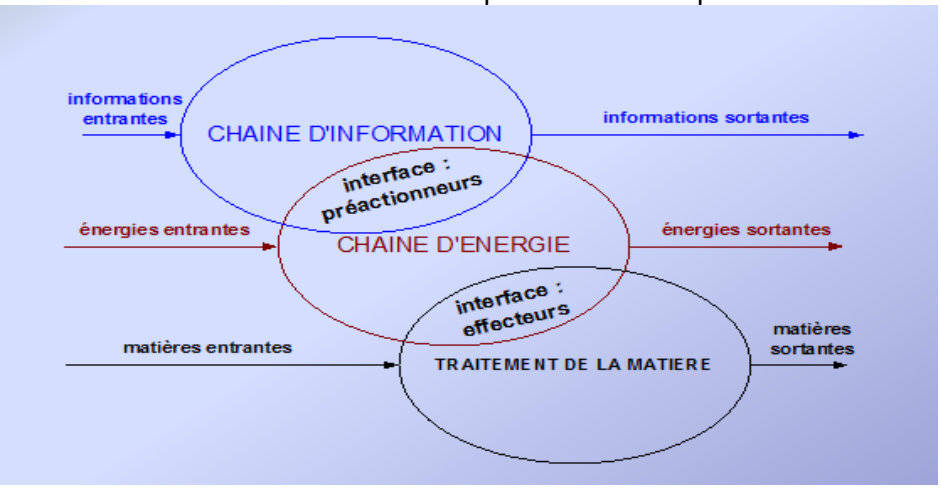
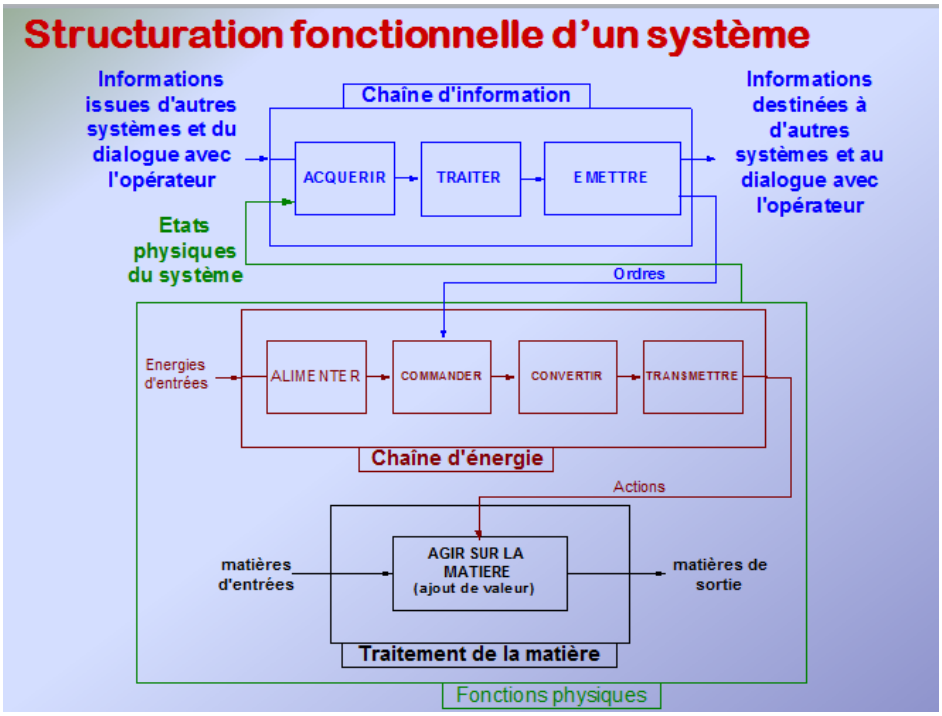
# Traitement de l'information

## I) Introduction :

Le traitement de l'information s'est développé de manière exponentielle depuis un demi-siècle. La domotique (grand public) ou l'automatisme (industriel) atteignent maintenant des niveaux techniques qui permettent d'améliorer le confort de vie ou l'optimisation de la production.




