



## 1. Blocs d'organisation (OB)

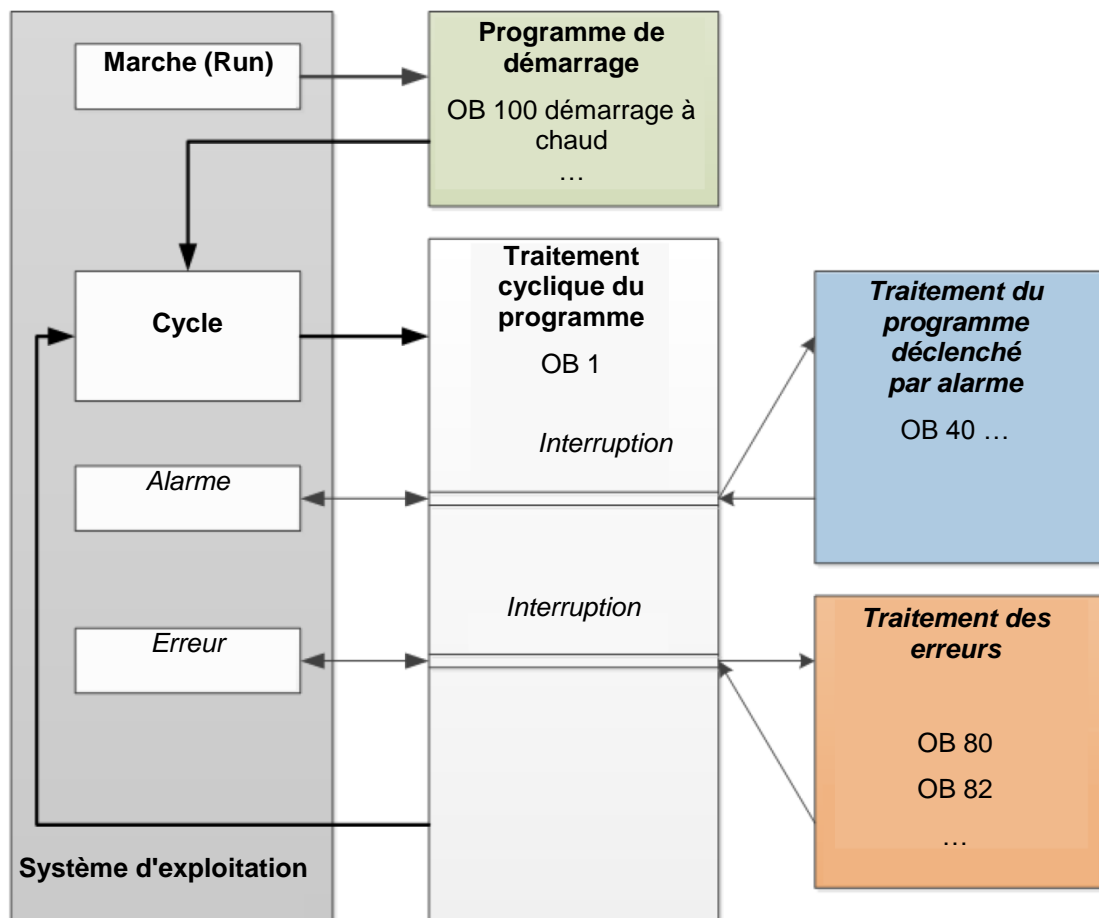


Figure 1 : Événements déclencheurs dans le système d'exploitation et appels de OB

Les blocs d'organisation (OB) constituent l'interface entre le système d'exploitation de l'automate (CPU) et le programme utilisateur. Ils sont appelés par le système d'exploitation et gère les opérations suivantes :

- Traitement cyclique du programme (OB1);
- Comportement au démarrage de l'automate (OB100);
- Traitement du programme déclenché par alarme;
- Traitement des erreurs.

Un projet doit contenir au moins un bloc d'organisation pour le traitement cyclique du programme. Un OB est appelé par un événement déclencheur comme représenté dans la Figure 1. Des priorités sont définies pour les différents OB afin que l'OB1 cyclique puisse par exemple être interrompu par un OB82 pour le traitement des erreurs.

Les réactions suivantes sont possibles après qu'un événement déclencheur s'est produit :

- Si vous avez affecté un OB à l'événement, il déclenchera l'exécution de l'OB affecté. Si la priorité de l'OB affecté est plus élevée que celle de l'OB en cours d'exécution, celui-ci est immédiatement exécuté (interruption). Si ce n'est pas le cas, le système attend d'abord jusqu'à ce que l'exécution de l'OB avec la priorité plus élevée soit terminée;
- Si l'événement n'est affecté à aucun OB, la réaction système par défaut est exécutée.

