Fiche

I. Analyser les données et en déduire des modifications à apporter à un programme

De nombreux OST fonctionnent grâce à un **programme de commande** (station météo, robot aspirateur, portail automatique). Ce programme traite des données issues de capteurs et agit pour exécuter des fonctions.
Pour faire évoluer la fonctionnalité d'un OST (par exemple, rendre un feu tricolore plus réactif ou un robot plus précis), il faut d'abord analyser les données manipulées par le programme. Ces données peuvent être :

- des **valeurs mesurées** (température, distance, luminosité) ;
- des **états logiques** (vrai/faux, ouvert/fermé) ;
- des **listes de données** stockant plusieurs valeurs successives (par exemple, l'évolution d'une température au cours du temps).

On doit apprendre à **identifier** les **données obsolètes** (non utilisées ou inutiles) et à **déterminer** les nouvelles données nécessaires pour répondre à la nouvelle fonctionnalité.
Si l'on veut que la lampe d'un salon connecté s'allume seulement quand il fait sombre et qu'une présence est détectée, le programme doit inclure deux capteurs : un de luminosité et un de mouvement. On utilisera alors un opérateur logique ET, car les deux conditions doivent être vraies pour que la lampe s'allume.
Les **opérateurs logiques** sont essentiels dans le raisonnement informatique :

- **ET** : les deux conditions doivent être vraies ;
- **OU** : au moins une des conditions doit être vraie ;
- **NON** : inverse la valeur logique (vrai devient faux, et inversement).

Les programmes peuvent également manipuler des listes, c'est-à-dire des ensembles de données. Par exemple, une liste peut contenir toutes les températures mesurées pendant la journée. Le programme peut ensuite parcourir cette liste pour calculer la température moyenne ou détecter un dépassement de seuil.
Ainsi, l'analyse des données permet de comprendre comment l'OST réagit et de déterminer les modifications à effectuer sur le programme pour adapter son comportement à une nouvelle situation.

II. Compléter un programme pour répondre à une nouvelle fonctionnalité

Une fois les besoins identifiés, il faut modifier ou compléter le programme pour obtenir la fonctionnalité souhaitée. Cela peut aussi nécessiter une évolution de la structure interne de l'objet : ajout, suppression ou remplacement d'un composant.
Par exemple, dans le cas d'un ventilateur automatique, on souhaiterait qu'il s'allume **uniquement** si la température dépasse 25 °C et si la pièce est occupée. Pour cela, on ajoute un capteur de présence et on modifie le programme pour inclure la condition logique : SI (Température > 25) ET (Présence = Vrai) ALORS Ventilateur = ON.
Il faut alors compléter un programme fourni en ajoutant des instructions ou des blocs de code adaptés. Ils doivent comprendre que chaque ligne du programme correspond à une action précise : lire une valeur, comparer, décider, agir.
Modifier un programme peut aussi impliquer des ajustements matériels :

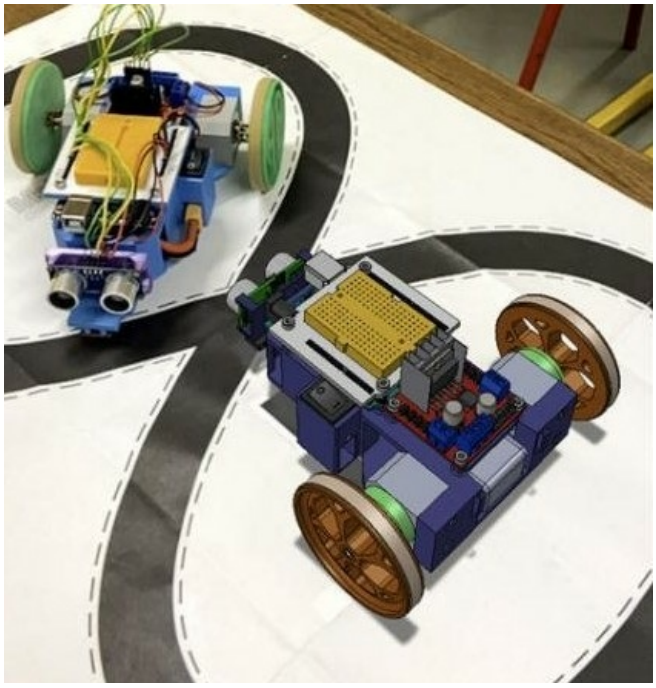
- **ajouter** un capteur ou un actionneur supplémentaire ;
- **supprimer** un composant inutile ;
- **vérifier** que les connexions électriques ou logiques sont cohérentes.