**On remarque que le contrôle d'un système nécessite d'acquérir les états physiques de celui-ci :**

**C'est le rôle des *capteurs (T4)***

**On remarque que le contrôle d'un système nécessite de traiter *(T5)* les informations acquises par les capteurs :**

**C'est le rôle des *microcontrôleurs ou microprocesseur***

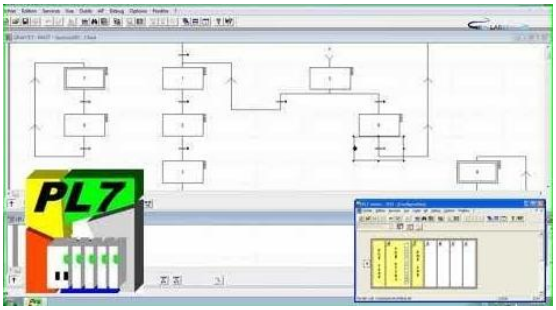
L'utilisation d'un microprocesseur ou d'un microcontrôleur s'intègre dans le traitement des informations de la partie commande d'un système automatisé. Ces informations proviennent essentiellement de la partie opérative (états du système) et des consignes données par l'opérateur, utilisateur du système.

## II) Les langages de programmation :

On peut classer les langages de programmation en trois niveaux.

- **Les langages propres :**

On utilise un langage particulier sur des systèmes dédiés à un fonctionnement précis. Par exemple, le PL7-2 de Télémécanique pour la programmation d'automates programmables TSX de la série 7. Il s'agit d'un langage propre à ce type d'automate.




Pl7-2 télémécanique

Pour la programmation, le point de vue est celui de **l'automaticien.**

- **Les langages assembleurs :**

Chaque microprocesseur possède son propre langage d'assemblage. Il s'agit d'un langage très proche du système, en effet il s'agit d'un langage mnémonique où chaque code opération est remplacé par une expression anglo-saxonne de quelques lettres. Sa transformation en code directement utilisable par le système informatique est très rapide, elle consiste en une simple **compilation** et un **transfert direct** des codes machine. Le langage assembleur nécessite la mise en œuvre d'un environnement de développement réduit et utilise le point de vue du **développeur en informatique industrielle.**

```
movb $0x61,%al
```

**compilation**

```
10110000 01100001
```

- **Les langages évolués :**

Un langage évolué est proche du langage courant (**en langue anglaise !!!).** Un tel langage est indépendant du type de microprocesseur mis en œuvre. L'avantage de son utilisation réside essentiellement en son universalité et sa transférabilité. Il nécessite cependant un environnement de développement conséquent et un point de vue **d'informaticien.**

*Exemple :*

……

……

## III) Exécution d'un programme :

La réalisation d'un programme suit toute une chaîne de développements permettant d'arriver à l'exécutable final (donc en langage binaire). Ces étapes sont au nombre de 4 et se déroulent de la manière suivante :

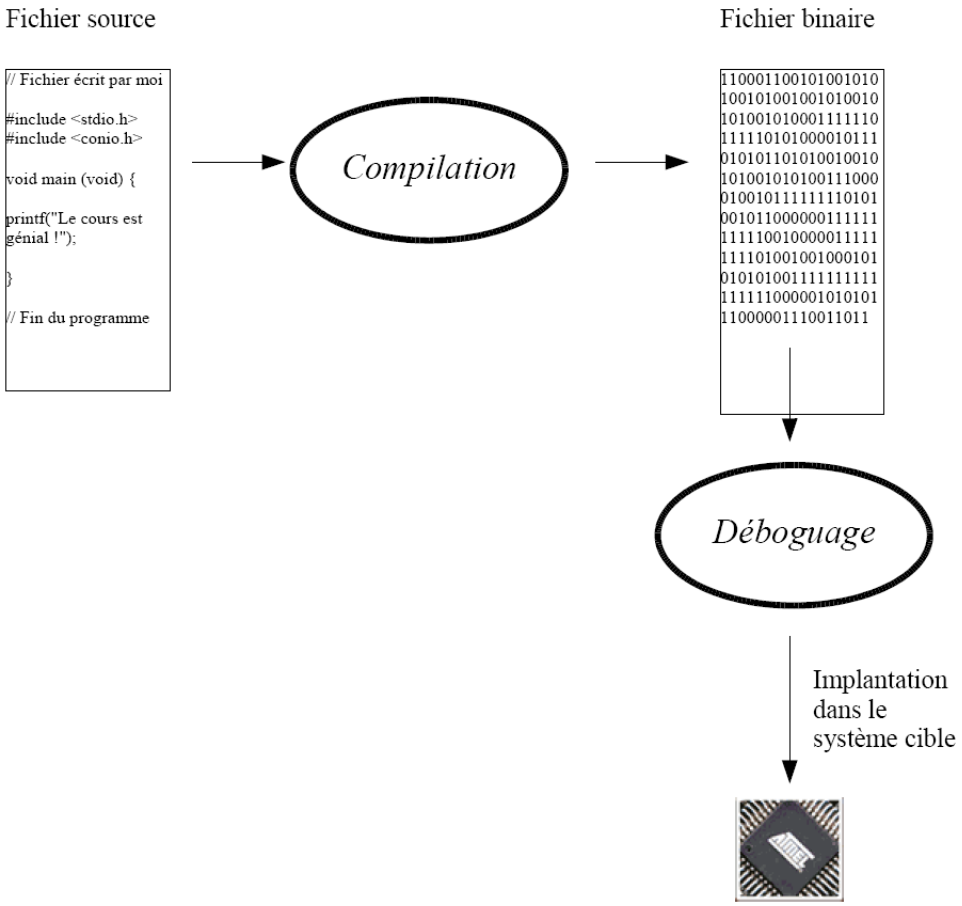
1. L'édition : pendant cette phase, on écrit un fichier source dans un langage déterminé. C'est le plus souvent un fichier texte (C, python), mais cela peut aussi être un fichier graphique (grafcet) comme sous AUTOMGEN.

