
Team J.A.I.S
Accueil
A propos
Ecole
Cours
Contact
En savoir plus
Bienvenue sur le site de Team J.A.I.S
Découvrez nos profils, nos stages et nos cours interactifs en Algo, C et PHP
Découvrir notre équipe
Accéder aux cours

# À propos de Team J.A.I.S

Nous sommes des étudiants en BTS SIO SLAM passionnés par l'informatique et l'apprentissage collaboratif. Ce site présente nos profils, nos stages  ainsi que des cours et exercices interactifs en Algorithmique, C et PHP.

Accéder aux cours

Team J.A.I.S



En savoir plus

Team J.A.I.S est composé de quatre étudiants motivés : Ibrahima, Alixe, Joel et Saobane. Passionnés par l'informatique, nous avons créé ce site pour partager nos expériences de stage et proposer des cours interactifs en Algorithmique, C et PHP

Notre équipe

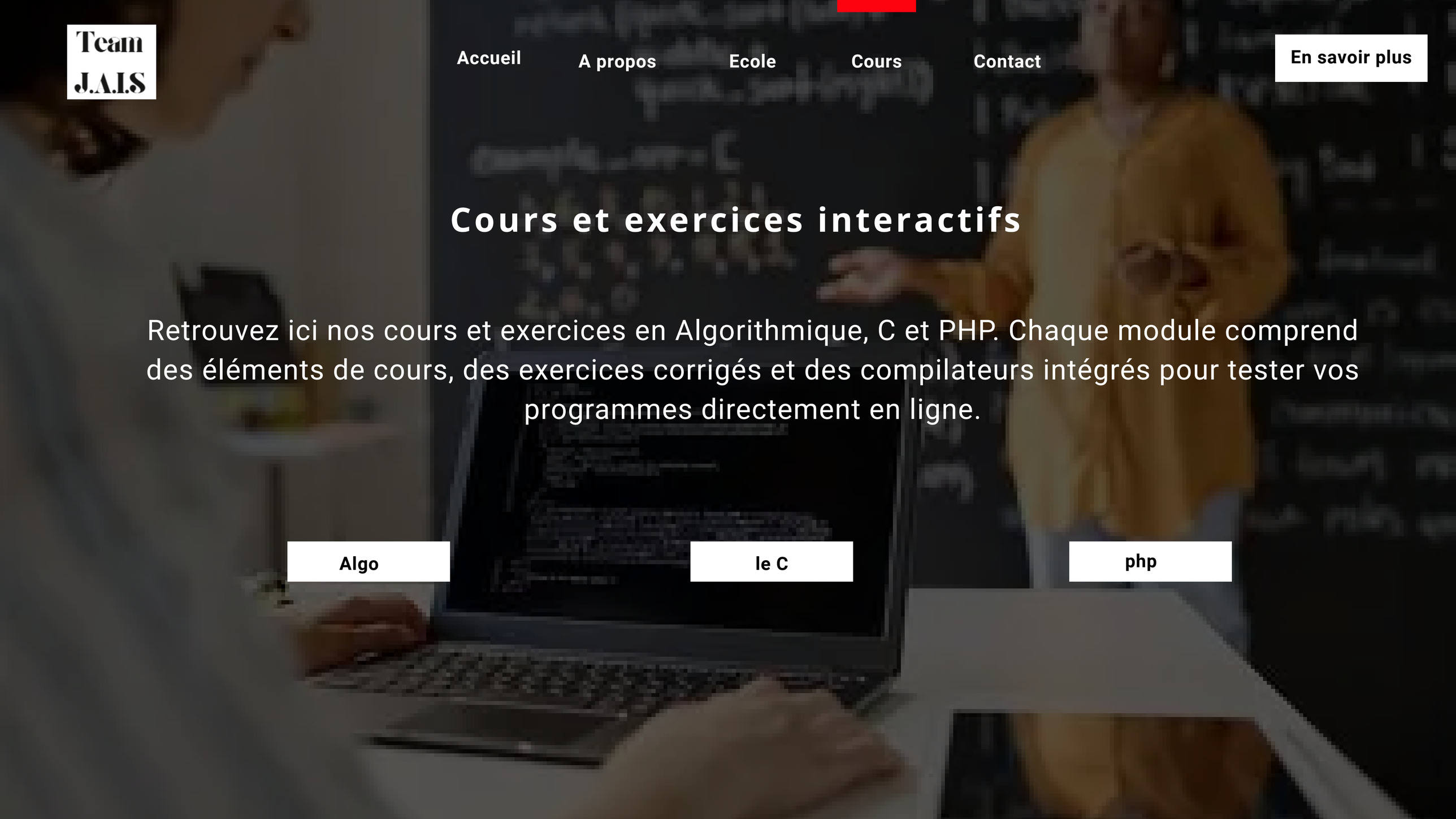
Team J.A.I.S



Je m’appelle Liamidi Saobane, étudiant en 2e année de BTS SIO SLAM à IRIS. Passionné par l’informatique, j’ai réalisé mon stage chez SAV PLUS Formation où j’ai pu créer et refaire la refonte de deux sites web. Cette expérience m’a permis de renforcer mes compétences et de mettre en pratique mes connaissances dans un contexte réel. Ce site présente mes compétences et mon parcours.

mon CV PDF | mon CV HTML | SAV PLUS | Ecole IRIS

Accéder aux cours



Team J.A.I.S

Accueil | A propos | Ecole | Cours | Contact

En savoir plus

# Cours et exercices interactifs

Retrouvez ici nos cours et exercices en Algorithmique, C et PHP. Chaque module comprend des éléments de cours, des exercices corrigés et des compilateurs intégrés pour tester vos programmes directement en ligne.