Le tableau 1 ci-dessous montre différents exemples d'événements déclencheurs pour un SIMATIC S7-1200. Il contient aussi des numéros d'OB possibles et les réactions système prédéfinies qui sont exécutées lorsque le bloc d'organisation (OB) correspondant n'est pas présent dans l'automate.



Évènement déclencheur	Numéros d'OB possibles	Réaction système prédéfinie
Mise en route	100, $\geq 123$	Ignorer
Programme cyclique	1, $\geq 123$	Ignorer
Alarme horaire	10 à 11	-
Alarme de mise à jour	56	Ignorer
Temps de cycle imparti dépassé une fois	80	Ignorer
Temps de cycle imparti dépassé deux fois	80	STOP
Alarme de diagnostic	82	Ignorer

Tableau 1 : Numéros d'OB pour différents évènements déclencheurs

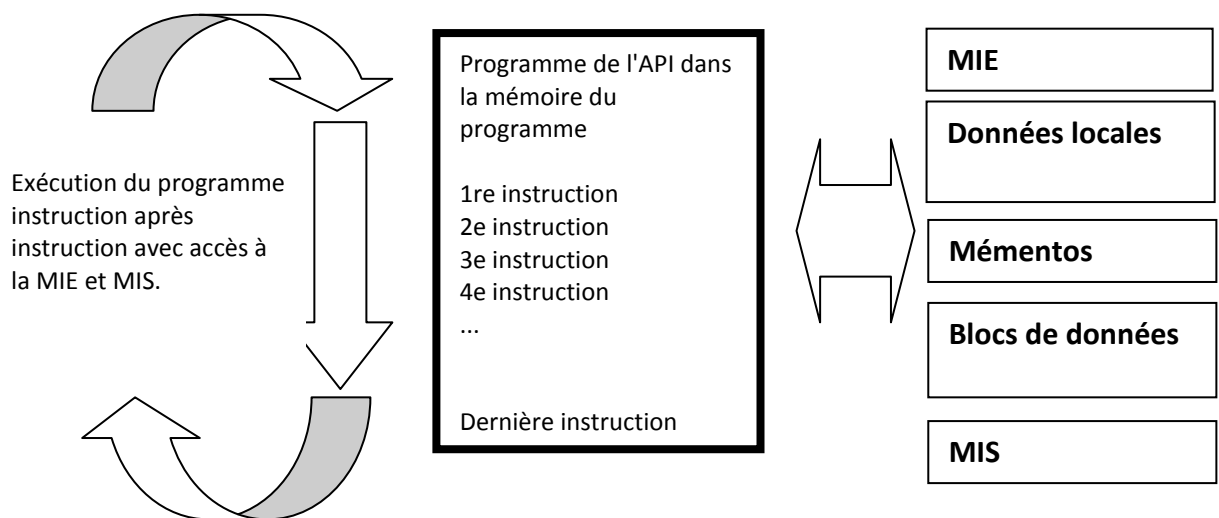
## 2. Mémoire image et traitement cyclique du programme

Si les entrées (I) et sorties (Q) sont adressées dans le programme utilisateur cyclique, les états des signaux ne sont pas interrogés directement par les modules d'entrées/sorties mais il est accédé à la zone de mémoire de la CPU. Cette zone de mémoire contient une image des états des signaux et est appelée mémoire image.

Le traitement cyclique du programme se déroule comme suit :

1. Au début du programme cyclique, le système vérifie si chacune des entrées est sous tension ou non. L'état de ces entrées est enregistré dans la **mémoire image des entrées (MIE)**. Si l'entrée est sous tension, l'information 1 ou "High" sera enregistrée. Si l'entrée n'est pas sous tension, l'information 0 ou "Low" sera enregistrée.
2. Le processeur exécute alors le programme stocké dans le bloc d'organisation cyclique. L'information d'entrée requise à cet effet est prélevée dans la **mémoire image des entrées (MIE)** lue auparavant et les résultats logiques sont écrits dans une **mémoire image des sorties (MIS)**.
3. A la fin du cycle, la **mémoire image des sorties (MIS)** est transférée comme état logique aux modules de sorties et celles-ci sont activées ou désactivées. La procédure reprend ensuite à partir du point 1.

Enregistrer l'état des entrées dans la MIE.