2. La compilation : Pendant cette phase, le fichier source est *transformé* en fichier binaire par le biais d'un autre programme que l'on appelle ***compilateur***.

3. Le débogage : avant d'implanter le programme il peut être nécessaire de le vérifier et de corriger les éventuelles erreurs (il est rare quand il n'y en ait pas). Pour cela on utilise un autre logiciel appelé ***débogueur*** et qui permet de simuler le programme. Les options que ces logiciels proposent permettent de cibler l'erreur beaucoup plus rapidement que de tâtonner jusqu'à tomber dessus.

4. L'implantation : une fois que tout est terminé, il n'y a plus qu'à implanter le programme dans le système cible, c'est à dire un automate ou bien un microcontrôleur par le biais d'un câble (« **téléversement** » pour Arduino)

Le schéma donné ci-dessous vous résume tout le processus de création et de chargement d'un programme.



## IV) L'ANALYSE ALGORITHMIQUE :

### 1. Introduction à la notion d'algorithme :

Un algorithme est la décomposition en une suite d'étapes d'un programme destiné à résoudre un problème ou à accomplir une tâche déterminée. On écrit habituellement un algorithme en pseudocode, consistant en une combinaison de texte humainement compréhensible (principalement pour la partie descriptive) et des portions de code s'approchant au plus près de la syntaxe d'un langage de programmation afin de faciliter l'écriture du programme final.

Un problème est d'abord caractérisé par un énoncé.

- **La première étape passe par l'identification des données du problème et des résultats à obtenir, c'est-à-dire définir les entrées et les sorties. C'est la *phase d'initialisation* : C'est la préparation du traitement du problème.**

- **La deuxième étape consiste à définir la méthode qui est utilisée pour obtenir les résultats recherchés, elle constitue ce que l'on appelle la définition de l'algorithme. C'est la *phase de traitement du problème :* On détermine les étapes du traitement et donc les instructions à donner pour une exécution automatique.**

- **La troisième étape consiste à coder l'algorithme en un langage (propre, assemblage ou évolué), C'est la *phase de codage*. Les résultats obtenus peuvent être affichés à l'écran, imprimés ou encore sauvegardés dans un fichier. Les sorties peuvent éventuellement être des graphiques, des images...**

**Enfin on peut éventuellement faire des tests doivent valider la conception de la commande.**

Exemple de réalisation d'un algorithme :

```
Algorithme Puissance
// algorithme qui calcule une puissance d'un nombre
Variables
   x,puissance : réels;
   k,i : entier;
Début
   x ← Saisie(); // L'utilisateur doit entrer un réel
   k ← Saisie(); // L'utilisateur doit entrer un entier
   puissance ← x; // initialisation de la variable puissance
   Pour i allant de 1 à k faire // « répéter k fois » n'existe pas…
      puissance ← puissance * puissance;
   fin pour
   Ecrire(puissance);
Fin
```

