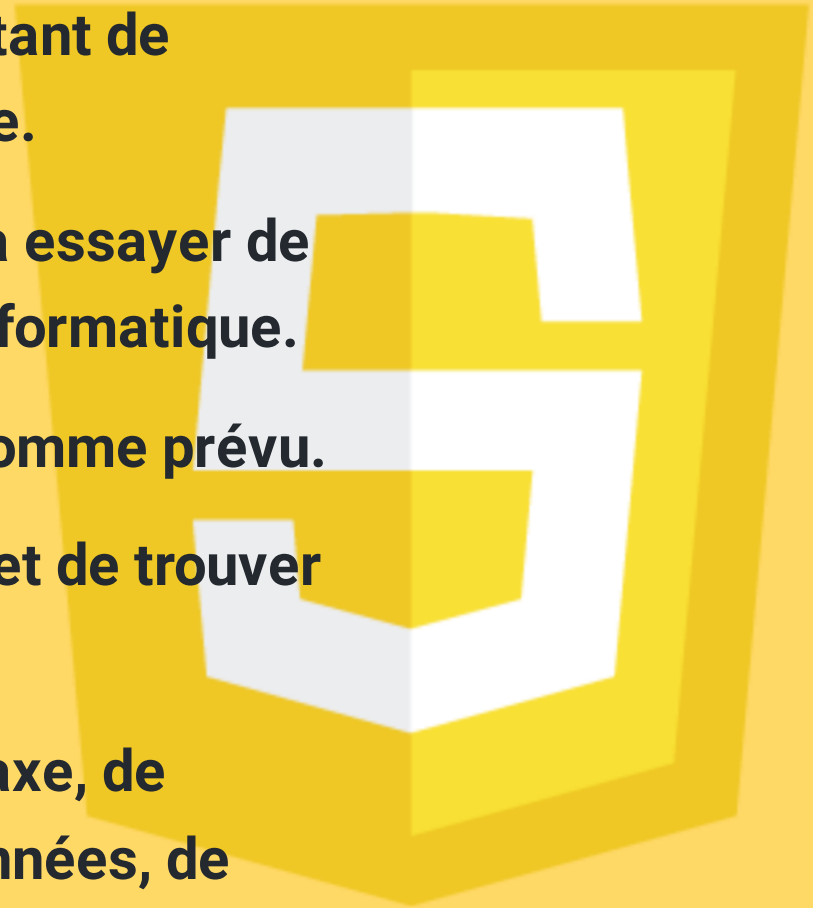
# Qu'est-ce que le debugage



- 📌 Le debugage est l'ensemble des techniques permettant de trouver les erreurs dans un programme informatique.
- 📌 Le debugage est un processus itératif qui consiste à essayer de comprendre ce qui ne va pas dans un programme informatique.
- 📌 Il survient quand le programme ne fonctionne pas comme prévu.
- 📌 Il s'agit d'une démarche active et réflexive qui permet de trouver des solutions aux problèmes rencontrés.
- 📌 Il permet de trouver des erreurs de logique, de syntaxe, de compilation, de configuration, de ressources, de données, de documentation, de conception, de conception, de conception, etc.

# Le debugage et les outils de debugage

- 📌 Le debugage peut être très chronophage, frustrant et fastidieux.
- 📌 Il est donc important de minimiser le temps passé à le faire.
- 📌 Le debugage peu se faire en utilisant des logs afin de retracer les étapes du programme.
- 📌 Le debugage peut se faire en utilisant des outils de debugage.

# Comment debuguer

- 📌 Il est important de comprendre les raisons qui ont mené à ce que le programme ne fonctionne pas comme prévu.
- 📌 Cela permet de mieux comprendre les problèmes et de pouvoir apporte des solutions.
- 📌 Il faut retracer les étapes du programme afin de comprendre ce qui ne va pas.

# Configuration de Visual Studio Code afin de déboguer depuis l'IDE

- 📌 Il est possible de déboguer un programme depuis Visual Studio Code.
- 📌 Il faut d'abord installer l'extension "Debugger for Chrome".
- 📌 Presser F5 pour lancer le débogage.
- 📌 Il est possible de mettre des points d'arrêts dans le code afin de pouvoir suivre l'exécution du programme.