Algo

le C

php

# Cours : Algorithmique

## Qu’est-ce qu’un algorithme ? (Définition)

Un algorithme est une méthode précise et ordonnée qui permet de résoudre un problème ou d’accomplir une tâche.
Il s’agit d’une suite finie d’étapes qui, à partir de données d’entrée (les informations qu’on donne), produit un résultat (la solution).
L’algorithmique est la discipline qui consiste à concevoir, décrire et analyser ces algorithmes. Elle ne dépend d’aucun langage de programmation : on peut écrire un algorithme en français, en pseudo-code, sous forme de schéma ou de tableau.
En résumé, avant d’écrire du code dans un langage comme C, PHP ou Python.., on réfléchit d’abord à l’algorithme : c’est le plan qui guidera la programmation.

## Objectifs de l’algorithmique

L’algorithmique a pour but **d’analyser un problème** afin de comprendre clairement ce qu’on veut résoudre, **de concevoir une méthode efficace** en définissant étape par étape la solution avant de programmer, **de structurer la pensée** pour apprendre à raisonner de manière logique et ordonnée, **de décrire une solution de façon indépendante du langage** afin de pouvoir exprimer un algorithme sans se soucier du code (C, PHP, Python…), et enfin de préparer la programmation en servant de plan pour écrire ensuite un programme fiable et lisible

# Structure générale d'un algorithme

Algorithme Nom_de_l_algorithme
// Partie description (en-tête)
Variables : [liste des variables utilisées avec leur type]
Constantes : [liste des constantes si nécessaire]

// Partie traitement (corps de l'algorithme)
Début
  [Instructions d'initialisation]
  [Entrées : lecture des données]
  [Traitements : calculs, opérations, conditions, boucles…]
  [Sorties : affichage des résultats]
Fin

## Explication:

**Algorithme Nom_de_l_algorithme**
- C'est le titre de ton algorithme.
- On donne toujours un nom significatif pour savoir de quoi il parle.

**// Partie description (en-tête)**
- Ici, on décrit ce qu'on va utiliser.
- On prépare tous les éléments avant de commencer les étapes.

**Variables : [liste des variables utilisées avec leur type]**
- On liste les variables dont on aura besoin.
- Pour chacune, on précise son type (nombre, texte, etc.).
- Cela permet de savoir quelles données on manipule.

**Constantes : [liste des constantes si nécessaire]**
- On liste les valeurs fixes (constantes) utilisées dans l'algorithme.
- Une constante est une information qui ne change pas pendant l'exécution.

**// Partie traitement (corps de l'algorithme)**
C'est le cœur de l'algorithme : les instructions à suivre.

Début
- C'est le point de départ de ton algorithme.
- Tout ce qui est écrit après sera exécuté dans l'ordre.

**[Instructions d'initialisation]**
- On initialise (prépare) les variables si besoin.
- Ex. mettre une variable à zéro avant de commencer un calcul.

**[Entrées : lecture des données]**
- Ici on lit ou reçoit les données d'entrée (ce que l'utilisateur fournit, ce qu'on récupère).

**[Traitements : calculs, opérations, conditions, boucles…]**
- On effectue les opérations nécessaires pour résoudre le problème :
  - calculs
  - tests (conditions si/alors/sinon)
  - répétitions (boucles tant que/pour…)

**[Sorties : affichage des résultats]**
- On produit le ou les résultats.
- C'est la partie où l'on affiche ou retourne la solution obtenue.

**Fin**
- C'est la fin de l'algorithme.
- On n'écrit plus rien après.

# Les éléments de base du pseudo-code

Le pseudo-code est une manière d'écrire un algorithme en français structuré.
Il n'appartient à aucun langage (C, PHP, Python…), mais il ressemble à du code pour aider à comprendre la logique.
Voici les éléments essentiels :

- **L'affectation:**

  x ← 5
  On donne une valeur à une variable.
  Ici, on met la valeur 5 dans x.
  (Le symbole "←" se lit "reçoit" ou "devient".)

- **La lecture (Entrée):**

  Saisir x
  On récupère une donnée saisie par l'utilisateur et on la stocke dans la variable x.
  C'est l'entrée de l'algorithme.

- **L'affichage :**

  Afficher x
  On affiche le contenu de la variable x (le résultat).
  C'est la sortie de l'algorithme.