Entête

Déclarations

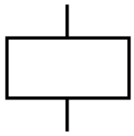
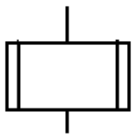
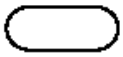
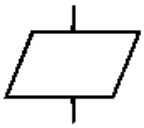
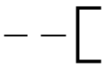
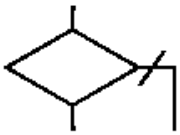
Corps=Description du calcul et des interactions

## 2. Ecriture des structures algorithmiques :

Pour décrire les algorithmes, nous utiliserons 2 types de représentation :

- par **organigramme ou algorigramme**
- par écriture en pseudo code ou **algorithme.**

La représentation par **algorigramme** est une méthode graphique de description des algorithmes.
Un organigramme permet de décrire les différentes opérations sous forme d'un schéma indiquant les différents ordres et conditions traités par l'algorithme. Les symboles utilisés sont les suivants :

| SYMBOLE | DESIGNATION | SYMBOLE | DESIGNATION |
|---|---|---|---|
| Symboles de traitement | | Symboles auxiliaires | |
|  | **Symbole général** Opération ou groupe d'opérations sur des données, instructions, pour laquelle il n'existe aucun symbole normalisé. |  | **Renvoi** Symbole utilisé deux fois pour assurer la continuité lorsqu'une partie de ligne de liaison n'est pas représentée. |
|  | **Sous-programme** Portion de programme considérée comme une simple opération. |  | **Début, fin , interruption** Début, fin ou interruption d'un algorigramme. |
|  | **Entrée-Sortie** Mise à disposition d'une information à traiter ou enregistrement d'une information traitée. |  | **Commentaire** Symbole utilisé pour donner des indications sur les opérations effectuées. |
| Symbole de test | | Les différents symboles sont reliés entre eux par des lignes de liaisons. | |
|  | **Branchement** Exploitation de conditions variables impliquant un choix parmi plusieurs. | | |
| Sens conventionnel des liaisons | | | |
| Le sens général des lignes de liaison doit être :<br>• De haut en bas<br>• De gauche à droite<br>Lorsque le sens général ne peut pas être respecté, des pointes de flèche à cheval sur la ligne indiquent le sens utilisé. | | | |

Un algorithme complet fait appel à des structures de contrôle qui se succèdent. La compréhension de ces structures est fondamentale, en ce sens que la description d'un algorithme en est une suite répétitive.

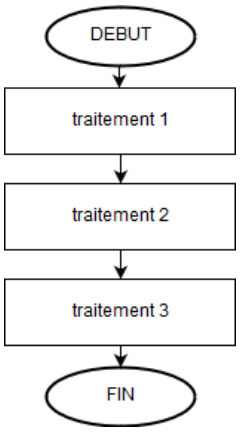
## 3. STRUCTURES ALGORITHMIQUES FONDAMENTALES :

Les opérations relatives à la résolution d'un problème peuvent en fonction de leur enchaînement, être organisées selon trois familles de structures :

- structures linéaires,
- structures alternatives,
- structures répétitives.

- **LA STRUCTURE LINÉAIRE OU SÉQUENCE**

La structure linéaire se caractérise par une suite d'actions à exécuter successivement dans l'ordre de leur énoncé.




Langage algorithmique

Début algorithme :

**FAIRE** traitement 1

**FAIRE** traitement 2

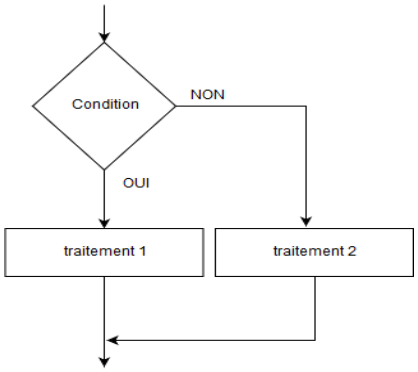
**FAIRE** traitement 3

Fin algorithme

- **LA STRUCTURE ALTERNATIVE OU SÉLECTION**

Une structure alternative n'offre que deux issues possibles s'excluant mutuellement. Les structures alternatives définissent une **fonction de choix** ou de **sélection** entre l'exécution de l'un ou de l'autre des deux traitements. Egalement désignées par **structures conditionnelles**, elles sont représentatives du **saut** ou rupture de séquence.