# TP

```
// Exemple de code qui fais un tapis
function tapis(n) {
        let str = "";
        for (let i = 0; i < n - 1; i++) {
                for (let j = 0; j < n - 1; j++) {
                        if (i % 2 == 0) {
                                if (j % 2 == 0) {
                                        str += "*";
                                } else {
                                        str += "#";
                                }
                        } else {
                                if (j % 2 == 0) {
                                        str += "#";
                                } else {
                                        str += "*";
                                }
                        }
                }
        }
        return str;
}

console.log(tapis(5));
```

# Rappels importants du langage

# Les éléments de premier ordre dans JavaScript (variables, fonctions, objets, tableaux, etc.)

# String

- 📌 Les strings sont des chaîne de caractères
- 📌 Les strings sont délimités par des guillemets simples ou doubles ainsi que par des backticks
- 📌 Les strings peuvent être concaténées avec l'opérateur `+`

Les strings ont des propriétés et des méthodes :

- 📌 `length` : retourne la longueur de la chaîne de caractères
- 📌 `split()` : sépare une chaîne de caractères en un tableau de sous-chaînes
- 📌 `join()` : rassemble les éléments d'un tableau en une chaîne de caractères

# Number

- 📌 **Les number sont des nombres**
- 📌 **Les number peuvent être des entiers ou des décimaux**
- 📌 **Les number peuvent être des nombres spéciaux (Infinity, -Infinity, NaN)**

**Les number ont des propriétés et des méthodes :**

- 📌 `toString()` **: convertit un nombre en chaîne de caractères**
- 📌 `parseInt()` **: convertit une chaîne de caractères en nombre**
- 📌 `parseFloat()` **: convertit une chaîne de caractères en nombre décimal**

# Boolean

- 📌 **Les boolean sont des valeurs booléennes**
- 📌 **Les boolean peuvent être `true` ou `false`**
- 📌 **Si la valeur est omise ou est 0, null, NaN, undefined ou une chaîne de caractères vide ("") -> l'objet a une valeur initiale de false (faux).**

# Null et undefined

- 📌 null est une valeur spéciale qui signifie "valeur nulle"
- 📌 undefined est une valeur spéciale qui signifie "valeur non définie"
- 📌 null est une valeur qui est assignée
- 📌 undefined est une valeur qui est automatiquement assignée par JavaScript

# Symbol

- 📌 **Les symbol ont été introduits dans ES6 et ont donc une utilisation limitée pour le moment.**
- 📌 **Les symbol sont des valeurs uniques et immuables**
- 📌 **Elles sont utilisées comme clés de propriété pour les objets afin d'éviter les conflits de noms de propriétés**

```
const identifiant = Symbol('clef1');
const password = Symbol('clef2');

let utilisateur = {
    [identifiant] : 'Pierrelegrand',
    [password] : "1234mdpsupersecretpourpierre"
};
```

# Les fonctions

- 📌 Les fonctions sont des blocs de code qui peuvent être appelés à plusieurs reprises dans un programme
- 📌 elles peuvent recevoir des paramètres (ou non)
- 📌 elles peuvent retourner des valeurs (ou non)
- 📌 elles peuvent être nommées ou anonymes
- 📌 elles peuvent etre retournées par d'autres fonctions et manipulées comme des variables

# Les objets

- 📌 Les objets sont des conteneurs qui peuvent contenir des propriétés et des méthodes
- 📌 Il est délimité par des accolades
- 📌 Il est composé de paires clé/valeur

