transitions modélise l'évolution de l'état d'une machine en fonction des événements qui peuvent se produire. Dans le langage SysML, il est appelé « *State Machine Diagram* » (stm).

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 finis 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.

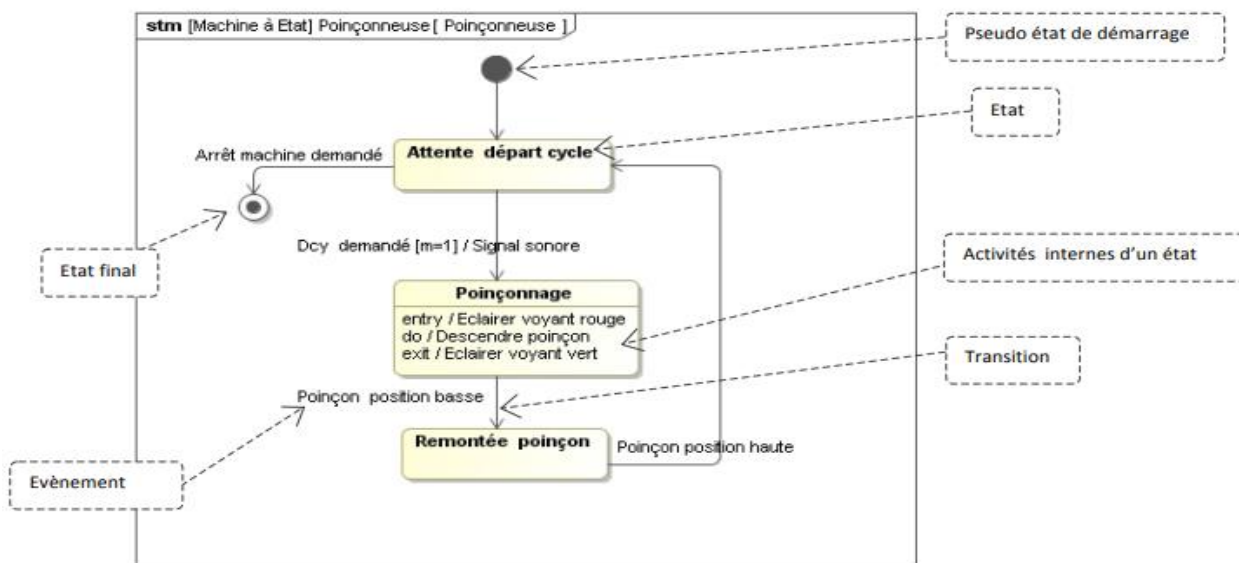
Exemple :

La figure ci-contre montre un exemple simple de diagramme états-transitions. 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.



8.2. Éléments constitutifs du diagramme

Sur un diagramme états-transitions, on trouve :

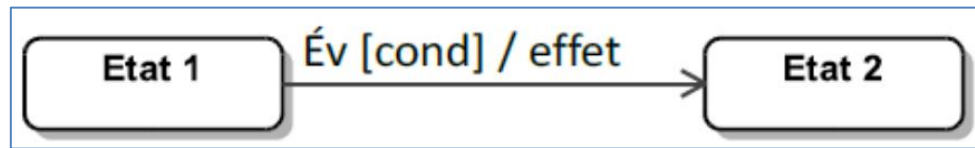


Les états

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 évènement.

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.

Autres éléments du diagramme

	Commentaire
	Test (ou pseudo-état de choix)

Exemple de l'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.