Transmettre l'état de la MIS aux sorties.

Figure 2 : Traitement cyclique du programme

Remarque : le temps requis par le processeur pour l'exécution du programme s'appelle le temps de cycle. Ce dernier dépend entre autres du nombre et du type d'instructions ainsi que de la puissance du processeur de l'automate.



### 3. Les fonctions (FC)

Les fonctions (FC) sont des blocs de code sans mémoire. Elles **n'ont pas de mémoire de données** dans laquelle il est possible d'enregistrer les valeurs de paramètres de bloc. C'est pourquoi tous les paramètres d'interface doivent être interconnectés lors de l'appel d'une fonction. Des blocs de données globaux doivent être créés pour stocker durablement les données.

Une fonction contient un programme qui est toujours exécuté quand un autre bloc de code appelle cette fonction.

Les fonctions peuvent par exemple servir dans les cas suivants :

- Retourner un résultat dépendant des valeurs d'entrée pour les fonctions mathématiques.
- Exécuter des fonctions technologiques comme des commandes uniques avec combinaisons binaires.

Une fonction peut également être appelée plusieurs fois à divers endroits du programme.

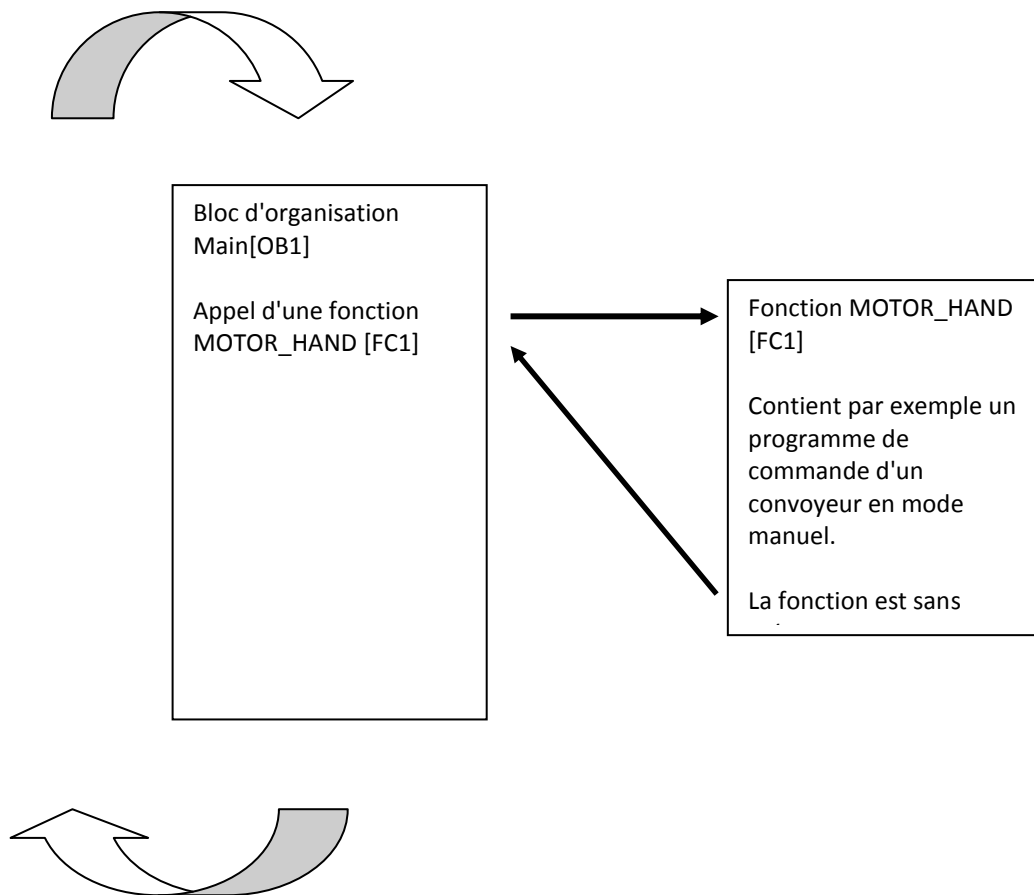


Figure 3 : Fonction avec appel d'un bloc d'organisation Main [OB1]



#### 4. Blocs fonctionnels (FB) et blocs de données d'instance

Les blocs fonctionnels sont des blocs de code qui mémorisent durablement leurs variables d'entrée, de sortie et d'entrée/sortie ainsi que leurs variables statiques dans des blocs de données d'instance afin qu'il soit possible d'y accéder même **après le traitement de blocs**. Pour cette raison, ils sont aussi appelés blocs avec mémoire.