- La structure alternative complète




Langage algorithmique

**SI** condition VRAIE

**ALORS** traitement 1

**SINON** traitement 2

**FINSI**

Exemple de traduction en langage C :

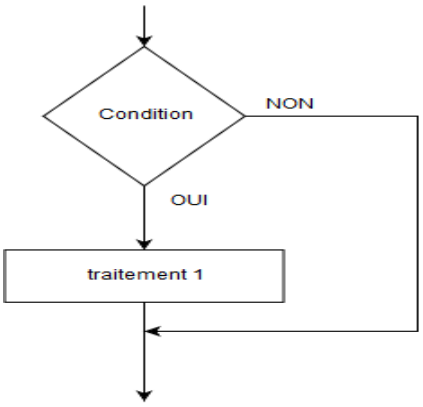
```c
int prix_voiture = 5500;

if(prix_voiture < 5000)
{
    printf("j'achete la voiture");
}
else
{
    printf("je n'achete pas la voiture");
}
```

Exemple de traduction en langage Python :

```python
prix_voiture = 5500

if prix_voiture < 5000 :
    print("j'achete la voiture")
else:
    print("je n'achete pas la voiture")
```

- **La structure alternative réduite :**

La structure alternative réduite se distingue de la précédente par le fait que seule la situation correspondant à la validation de la condition entraîne l'exécution du traitement. La situation opposée conduit à la sortie de la structure.




Langage algorithmique

**SI** condition VRAIE

**ALORS** traitement 1

**FINSI**

Exemple de traduction en langage C :

```c
int température = 34;

if(temperature = 35 )
{
    printf("il fait trop chaud");
}
printf("je continu les instructions");
```

Exemple de traduction en langage Python :

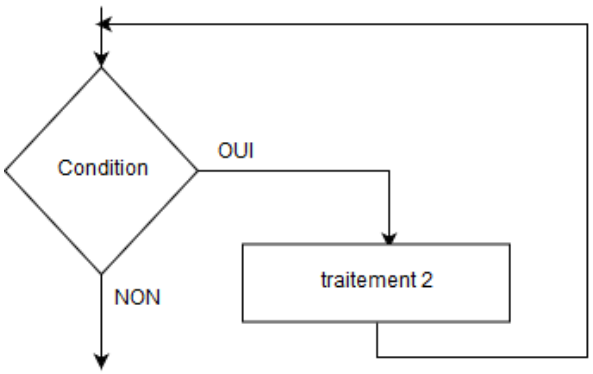
```python
température = 34

if temperature = 35 :
    print("il fait trop chaud")

print("je continu les instructions")
```

- **LES STRUCTURES RÉPÉTITIVES**

Une structure répétitive ou itérative répète l'exécution d'un traitement.

- **Boucle : TANT QUE ... FAIRE ...**

Dans cette structure on commence par tester la condition, si elle est vraie alors le traitement est exécuté.




Langage algorithmique

**TANT QUE** condition vraie

**FAIRE** traitement 1

**FIN TANT QUE**

Exemple de traduction en langage C :

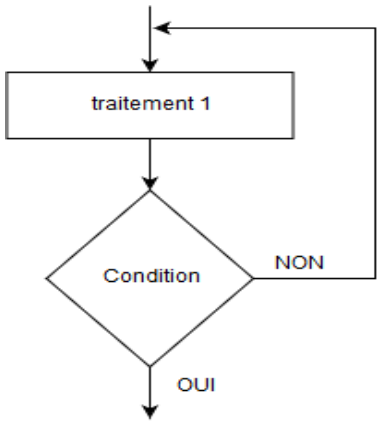
```c
while(position_volet == "ouvert")
{
    printf("volet ouvert");
}
printf("volet fermé");
```

Exemple de traduction en langage Python :

```python
while position_volet == "ouvert" :
    print("volet ouvert")

print("volet fermé")
```

- **Boucle : REPETER ... JUSQU'À ...**

Dans cette structure le **traitement est exécuté une première fois** puis sa répétition se poursuit jusqu'à ce que la condition soit vérifiée.