{clef1: valeur1, clef2: valeur2, clef3: valeur3 ...}

- 📌 Les objets peuvent être imbriqués

# Les tableaux

- 📌 **Les tableaux sont des listes de valeurs**
- 📌 **Ils sont délimités par des crochets `[ ]`**
- 📌 **Ils peuvent etre vides ou imbriqués**
- 📌 **Ils peuvent contenir des valeurs de types différents**
- 📌 **Les methodes `push()` et `pop()` permettent d'ajouter et de supprimer des éléments à la fin du tableau**
- 📌 **Les methodes `shift()` et `unshift()` permettent d'ajouter et de supprimer des éléments au début du tableau**
- 📌 **Les methodes `splice()` et `slice()` permettent d'ajouter et de supprimer des éléments à une position donnée du tableau**

# La portée des données

# Les variables globales

- 📌 **Les variables globales sont des variables qui sont déclarées en dehors de toute fonction**
- 📌 **Elles sont accessibles partout dans le programme**
- 📌 **Elles sont déclarées avec le mot clé `var` ou `let` ou `const`**
- 📌 **Elles peuvent être modifiées à tout moment**

# Les variables locales

- 📌 **Les variables locales sont des variables qui sont déclarées à l'intérieur d'une fonction**
- 📌 **Elles sont accessibles uniquement dans la fonction dans laquelle elles sont déclarées**
- 📌 **Elles sont déclarées avec le mot clé `var` ou `let` ou `const`**
- 📌 **Si elles sont utilisées en dehors de la fonction, elles sont considérées comme non définies `ReferenceError: mavariable is not defined`**

# Les fonctions

# Les differents types de fonctions

- 📌 Les fonctions nommées
- 📌 Les fonctions asynchrones
- 📌 Les fonctions anonymes
- 📌 Les fonctions fléchées
- 📌 Les fonctions imbriquées
- 📌 Les fonctions auto-exécutées
- 📌 Les fonctions de rappel

# Petit rappel sur les fonctions

# La syntaxe generale d'une fonction

```
function nomDeMaFonction(parametre1, parametre2, parametre3) {
                // code à executer
                return valeurARetourner;
        }
```

# Les paramètres

- 📌 **Les paramètres sont des variables qui sont déclarées dans la définition de la fonction**
- 📌 **Les paramètres peuvent avoir une valeur par défaut**
- 📌 **Les paramètres peuvent être optionnels**
- 📌 **Si le nombre d'arguments est inférieur au nombre de paramètres, les paramètres non renseignés auront la valeur** `undefined`
- 📌 **Si le nombre d'arguments est supérieur au nombre de paramètres, les arguments supplémentaires sont ignorés**

# Les valeurs de retour

- 📌 **Les fonctions peuvent retourner une valeur avec le mot clé `return`**
- 📌 **Si la fonction ne contient pas de mot clé `return`, elle retourne `undefined`**
- 📌 **Si la fonction contient plusieurs mots clés `return`, seul le premier sera pris en compte**
- 📌 **Nous pouvons retourner n'importe quel type de données (function y compris)**

# Les fonctions nommées

- 📌 **Les fonctions nommées sont des fonctions qui sont déclarées avec le mot clé `function`**
- 📌 **Elles ont un nom et peuvent être appelées à partir de ce nom**
- 📌 **Elles peuvent être appelées avant leur déclaration**

# Les fonctions anonymes

- 📌 Les fonctions anonymes sont des fonctions qui sont déclarées sans nom
- 📌 Elles ne peuvent pas être appelées à partir de leur nom
- 📌 Elles peuvent etre affectées à une variable ou à une propriété d'un objet pour être appelées
- 📌 Elles peuvent etre passées en paramètre d'une autre fonction pour être appelées

# Les fonctions fléchées

- 📌 Les fonctions fléchées sont des fonctions qui sont déclarées avec le mot clé `=>`
- 📌 Elles sont plus courtes que les fonctions nommées
- 📌 Elles n'ont pas de mot clé `return`, il est implicite (sauf si on utilise des accolades)
- 📌 Le mot clef `this` va dans ce cas être substitué par l'objet utilisant la méthode lors de son appel.
- 📌 La valeur de this ne va pas dépendre de l'endroit où la méthode a été déclarée mais de l'objet qui l'appelle.