Les blocs fonctionnels peuvent aussi travailler avec des variables temporaires. Cependant, les variables temporaires ne sont pas enregistrées dans le DB d'instance mais disponibles uniquement tout le temps d'un cycle.

Les FB sont utilisés pour des tâches qui ne peuvent être mises en œuvre avec des fonctions :

- Toujours quand les temporisations et les compteurs sont nécessaires dans un bloc;
- Toujours quand une information doit être enregistrée dans le programme.  
Par ex. un indicatif de mode de fonctionnement avec un bouton.

Les FB sont toujours exécutés quand un bloc fonctionnel est appelé par un autre bloc de code. Un FB peut aussi être appelé plusieurs fois à divers endroits du programme. Ceci facilite la programmation de fonctions complexes et répétitives.

Un appel d'un bloc fonctionnel est désigné par le terme "instance". Pour chaque instance d'un FB, une zone mémoire lui est affectée, contenant les données utiles au traitement du bloc. Cette mémoire est fournie par des blocs de données que le logiciel génère automatiquement.

Il est également possible de fournir de la mémoire pour plusieurs instances dans un bloc de données sous forme de **multi-instance**. La taille maximale des DB d'instance varie selon la CPU. Les variables déclarées dans le bloc fonctionnel déterminent la structure du bloc de données d'instance.

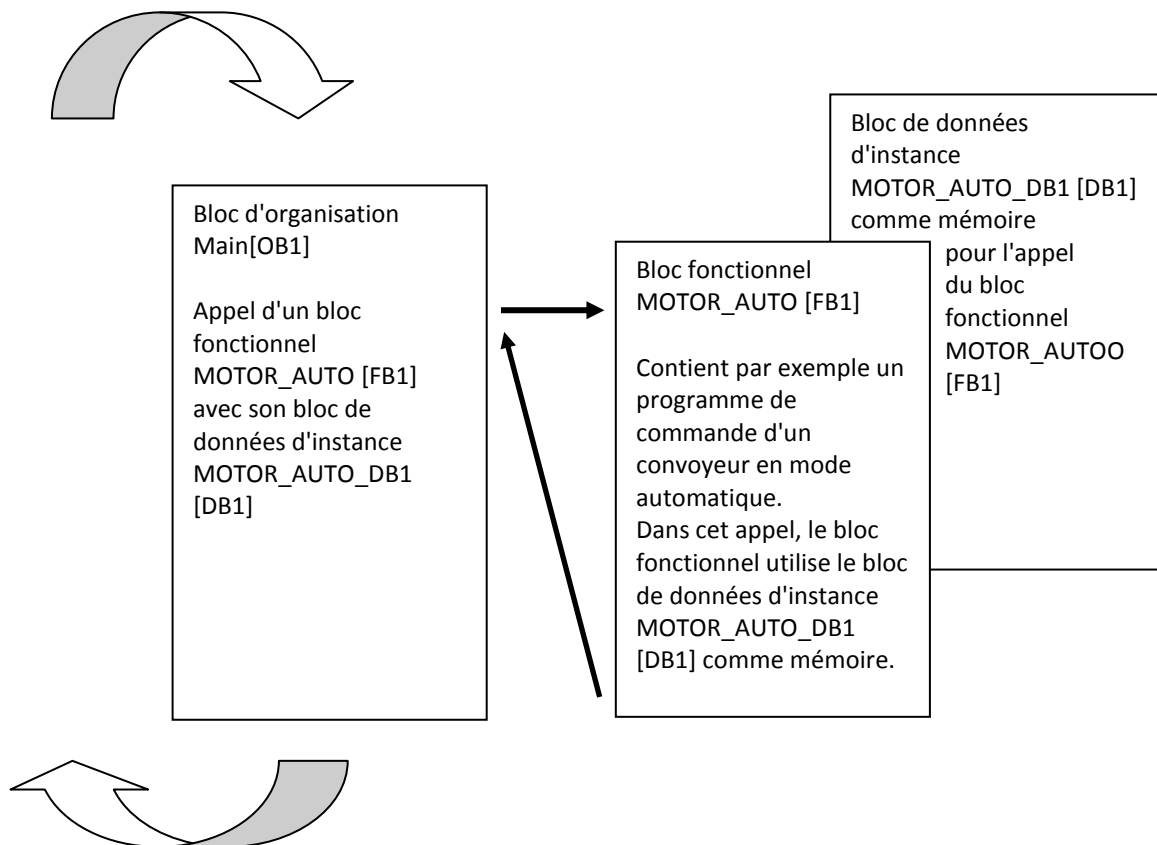


Figure 4 : Bloc fonctionnel et instance avec appel d'un bloc d'organisation Main [OB1]



## 5. Blocs de données globaux (DB)

Contrairement aux blocs de code, les blocs de données ne contiennent pas d'instructions, mais ils sont utilisés pour enregistrer les données utilisateur.