**Les structures de contrôle**

Elles permettent de contrôler l'ordre d'exécution des instructions :

- faire un choix
- répéter des actions

- **La condition (test):**

Permet de faire un choix selon un critère.
Syntaxe :

Si <condition> Alors
  [instructions si c'est vrai]
Sinon
  [instructions si c'est faux]
FinSi

- Si vérifie une condition (exemple : x > 0)
- Alors exécute un bloc si c'est vrai
- Sinon exécute un autre bloc si c'est faux
- FinSi marque la fin du test

- **La boucle POUR (répétition avec compteur)**

Permet de répéter un bloc d'instructions un nombre précis de fois.
Syntaxe :

Pour i allant de [début] à [fin] Faire
    [instructions à répéter]

FinPour

- i est un compteur (il augmente tout seul)
- On répète les instructions pour toutes les valeurs de début à fin

- **La boucle TANT QUE (répétition conditionnelle)**

Permet de répéter un bloc d'instructions tant qu'une condition est vraie.
Syntaxe :

TantQue <condition> Faire
    [instructions à répéter]
    [incrémentation ]
FinTantQue

- On teste la condition avant chaque tour
- La boucle s'arrête dès que la condition devient fausse

**Exercice 1:** Somme de deux nombres

Objectif : Écrire un algorithme qui :

1. Demande à l'utilisateur deux nombres
2. Calcule leur somme
3. Affiche le résultat

```
Algorithme SommeDeuxNombres
Variables :
   nombre1, nombre2, somme : Réel
Début
   // Saisie des nombres
   Saisir nombre1
   Saisir nombre2

   // Calcul de la somme
   somme ← nombre1 + nombre2

   // Affichage du résultat
   Afficher "La somme des deux nombres est : ", somme
Fin
```

**Exercice 2:** Vérifier si un nombre est pair ou impair

Objectif : Écrire un algorithme qui :

1. Demande à l'utilisateur un nombre
2. Vérifie s'il est pair ou impair
3. Affiche le résultat

```
Algorithme PairOuImpair
Variables :
   nombre : Entier
Début
   // Saisie du nombre
   Saisir nombre

   // Vérification si le nombre est pair ou impair
   Si nombre % 2 = 0 Alors
      Afficher "Le nombre est pair"
   Sinon
      Afficher "Le nombre est impair"
   FinSi
Fin
```

**Exercice 3**: Calcul de la moyenne de 10 nombres

Objectif : Écrire un algorithme qui :

1. Demande à l'utilisateur 10 nombres
2. Calcule la moyenne
3. Affiche la moyenne

```
Algorithme MoyenneDixNombres
Variables :
    i, nombre : Entier
    somme, moyenne : Réel
Début
    somme ← 0

    // Saisie des 10 nombres et calcul de la somme
    Pour i ← 1 à 10 Faire
        Saisir nombre
        somme ← somme + nombre
    FinPour

    // Calcul de la moyenne
    moyenne ← somme / 10

    // Affichage du résultat
    Afficher "La moyenne des 10 nombres est : ", moyenne
Fin
```

**Exercice 4:** Somme des éléments d'un tableau

Objectif : Écrire un algorithme qui :

1. Demande à l'utilisateur combien de nombres il veut saisir (taille du tableau)
2. Lit les nombres et les stocke dans un tableau
3. Calcule la somme de tous les éléments du tableau
4. Affiche le résultat

```
Algorithme SommeTableau
Variables :
    i, n, nombre : Entier
    somme : Réel
    tableau : Tableau de Réel
Début
    // Saisie de la taille du tableau
    Saisir n

    // Initialisation de la somme
    somme ← 0

    // Saisie des éléments du tableau et calcul de la somme
    Pour i allant de 1 à n Faire
        Saisir nombre
        tableau[i] ← nombre
        somme ← somme + nombre
    FinPour

    // Affichage du résultat
    Afficher "La somme des éléments du tableau est : ", somme
Fin
```

# Cours : le langage C

## Qu’est-ce que le C ? (Définition)

Le C est un langage de programmation structuré, généraliste et impératif, créé au début des années 1970 par Dennis Ritchie. Il permet de décrire de manière précise les instructions que l’ordinateur doit exécuter pour résoudre un problème ou réaliser une tâche. Le C est un langage compilé, ce qui signifie que le code écrit doit être traduit en langage machine par un compilateur avant de pouvoir être exécuté. Il est également très apprécié pour sa portabilité : un programme écrit en C peut être exécuté sur différents systèmes avec peu de modifications.
Le langage C se distingue par sa structuration, utilisant des fonctions et des blocs de code pour organiser les programmes, et par le contrôle précis de la mémoire, permettant au programmeur de gérer directement les ressources grâce aux pointeurs. Il est largement utilisé pour développer des logiciels systèmes, comme les systèmes d’exploitation, des applications performantes et des programmes nécessitant une gestion fine de la mémoire.

## Objectifs du C

Le langage C a été conçu pour permettre aux programmeurs de décrire précisément les instructions que l’ordinateur doit exécuter, tout en offrant une grande flexibilité et performance. Ses principaux objectifs sont de permettre la programmation système et applicative, en développant des logiciels systèmes comme les systèmes d’exploitation ainsi que des applications performantes. Il offre également un contrôle précis de la mémoire grâce aux pointeurs et à la gestion directe des ressources, ce qui permet d’optimiser l’utilisation de l’ordinateur.
Le C favorise la portabilité et la réutilisation du code : un programme écrit en C peut être exécuté sur différents systèmes avec peu de modifications, facilitant ainsi le partage et la maintenance. Il encourage aussi la structuration et l’organisation du code, en utilisant des fonctions et des blocs pour rendre le programme lisible, modulaire et facile à comprendre. Enfin, le C sert de base à l’apprentissage de la programmation, permettant de comprendre le fonctionnement interne des ordinateurs et constituant un excellent point de départ pour maîtriser d’autres langages de programmation.