# Les fonctions imbriquées

- 📌 **Les fonctions imbriquées sont des fonctions qui sont déclarées à l'intérieur d'une autre fonction**
- 📌 **Elles sont accessibles uniquement dans la fonction dans laquelle elles sont déclarées**

# Les fonctions auto-exécutées

- 📌 Les fonctions auto-exécutées sont des fonctions qui sont déclarées et exécutées en même temps
- 📌 Elles peuvent être utiles pour créer un espace de nommage
- 📌 Elles peuvent être utiles pour ne pas polluer l'espace global

# Les fonctions de rappel

- 📌 Les fonctions de rappel sont des fonctions qui sont passées en paramètre d'une autre fonction
- 📌 Elles sont appelées à l'intérieur de la fonction qui les a reçues
- 📌 Elles peuvent être anonymes ou pas
- 📌 Elles peuvent être appelées plusieurs fois

# Les fonctions asynchrones

- 📌 Les fonctions asynchrones sont des fonctions qui permettent d'exécuter du code de manière asynchrone (en parallèle)
- 📌 Elles sont déclarées avec le mot clé `async`
- 📌 Le mot-clé await est uniquement valide au sein des fonctions asynchrones. Si ce mot-clé est utilisé en dehors du corps d'une fonction asynchrone
- 📌 Le mot-clé await fait en sorte que l'exécution de la fonction asynchrone soit suspendue jusqu'à ce que la Promise soit résolue

# Pièges du langage

# Typage faible

- 📌 **Le typage faible signifie que le langage ne vérifie pas le type des variables**
- 📌 **Le typage faible signifie que le langage ne vérifie pas le type des paramètres**
- 📌 **Le typage faible signifie que le langage ne vérifie pas le type des valeurs de retour**
- 📌 **Si on utilise le mot clé `use strict`, le typage faible est désactivé**

# "Hoisting"

## Qu'est-ce que le hoisting ?

**Le hoisting est un mécanisme par lequel les déclarations de variables et de fonctions sont déplacées en haut de leur portée (scope) avant l'exécution du code.**

**Le hoisting n'affecte que les déclarations et pas les initialisations.**

# Exemple

```
function test() {
  console.log(a); // undefined
  console.log(foo()); // 2

  var a = 1;
  function foo() {
    return 2;
  }
}

test();
```

# Quel est l'avantage du hoisting ?

**Le hoisting permet de déclarer les variables et les fonctions où elles sont utilisées.**

# Quel est le problème du hoisting ?

**Le hoisting peut être source d'erreurs.**

# Comment éviter le hoisting ?

- 📌 On peut utiliser le mot clé `use strict` pour désactiver le hoisting
- 📌 On peut déclarer les variables et les fonctions en haut de leur portée
- 📌 On peut déclarer les variables et les fonctions avec `let` et `const`

# Programmation objet

- 📌 La programmation objet est un paradigme de programmation qui permet de modéliser des objets du monde réel
- 📌 Nous pouvons créer des objets avec des propriétés et des méthodes
- 📌 Nous pouvons créer des classes qui vont nous permettre de créer des objets de manière plus simple

# Différentes façons de créer des objets

# Object literal

```
let person = {
  name: 'John',
  age: 30,
  greet: function() {
        console.log('Hello');
  }
};
```

# Constructeurs

```
function Voiture(fabricant, modele, annee) {
  this.fabricant = fabricant;
  this.modele = modele;
  this.annee = annee;
}


let maVoiture = new Voiture('Ford', 'Mustang', 1969);
```

# Valeur de "this" dans un objet

- 📌 La valeur de `this` dans un objet est l'objet lui-même
- 📌 On peut utiliser `this` pour accéder aux propriétés et aux méthodes de l'objet
- 📌 On peut egalement ajouter des propriétés et des méthodes à l'objet avec `this`

# Prototype

- 📌 **Le prototype est un objet qui va être utilisé comme modèle pour créer d'autres objets**
- 📌 **Le prototype est utilisé pour définir des propriétés et des méthodes qui seront partagées par tous les objets créés à partir de ce prototype**