Les blocs de données contiennent donc des données variables qui sont utilisées dans le programme utilisateur. La structure des blocs de données globaux peut être définie au choix.

Les blocs de données globaux stockent des données qui peuvent être utilisés **par tous les autres blocs** (voir figure 5). L'accès aux blocs de données d'instance doit être réservé au bloc fonctionnel correspondant. La taille maximale des blocs de données varie selon la CPU.

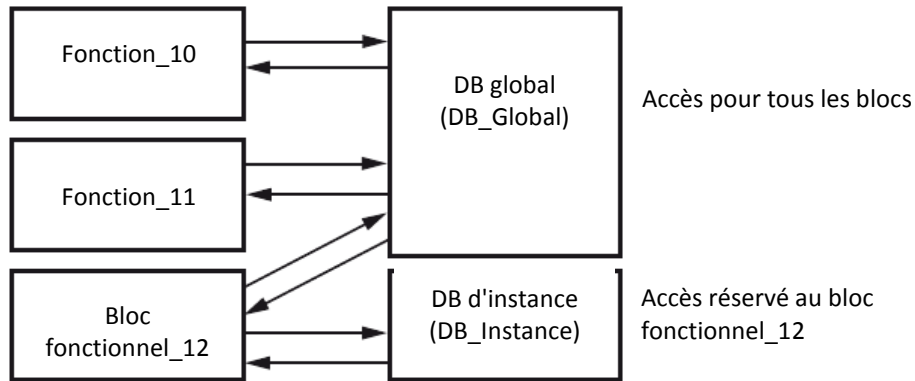


Figure 5 : Différence entre bloc de données global et bloc de données d'instance.

Exemples d'application pour les **blocs de données globaux** :

- Enregistrement des informations pour la gestion d'un magasin. "Où se trouve quel produit ?"
- Enregistrement des recettes de produits donnés.



## 6. Blocs de code compatibles avec la bibliothèque

Un programme utilisateur peut être créé de façon linéaire ou structurée. La **programmation linéaire** consiste à écrire le programme utilisateur complet dans l'OB de cycle. Cela n'est toutefois recommandé que pour des programmes simples pour lesquels on utilise désormais d'autres systèmes de commande plus économique telle que LOGO!

Une **programmation structurée** est recommandée pour des programmes plus complexes. Vous pouvez subdiviser la tâche d'automatisation complexe en plusieurs petites tâches partielles à réaliser par des fonctions et blocs fonctionnels. Il convient de créer des blocs de code compatibles avec la bibliothèque pour cela. Autrement dit, les paramètres d'entrée et les paramètres de sortie d'une fonction ou d'un bloc fonctionnel sont définis de manière générale et les variables globales actuelles (entrées/sorties) ne leur sont attribuées que lors de l'utilisation du bloc.