# Cours : le langage C

## Structure générale du C

Un programme en C suit une structure précise qui permet au compilateur de comprendre et d'exécuter correctement les instructions. La structure générale comprend plusieurs parties essentielles :

1. **Les directives du préprocesseur**
   - Ces lignes commencent par # (comme #include <stdio.h>) et servent à inclure des fichiers ou définir des constantes avant la compilation.

1. **La déclaration des variables et constantes**
   - Avant d'utiliser des données, il faut déclarer les variables en précisant leur type (int, float, char, etc.) et, si nécessaire, définir des constantes.

1. **La fonction principale main()**
   - C'est le point de départ du programme. Toutes les instructions s'exécutent à partir de cette fonction.
   - Sa syntaxe de base est :

   - int main() {
   -  // instructions
   - return 0;
   - }

4. **Les instructions ou traitements**
   - Elles se trouvent à l'intérieur de la fonction main() ou d'autres fonctions.
   - C'est ici qu'on effectue les calculs, conditions, boucles, appels de fonctions, etc.

5. **Les fonctions supplémentaires**
   - Pour organiser et structurer le programme, on peut créer d'autres fonctions en dehors de main().
   - Elles permettent de réutiliser du code et de rendre le programme plus lisible.

# Cours : le langage C

## Structure générale du C

### Types de données:

Les types définissent la nature des données qu'une variable peut contenir.

**Exemples :**

```
int nombre = 5;       // entier
float prix = 10.5;    // nombre décimal
char lettre = 'A';    // caractère
double grandeValeur = 123.456789; // nombre décimal précis
```

### Variables:

Une variable est un espace de stockage pour une donnée, que l'on peut modifier pendant l'exécution du programme.

**Exemple :**

- `int age;` // déclaration d'une variable entière
- `age = 20;` // affectation d'une valeur

# Cours : le langage C

## Structure générale du C

**Constantes:**

Une constante est une valeur fixe qui ne change pas pendant l'exécution du programme

**Exemples :**

```
const float TAUX_TVA = 0.2;  // constante
#define PI 3.14159           // constante via préprocesseur
```

NB: Le préprocesseur lit ton code source et traite certaines instructions spéciales commençant par #, comme :

- #include → inclure des fichiers ou bibliothèques
- #define → définir des constantes symboliques
- #ifdef, #ifndef → conditions de compilation

Ces instructions ne font pas partie du langage C "normal", mais elles permettent de préparer le code avant qu'il soit compilé.

Exemple avec une constante :

```
#define PI 3.14159    // le préprocesseur remplace partout PI par 3.14159 avant compilation
```

- Les directives du préprocesseur commencent toujours par #
- Elles sont traitées avant la compilation, donc elles n'occupent pas de mémoire comme une variable normale
- Elles servent à définir des constantes, inclure des fichiers ou contrôler la compilation conditionnelle

# Cours : le langage C

## Opérateurs:

- Définition : Les opérateurs servent à effectuer des calculs ou des comparaisons.
- Exemples :

- int somme = 5 + 3;  //(+) opérateur arithmétique
- ==  // opérateur relationnel
- &&  // opérateur logique

## Structures de contrôle:

Elles permettent de modifier le flux d'exécution selon des conditions ou de répéter des actions.

**Exemples :**

```
// Conditionnelle
if(age >= 18) {
    printf("Adulte\n");
} else {
    printf("Mineur\n");
}

// Boucle
for(int i = 1; i <= 5; i++) {
    printf("%d\n", i);
}
```

**Fonctions:**

Une fonction est un bloc d'instructions qui réalise une tâche spécifique et peut être réutilisé.

- Exemple :

- int addition(int a, int b) {
- return a + b;
- }

- int resultat = addition(5, 3);
- printf("%d", resultat);

- Le mot int avant le nom de la fonction indique que la fonction va retourner un entier.
- Le type de retour doit correspondre à ce que la fonction renverra avec return.
- addition est le nom choisi pour la fonction, il doit être explicite et refléter ce qu'elle fait.
- On utilise ce nom pour appeler la fonction plus tard dans le programme.

(int a, int b)

- Ce sont les données que la fonction reçoit pour effectuer son calcul.
- Ici a et b sont des paramètres de type entier (int).
- Les paramètres sont utilisés à l'intérieur de la fonction comme des variables locales

- {
- return a + b;
- }
- Les instructions à l'intérieur des { } définissent ce que la fonction fait.
-
- Ici, la fonction additionne a et b.
-
- return renvoie le résultat à l'endroit où la fonction a été appelée.

int resultat = addition(5, 3);
On appelle la fonction addition avec les valeurs 5 et 3.

La fonction retourne 8 qui est stocké dans la variable resultat.

**Entrées et sorties:**

Permettent de communiquer avec l'utilisateur pour lire des données ou afficher des résultats.