# Exemple

```
function Voiture(fabricant, modele, annee) {
        this.fabricant = fabricant;
        this.modele = modele;
        this.annee = annee;
}

Voiture.prototype.rouler = function() {
        console.log('Vroum vroum');
}

let maVoiture = new Voiture('Ford', 'Mustang', 1969);
let maVoiture2 = new Voiture('porche', '911', 2019);

Voiture.prototype.stop = function() {
        console.log('* crissement de pneu *');
}

maVoiture.rouler();
maVoiture2.rouler();
maVoiture.stop();
maVoiture2.stop();
```

# Classes

- 📌 Les classes sont des modèles qui vont nous permettre de créer des objets
- 📌 Les classes sont une façon plus simple de créer des objets
- 📌 C'est une syntaxe plus concise pour créer des objets (sucre syntaxique)

# Exemple

```
class Voiture {
  constructor(fabricant, modele, annee) {
        this.fabricant = fabricant;
        this.modele = modele;
        this.annee = annee;
  }
}

let maVoiture = new Voiture('Ford', 'Mustang', 1969)
```

# Diverses façons d'hériter

# Heritage par prototype

```
function Ligne(longueur){
        this.longueur = longueur;
        this.taille = function(){
                console.log('Longueur : ' + this.longueur);
        };
}

function Rectangle(longueur, largeur){
        Ligne.call(this, longueur);
        this.largeur = largeur;
        this.aire = function(){
                console.log('Aire : ' + this.longueur * this.largeur);
        };
}

let rec = new Rectangle(5, 4, 3);
rec.aire();
rec.taille();
```

# Heritage par classe

```
class Ligne{

        constructor(longueur){
                this.longueur = longueur;
        }

        taille(){
                console.log('Longueur : ' + this.longueur);
        }

}

class Rectangle extends Ligne{

        constructor(longueur, largeur){
                super (longueur);
                this.largeur = largeur;
        }

        aire(){
                console.log('Aire : ' + this.longueur * this.largeur);
        }
}

let rec = new Rectangle(5, 4, 3);
rec.aire();
rec.taille();
```

# Visibilité

- 📌 En JavaScript, il n'y a pas de mot clé pour définir la visibilité des propriétés et des méthodes
- 📌 Par convention, on utilise le préfixe `_` pour définir une propriété ou une méthode privée
- 📌 Par convention, on utilise le préfixe `$` pour définir une propriété ou une méthode protégée
- 📌 Par convention, on utilise le préfixe `__` pour définir une propriété ou une méthode statique
- 📌 Le symbole `#` est utilisé pour définir une propriété privée, elle est disponible à partir de la version 2020 de JavaScript, elle n'est pas encore supportée par tous les navigateurs

# Les setters et les getters

- 📌 En javascript il existe des mots clés pour définir des setters et des getters
- 📌 Le mot clé `get` permet de définir un getter
- 📌 Le getter est une méthode qui va être appelée quand on va accéder à une propriété
- 📌 Le mot clé `set` permet de définir un setter
- 📌 Le setter est une méthode qui va être appelée quand on va modifier une propriété

# Exemple

```
class Person {
  constructor(name) {
        this._name = name;
  }

  get name() {
        return this._name;
  }

  set name(value) {
        if (value.length > 2) {
          this._name = value;
        }
        else {
          console.log('Le nom doit contenir au moins 3 caractères');
        }
  }
}


let person = new Person('John');

console.log(person.name); // John
person.name = 'J'; // Le nom doit contenir au moins 3 caractères
console.log(person.name); // John
person.name = 'Jane';
console.log(person.name); // Jane
```

# TP

- 📌 Le but de ce TP est de créer un jeu de combat entre deux personnages
- 📌 Le jeu doit permettre de choisir entre 3 personnages différents
- 📌 Le jeu doit permettre de choisir entre differentes attaques a chaque tour
- 📌 Vous devez utiliser les classes et l'heritage pour créer les personnages et les attaques
- 📌 Vous devez utiliser les getters et les setters pour définir les points de vie des personnages
- 📌 Integrer le jeu dans une page HTML afin de pouvoir le jouer dans le navigateur