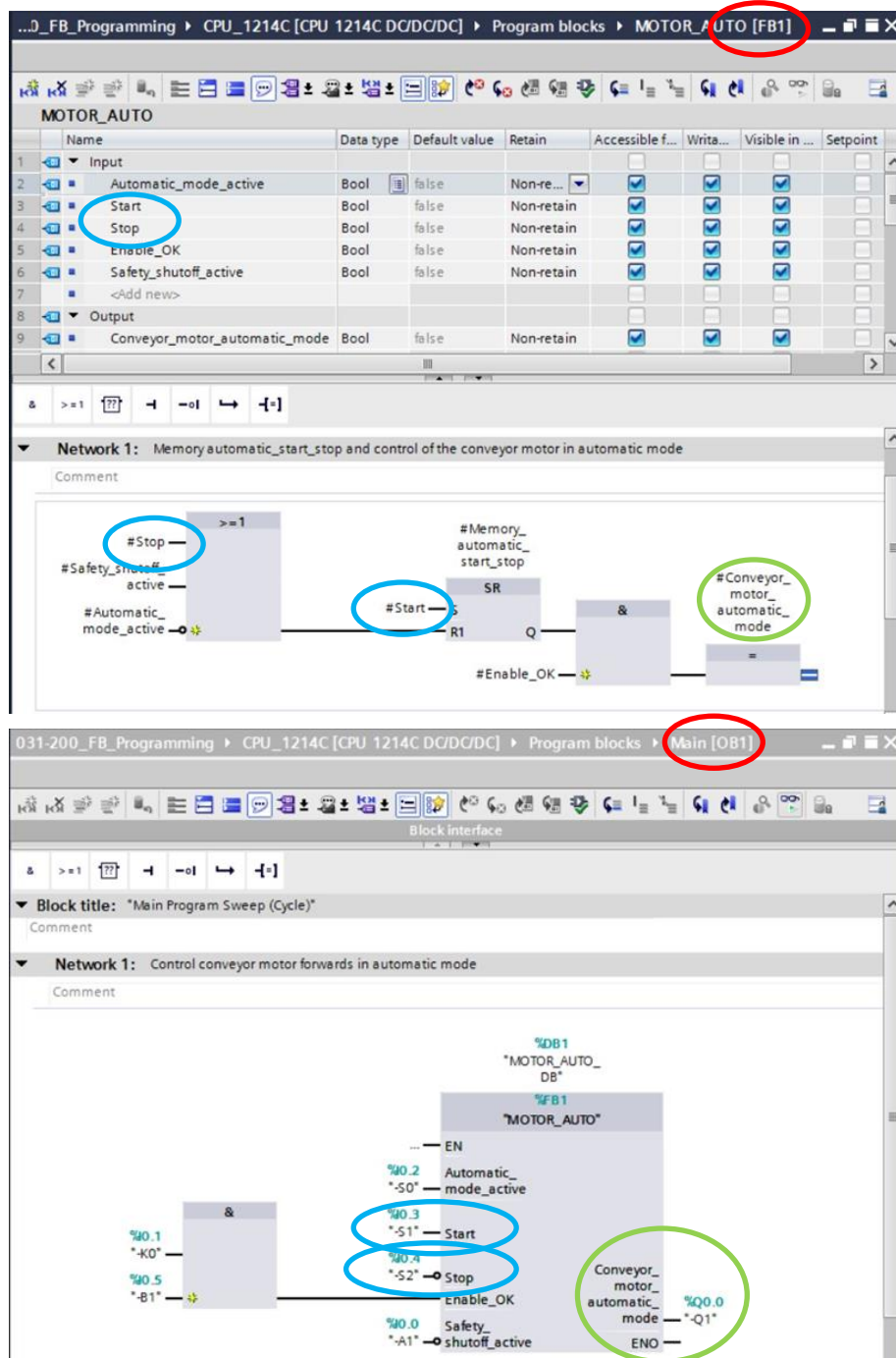


Figure 6 : Bloc fonctionnel compatible avec la bibliothèque avec appel dans OB1



## Questionnaire

Choisissez la bonne réponse

- ☐ Le bloc OB1 permet d'organiser le programme
- ☐ Le bloc OB1 n'est appelé qu'une seule fois lors de l'exécution du programme
- ☐ Le bloc OB1 c'est le bloc principal, il est scruté de manière périodique

C'est quoi un DB d'instance dans TIA Portal ?

- ☐ C'est un bloc de données généré lors de la création d'un bloc FC
- ☐ C'est un bloc fonctionnel généré automatiquement par le programme
- ☐ C'est un bloc de données généré lors de la création d'un FB

Choisissez la bonne réponse

- ☐ Le bloc OB82 est appelé lors d'une erreur de temps de cycle
- ☐ Le bloc OB82 est appelé lors d'une erreur d'accès aux E/S de l'automate
- ☐ Le bloc OB82 est appelé lors d'une erreur d'alimentation

C'est quoi le bloc OB100 ?

- ☐ C'est le bloc qui gère les erreurs matérielles
- ☐ C'est le bloc de démarrage il sert à initialiser les variables de l'automate au premier scan

Le bloc OB100 doit-il être obligatoirement présent dans un programme ?

- ☐ Oui
- ☐ Non

Peut-on appeler un bloc d'organisation OB ?

- ☐ Non
- ☐ Oui

Quel est l'ordre d'exécution cyclique d'un programme?	
A quel moment une fonction peut être appelée?	
A quel moment un bloc fonctionnel peut être appelé?	
Quelle différence entre un bloc de données globales et un bloc de données d'instance?	
Quelles sont les éléments indispensables à une programmation structurée?	
Définir MIE et MIS	