**Exemples :**

- int age;
- printf("Entrez votre âge : ");
- scanf("%d", &age);  // lecture

printf("Votre âge est : %d\n", age); // affichage

## Exercices sur les fonctions en C:

Exercice 1 – Addition de deux nombres
Objectif : Écrire une fonction addition qui :

1. Prend deux nombres en paramètre
2. Retourne leur somme
3. Affiche le résultat dans main()

```c
#include <stdio.h>

// Fonction qui additionne deux nombres
int addition(int a, int b) {
    return a + b;
}

int main() {
    int x, y, resultat;

    printf("Entrez le premier nombre : ");
    scanf("%d", &x);

    printf("Entrez le deuxieme nombre : ");
    scanf("%d", &y);

    resultat = addition(x, y);

    printf("La somme est : %d\n", resultat);

    return 0;
}
```

Exercice 2 – Vérifier si un nombre est pair

Objectif : Écrire une fonction estPair qui :

1. Prend un nombre entier en paramètre
2. Retourne 1 si le nombre est pair et 0 sinon
3. Affiche le résultat dans main()

```c
#include <stdio.h>

// Fonction qui vérifie si un nombre est pair
int estPair(int n) {
    if (n % 2 == 0) {
        return 1;  // vrai si pair
    } else {
        return 0;  // faux si impair
    }
}

int main() {
    int nombre;

    printf("Entrez un nombre : ");
    scanf("%d", &nombre);

    if (estPair(nombre)) {
        printf("Le nombre %d est pair.\n", nombre);
    } else {
        printf("Le nombre %d est impair.\n", nombre);
    }

    return 0;
}
```

Exercice 3 – Calcul de la moyenne de trois nombres

Objectif : Écrire une fonction moyenne qui :

1. Prend trois nombres en paramètre
2. Calcule et retourne leur moyenne
3. Affiche la moyenne dans main()

```c
#include <stdio.h>

// Fonction qui calcule la moyenne de trois nombres
float moyenne(float a, float b, float c) {
    return (a + b + c) / 3;
}

int main() {
    float n1, n2, n3, moy;

    printf("Entrez le premier nombre : ");
    scanf("%f", &n1);

    printf("Entrez le deuxieme nombre : ");
    scanf("%f", &n2);

    printf("Entrez le troisieme nombre : ");
    scanf("%f", &n3);

    moy = moyenne(n1, n2, n3);

    printf("La moyenne est : %.2f\n", moy);

    return 0;
}
```

# Cours : le PHP

## Qu’est-ce que le PHP ? (Définition)

Le PHP (Hypertext Preprocessor, anciennement Personal Home Page) est un langage de programmation libre et open-source spécialement conçu pour le développement d’applications web dynamiques. Il est exécuté côté serveur (server-side), ce qui signifie que le code PHP est interprété sur le serveur avant d’être envoyé sous forme de page HTML au navigateur de l’utilisateur.

Créé en 1994 par Rasmus Lerdorf, PHP est aujourd’hui l’un des langages les plus populaires pour développer des sites web interactifs, des applications web et des systèmes de gestion de contenu (CMS) comme WordPress, Drupal ou Joomla.

PHP est intégré directement dans le HTML, ce qui permet de mélanger facilement du code HTML (structure de la page) et du code PHP (logique et traitement). Il est capable de :

- Gérer des formulaires (collecter et traiter des données envoyées par les utilisateurs)
- Interagir avec des bases de données (comme MySQL) pour stocker et récupérer des informations
- Générer des pages dynamiques dont le contenu change en fonction de l’utilisateur, du moment ou des données reçues
- Gérer les sessions et cookies pour authentifier les utilisateurs ou stocker des préférences
- Envoyer des e-mails, manipuler des fichiers sur le serveur, etc.

PHP est aussi multiplateforme (il fonctionne sous Windows, Linux, macOS) et très largement supporté par les hébergeurs web. Son apprentissage est relativement accessible et sa communauté est vaste, offrant de nombreuses bibliothèques et frameworks (Laravel, Symfony, CodeIgniter…) pour accélérer le développement.

# Cours : le PHP

## Objectifs du PHP

Le langage PHP a été conçu pour simplifier et automatiser la création de pages web dynamiques et d'applications web interactives. Ses principaux objectifs sont :

- Générer des pages web dynamiques : permettre d'afficher un contenu qui change en fonction des données reçues, des actions de l'utilisateur ou du contexte (date, heure, connexion…).
- Traiter des formulaires et des données utilisateur : recueillir des informations envoyées par des formulaires HTML et les exploiter (calculs, enregistrements, traitements divers).
- Interagir avec des bases de données : lire, écrire, modifier ou supprimer des données dans des bases (comme MySQL, PostgreSQL), ce qui est essentiel pour les sites interactifs.
- Gérer les sessions et l'authentification : offrir des fonctionnalités comme l'inscription, la connexion d'utilisateurs et la mémorisation des préférences grâce aux sessions et aux cookies.
- Automatiser des tâches côté serveur : envoi d'e-mails, génération de fichiers, gestion des fichiers sur le serveur, création d'API…
- S'intégrer facilement avec le HTML et les autres technologies web : mélanger du code HTML, CSS, JavaScript et PHP pour construire des applications complètes.