Ces étapes permettent de faire évoluer un objet existant pour répondre à un besoin nouveau sans avoir à le remplacer entièrement. Cela participe à une approche durable et responsable de la technologie.

III. Tester et valider, dans un environnement simulé ou réel, une modification du programme

Après avoir modifié le programme, il est essentiel de **tester** et **valider** le nouveau fonctionnement. Cette phase permet de vérifier que le système répond bien aux consignes et qu'aucune erreur n'a été introduite. Les tests peuvent se faire dans un **environnement simulé** (logiciel de programmation type mBlock, Scratch, Tinkercad, Arduino IDE en mode simulation) ou sur **l'objet réel** (robot, carte microcontrôleur, maquette d'OST).
L'objectif est de vérifier la réaction du système face à des événements précis. Un événement est une action ou une situation qui déclenche une séquence d'instructions : par exemple, appuyer sur un bouton, recevoir un signal d'un capteur, dépasser une température seuil, détecter un mouvement.
Lors du test, on **observe** le comportement du système et on le **compare** au résultat attendu. Si le fonctionnement est conforme, la modification est validée. Sinon, il faut corriger le programme en ajustant les conditions, les délais ou la gestion des variables.

Par exemple, dans un robot suiveur de ligne, si le robot s'écarte de sa trajectoire, le test permettra de détecter un défaut dans la lecture des capteurs de lumière ou dans la vitesse des moteurs. Le programme sera ajusté en conséquence.



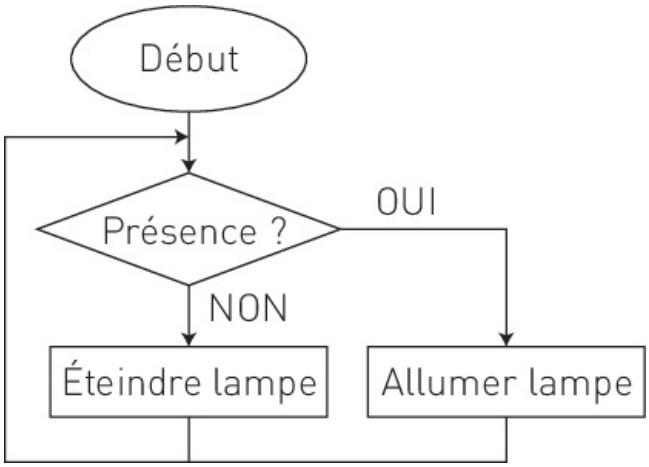
Tester et valider, c'est donc vérifier la cohérence entre la théorie et la pratique. Cette démarche expérimentale est essentielle, car elle forme à la rigueur, à l'observation et à la logique du raisonnement scientifique.

IV. Logigramme et programmation d'un objet embarqué

Pour programmer un automate, un système embarqué, on commence par faire une description textuelle du comportement voulu grâce à des mots comme « si », « alors », « sinon », « tant que »..., puis une description graphique (logigramme). On réalise l'algorithme du comportement de notre système embarqué (automate).

• Algorithmes de base

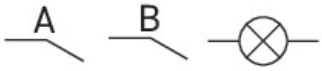
Nous avons déjà présenté les fonctions de base d'un algorithme qui sont « **l'action** » et la « **question** », l'événement prenant 2 valeurs possibles (oui-non, ouvert-fermé).