Langage algorithmique

RÉPÉTER

traitement 1

JUSQU'À condition VRAIE

Exemple de traduction en langage C :

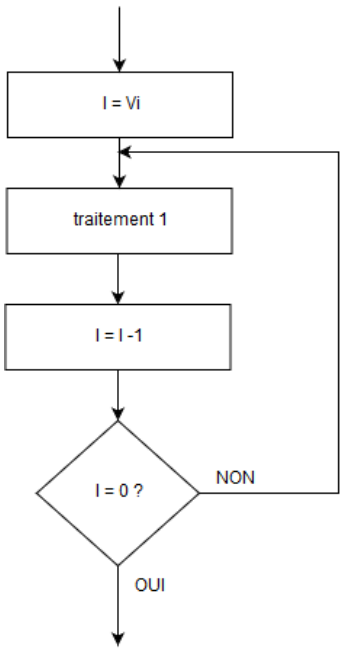
```c
float N;
do
   {
    printf("Introduisez un nombre entre 1 et 10 :");
    scanf("%f", &N);
   }
while (N<1 || N>10);
```

Cette structure **n'existe pas telle quelle en python.**

- **STRUCTURE RÉPÉTITION CONTRÔLÉE**

Dans cette structure la sortie de la boucle d'itération s'effectue lorsque le nombre souhaité de répétitions est atteint. D'où l'emploi d'une variable de boucle (indice i) caractérisée par :

- sa valeur initiale ;
- sa valeur finale ;
- son pas de variation.




Langage algorithmique

**POUR** I = Vi à 0 (par pas 1)

**FAIRE** traitement 1

**FIN POUR**

**En langage C :**

```c
for (int i = 0; i < 5; ++i)              //valeur initiale; valeur finale; pas de variation
{
    printf("la variable i vaut \n", i);
}
```

*Le résultat :*

```
la variable i vaut 0
la variable i vaut 1
la variable i vaut 2
la variable i vaut 3
la variable i vaut 4
```

*Remarque :*

i++ est un raccourci pour la commande i = i + 1

**En Python :**

```python
for i in range(0,5,1):
        print("la variable i vaut ", i)
```

*Remarque :*

i+= est un raccourci pour la commande i = i + 1

**Pour programmer :**

**Cours python sur les boucles :**

https://courspython.com/boucles.html

Avec essai dans python tutor pour la compréhension du déroulement du programme.

http://www.france-ioi.org/algo/course.php?idChapter=651&idCourse=2457

**Cours python sur les listes :**

https://python.doctor/page-apprendre-listes-list-tableaux-tableaux-liste-array-python-cours-debutant

## V) Programmation : les fonctions :

### 1. Définition :

Une fonction est un sous-programme qui peut s'exécuter de manière autonome.
Pour utiliser les fonctions, deux étapes sont nécessaires :

1. **Définir la fonction**
2. **Appeler la fonction**

- Les fonctions aident le programmeur à rester organisé.
- Les fonctions réduisent considérablement le nombre de lignes de code dans un fichier de programme, car ces sections de code sont réutilisées plusieurs fois.
- Les fonctions facilitent la réutilisation du code dans d'autres programmes,
- L'utilisation de fonctions rend souvent le code plus lisible.

### 2. Définir une fonction : Arduino :

En langage C Arduino, la syntaxe la plus courante pour définir une fonction est la suivante :

**Remarque :**

**Le mot clef return :**
Dans le code un mot clef return a deux actions :

- il stoppe l'exécution de la fonction
- il indique la valeur à renvoyer par la fonction.

Le mot clé return n'est pas obligatoire, s'il est absent, on parle de procédure (ne renvoie pas de valeur).

### 3. Définir une fonction : Python :

En langage python la syntaxe la plus courante pour définir une fonction est la suivante :

```
Def nom_de_la_fonction(argument1, argument2, …) :
        #code à exécuter
```

Cours :
https://python.doctor/page-apprendre-creer-fonction-en-python

**Application :**
Créer une fonction qui calcul l'aire d'un cercle, avec pour variable le rayon du cercle.

- Ecrire cette fonction en langage C sous Arduino avec le résultat S du calcul de l'aire indiqué dans le moniteur série.
- Ecrire cette fonction en langage python sous EduPython.

**Appeler le professeur pour vérifier et tester les fonctions lorsque celle-ci sont ecrites.**