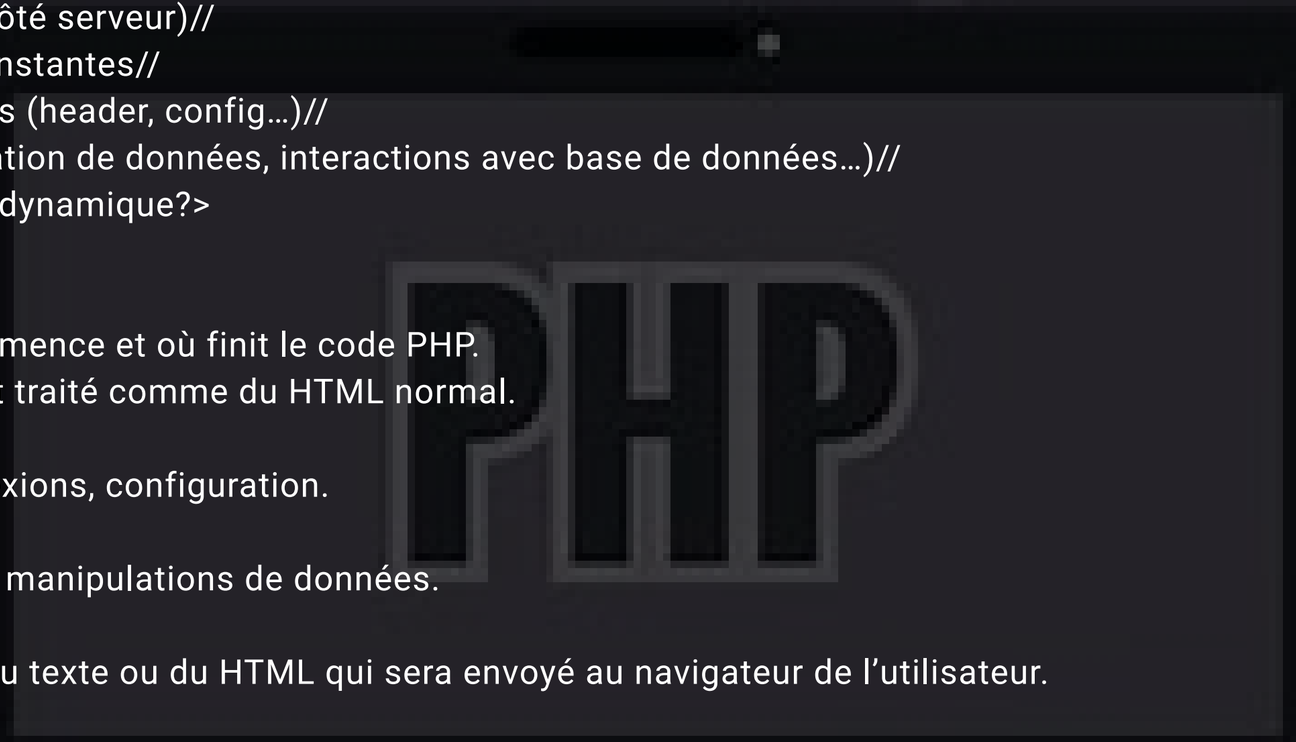
PHP vise à rendre les sites web interactifs, personnalisés et connectés aux données, tout en simplifiant le travail du développeur et en restant accessible et rapide à mettre en œuvre.

# Cours : le PHP

## Structure générale de PHP

Tout fichier PHP suit une organisation basique. On l'écrit dans un fichier avec l'extension .php et le code est placé entre des balises spéciales :

<?php// Zone PHP (code exécuté côté serveur)//
Déclaration de variables et constantes//
Inclusion éventuelle de fichiers (header, config…)//
Traitements (calculs, récupération de données, interactions avec base de données…)//
Génération du contenu HTML dynamique?>




Balises PHP <?php … ?>

- – Indiquent au serveur où commence et où finit le code PHP.
- – Tout ce qui est en dehors est traité comme du HTML normal.
- Déclarations / Initialisations
- – Variables, constantes, connexions, configuration.
- Traitements
- – Calculs, conditions, boucles, manipulations de données.
- Affichage (sortie)
- – Le code PHP peut produire du texte ou du HTML qui sera envoyé au navigateur de l'utilisateur.
- Mélange HTML / PHP
- – On peut alterner du HTML pur et du PHP pour insérer dynamiquement du contenu.

Un script PHP démarre toujours par <?php et se termine par ?> (le dernier peut être omis dans les fichiers purement PHP), et à l'intérieur on structure le code comme un "plan" : déclarations → traitement → affichage.

# Cours : le PHP

## Les éléments de base de PHP

Les variables

- Définition : Une variable est un espace mémoire qui permet de stocker une valeur.
- En PHP, toutes les variables commencent par le signe $.
- Exemple :

```
<?php$nom = "Saobane";  // chaîne de caractères
$age = 20;         // entier?>
```

Les constantes

- Définition : Valeur fixe qui ne change pas pendant l’exécution du script.
- Déclaration avec define() ou const :