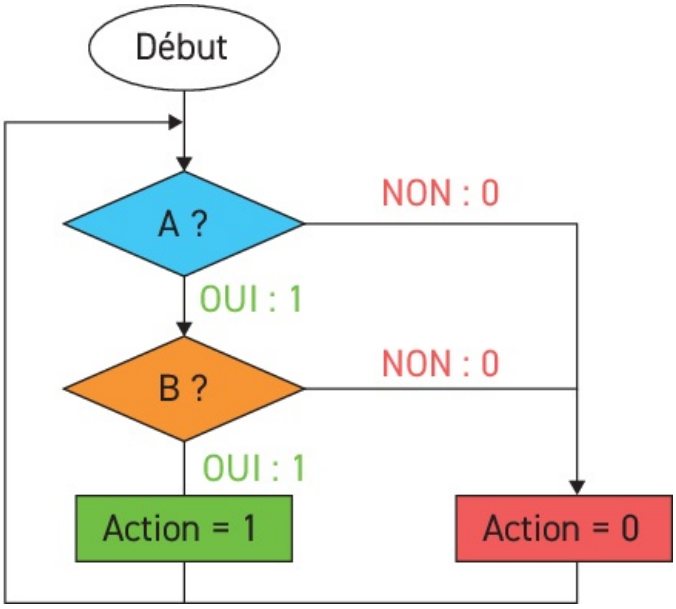


• Fonctions ET/OU

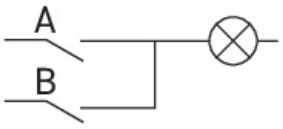
Les fonctions ET/OU permettent de relier plusieurs « questions ».

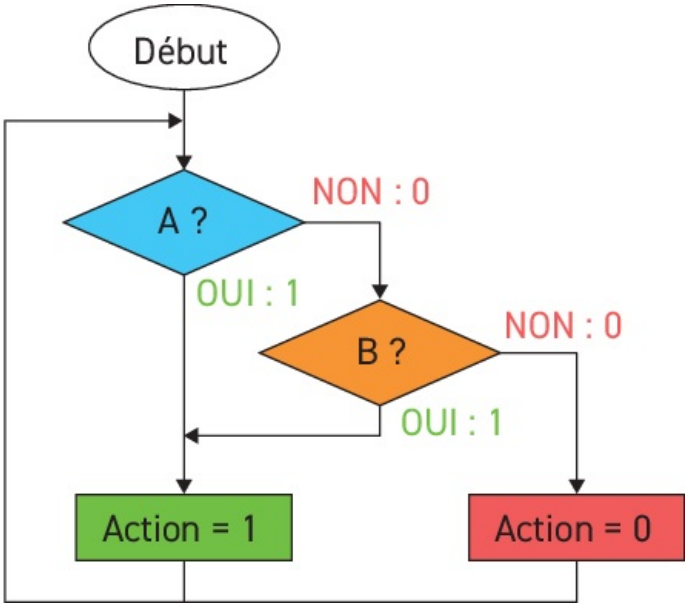
Fonction ET		
A ?	B ?	Sortie
0	0	0
0	1	0
1	0	0
1	1	1