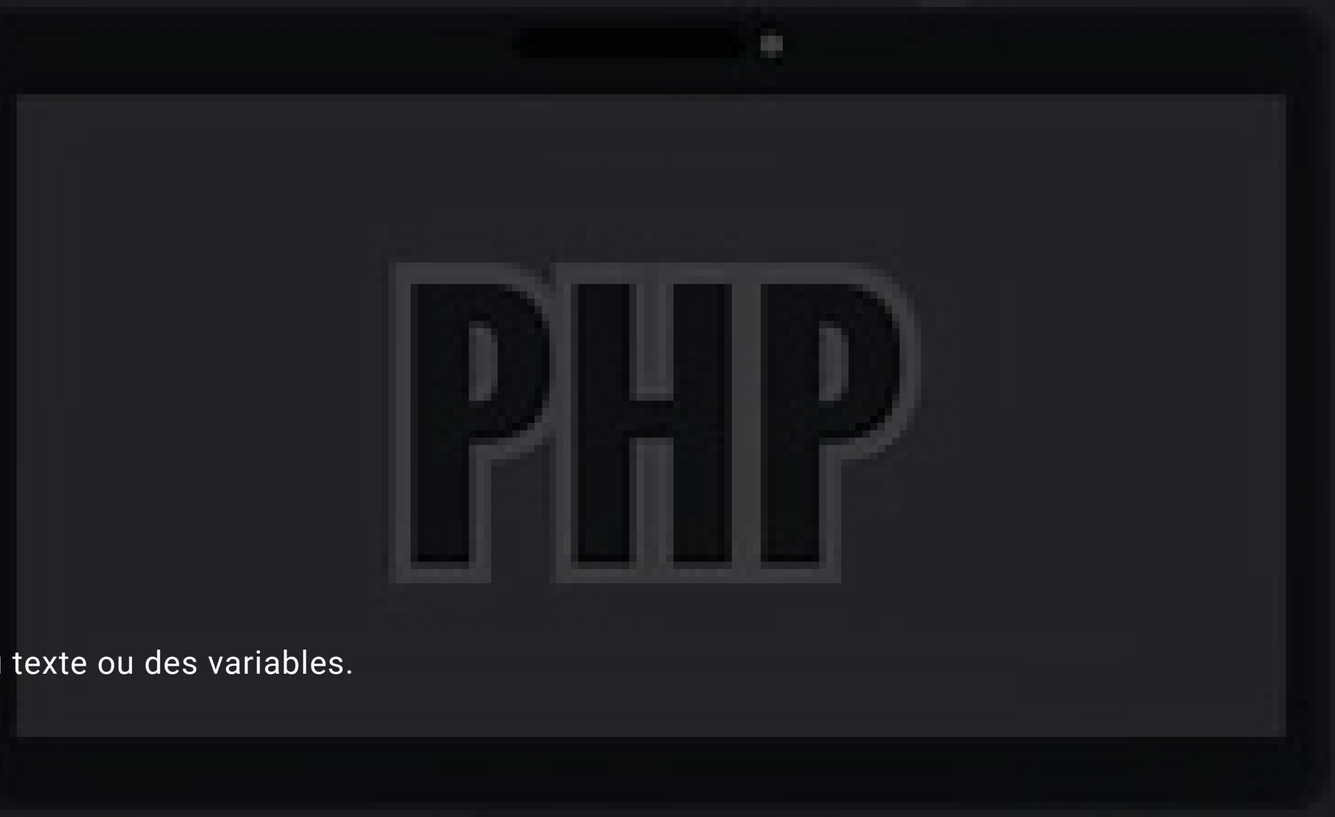
```
<?phpdefine("PI", 3.14);    // constanteconst TVA = 0.2;       // constanteecho PI;
?>
```

# Cours : le PHP

## Les éléments de base de PHP

Les types de données principaux

- Chaînes de caractères (string)
- Entiers (int)
- Flottants (float)
- Booléens (true/false)
- Tableaux (array)
- Objets (object)




Exemple :

```
<?php$texte = "Bonjour";
$nombre = 42;
$prix = 10.5;
$valide = true;
?>
```

**L’affichage:**

echo ou print permettent d’afficher du texte ou des variables.

```
<?php
$nom = "Saobane";
echo "Bonjour " . $nom;
?>
```

# Cours : le PHP

## La lecture (entrées)

- En PHP côté serveur, on récupère les données d'un formulaire ou de l'URL avec $_GET ou $_POST.

<?php$nom = $_POST['nom']; // récupère le champ "nom" d'un formulaire envoyé en POST?>

## Les conditions

**Permettent d'exécuter des instructions selon une situation.**

```
<?php
$age = 20;
if ($age >= 18) {
    echo "Majeur";
} else {
    echo "Mineur";
}
?>
```

## Les boucles

**Boucle for :**

```
for ($i = 1; $i <= 5; $i++) {
    echo $i;
}
```

**Boucle while :**

```
$x = 1;
while ($x <= 5) {
    echo $x;
    $x++;
}
```

**Boucle foreach (parcourir un tableau) :**

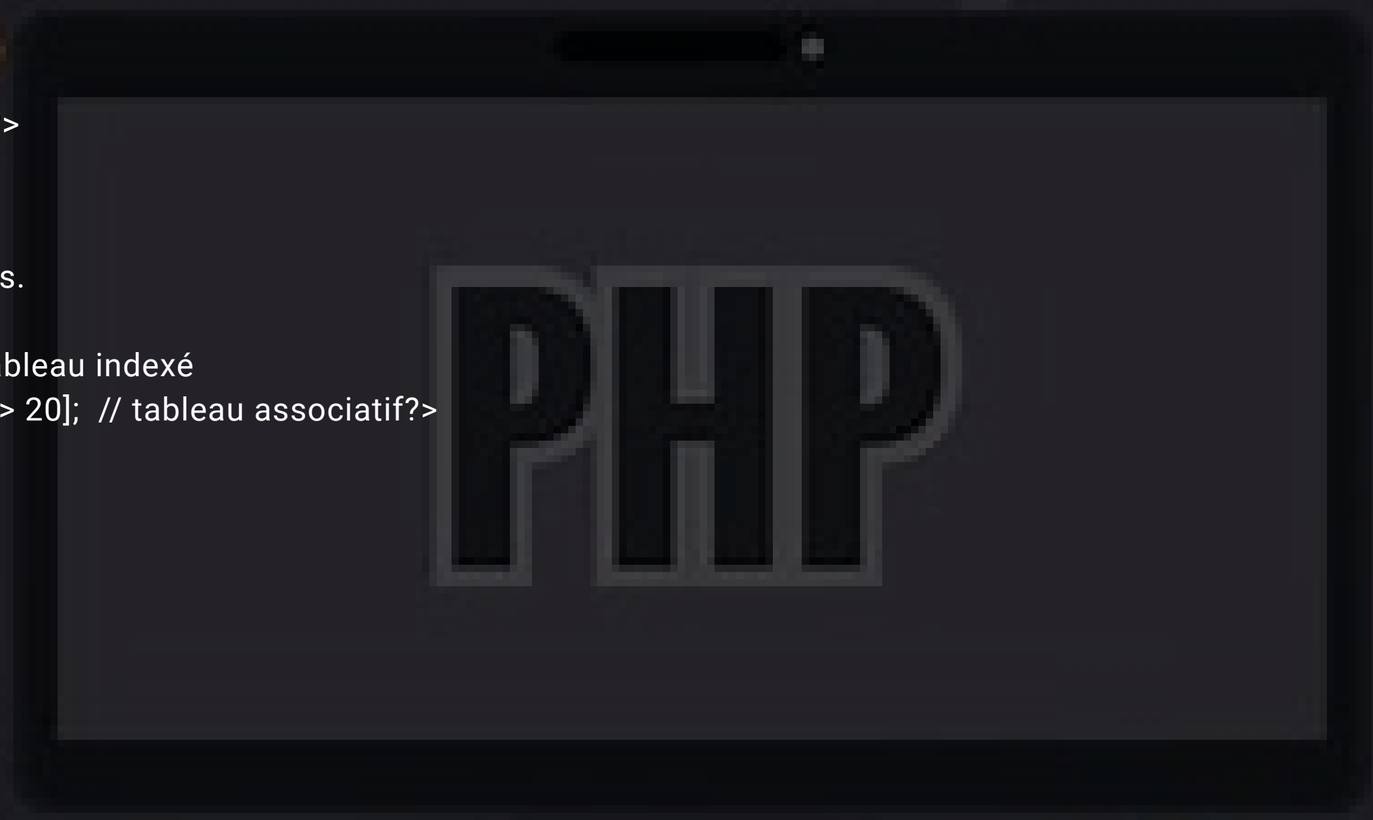
```
$prenoms = ["Ibrahima", "Alixe", "Joël", "Saobane"];
foreach ($prenoms as $p) {
    echo $p;
}
```

# Cours : le PHP

## Les fonctions

Bloc de code réutilisable qui exécute une tâche précise.

```
<?phpfunction addition($a, $b) {
    return $a + $b;
}
echo addition(5, 3); // Affiche 8?>
```




### Les tableaux

Variable qui contient plusieurs valeurs.

```
<?php$notes = [15, 12, 18];          // tableau indexé
$eleve = ["nom" => "Saobane", "age" => 20];  // tableau associatif?>
```

Exercice 1: Somme de deux nombres

Objectif :

Demander à l’utilisateur deux nombres via un formulaire et afficher leur somme.

📝 Énoncé :

- Créer un formulaire HTML qui saisit deux nombres.
- Récupérer ces nombres en PHP (avec $_POST).
- Calculer leur somme.
- Afficher le résultat.

```
<!-- somme.php -->
<form method="post">
    Nombre 1 : <input type="number" name="nb1"><br>
    Nombre 2 : <input type="number" name="nb2"><br>
    <input type="submit" value="Calculer la somme">
</form>

<?php
if (isset($_POST['nb1']) && isset($_POST['nb2'])) {
    $a = $_POST['nb1'];
    $b = $_POST['nb2'];
    $somme = $a + $b;
    echo "La somme est : " . $somme;
}
?>
```

Exercice 2 – Vérifier si un nombre est pair ou impair
Objectif :
Demander un nombre à l'utilisateur et indiquer s'il est pair ou impair.
✅ Correction :

```
<!-- pair.php -->
<form method="post">
    Entrez un nombre : <input type="number" name="nb"><br>
    <input type="submit" value="Vérifier">
</form>

<?phpif (isset($_POST['nb'])) {
    $n = $_POST['nb'];
    if ($n % 2 == 0) {
        echo "Le nombre $n est pair.";
    } else {
        echo "Le nombre $n est impair.";
    }
}
?>
```

Exercice 3 – Calcul de la moyenne d’un tableau de notes
Objectif :
Définir un tableau de notes, calculer la moyenne et l’afficher.
✅ Correction :

```
<?php// Tableau de notes$notes = [12, 15, 9, 18, 14, 16];

// Calcul de la somme$somme = 0;
foreach ($notes as $note) {
    $somme += $note;
}

// Calcul de la moyenne$moyenne = $somme / count($notes);

// Affichageecho "La moyenne des notes est : " . $moyenne;
?>
```