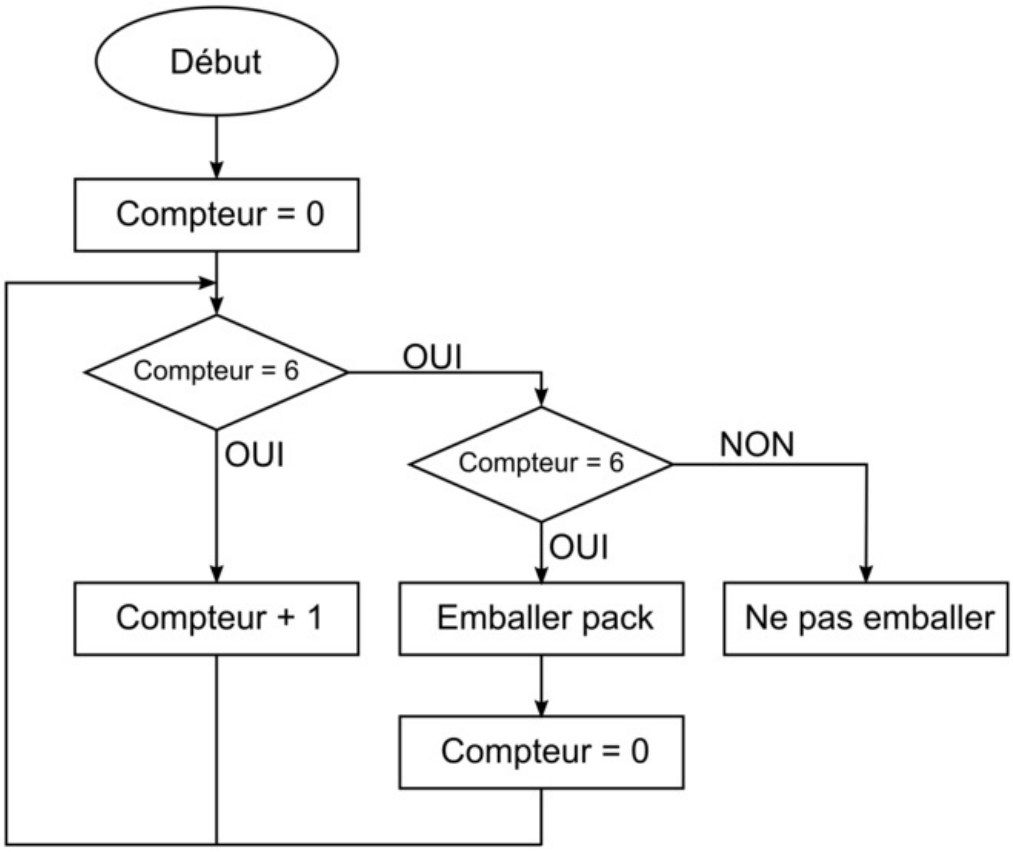
Fonction OU		
A ?	B ?	Sortie
0	0	0
0	1	1
1	0	1
1	1	1





• Algorithme par variables

Pour les besoins de certains algorithmes, on va devoir utiliser des variables (sortes de boîtes).
Il faut d'abord les **déclarer** (nom et contenu), et les **initialiser**, c'est-à-dire leur donner une valeur de départ (souvent 0).
Puis définir leur **évolution** (incrément de 1 à chaque passage).
Par exemple, pour emballer des packs de bouteilles d'eau, le capteur va détecter le passage de chaque bouteille, puis emballer le pack dès qu'il y en a 6. La variable est le nombre de bouteilles d'eau, elle est remise à 0 après avoir atteint 6.



• **Utilisations de sous-programmes**

L'utilisation de **sous-programmes** permet une meilleure lisibilité lors d'une succession d'actions identiques.
À la place d'une « action », on va mettre un « appel sous-programme ».

À retenir :

1. Un **programme** commande les **actions** d'un OST en traitant des données issues de capteurs. Modifier un programme, c'est **ajuster** les données et les conditions logiques pour changer le comportement de l'objet.

2. **Compléter** un programme peut nécessiter de **modifier** la **structure** matérielle de l'OST.

3. Le **test** et la **validation** garantissent que la modification fonctionne réellement, que ce soit en simulation ou sur l'objet réel.

Définitions importantes :

• **Opérateur logique** : symbole utilisé pour combiner des conditions (ET, OU, NON).

• **Liste** : structure permettant de stocker plusieurs valeurs dans un programme.

• **Capteur** : élément qui mesure une grandeur physique et envoie une donnée.

• **Actionneur** : élément qui réalise une action (moteur, lampe, alarme).

• **Programme** : suite d'instructions qui permet de commander un système technique.

• **Événement** : action ou situation déclenchant une séquence d'instructions dans un programme.

• **Simulation** : reproduction virtuelle du fonctionnement d'un système réel pour le tester sans risque.