Initiation en JavaScript

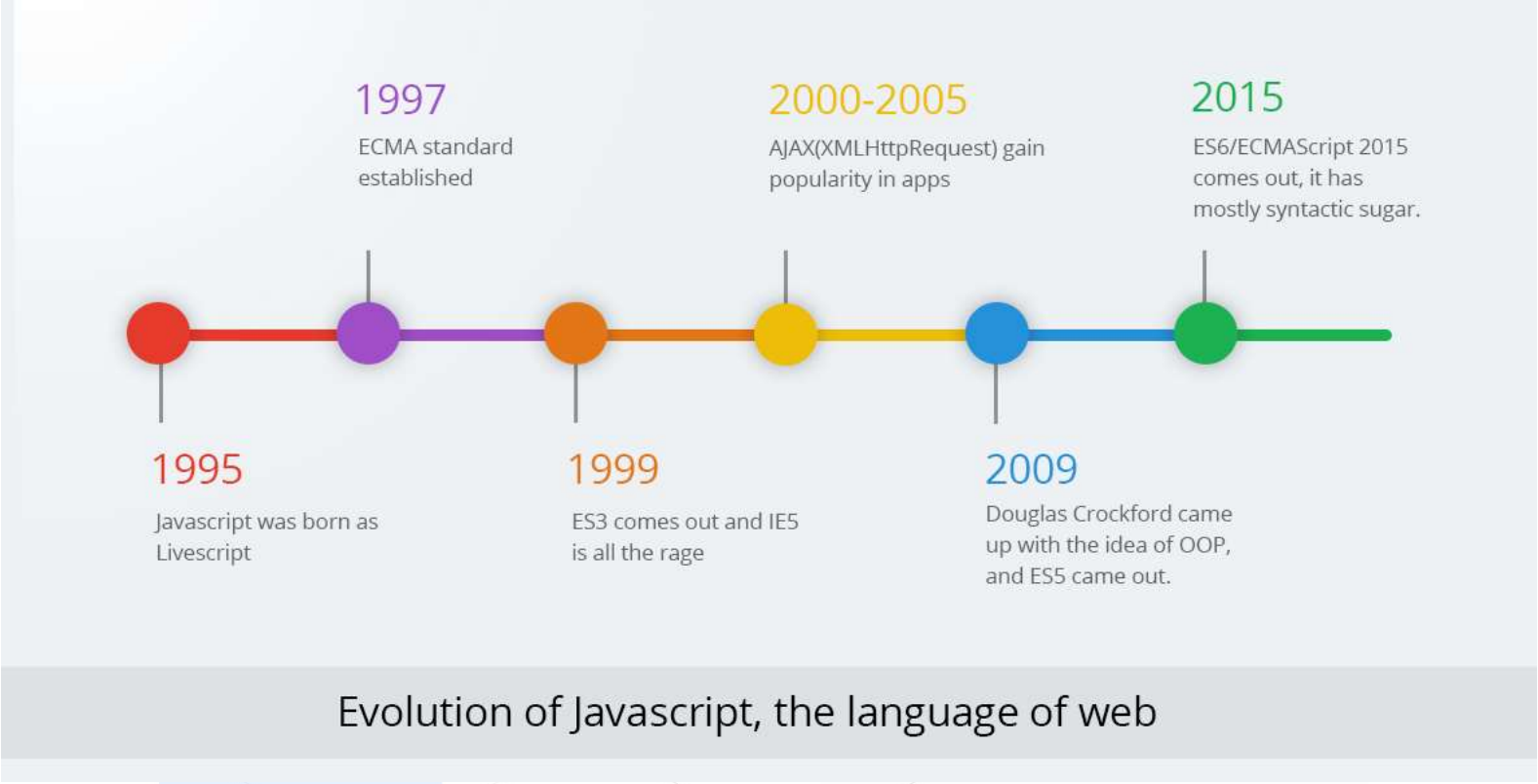
Qu'est-ce le JavaScript ?

- Langage créé en 1995
- Issu d'un langage de développement coté serveur (LiveScript)
- Un partenariat Netscape/Sun donnera son nom définitif : JavaScript
- Soumis à L'ECMA (organisme de standardisation) en 1996.
- JavasScript devient un langage à part entière 1997.
- Langage interprété :
  - ❖côté client par le navigateur
  - ❖côté serveur avec NodeJS
- Langage orienté objet.
- Concrètement :
  - JavaScript va nous permettre de manipuler une page HTML directement dans le navigateur Web de l'utilisateur.

Qu'est-ce le JavaScript ?

- Dans un Navigateur, JavaScript va nous permettre :
  - de spécifier des changements dans le document
    - après son chargement
    - au cours de sa vie dans la fenêtre du navigateur
    - sur le contenu, la structure, le style
  - en interceptant des événements (mouvements de souris, clic sur une touche de clavier, clic avec les doigts, ...)
  - d'échanger avec un serveur (AJAX)
  - de dessiner (canvas/svg)
  - de se géolocaliser
  - d'enregistrer localement du contenu (cache/bdd)
  - ...

Evolution de JavaScript



Evolution of Javascript, the language of web

JavaScript se base sur ECMAScript 6 (ES6) en tant que norme standard pour le développement web.

Préparation des Outils

- Nous allons utiliser VSCode comme éditeur de texte : https://code.visualstudio.com/
- Installez également l'extension : ES7+ React/Redux/React-Native snippets

- Les navigateurs
  - Chrome
  - Firefox
  - Edge

Digital Academy by insy2s

Bases du langage

Intégrer du code JS dans ma page

- Il y a deux manières d'ajouter du code JavaScript dans une page :
  1. En liant depuis la page HTML un fichier externe, dans lequel sont placées les instructions JavaScript :

```HTML
<script type="text/javascript" src="monscript.js"></script>
```

  2. En ajoutant le code JavaScript à l'intérieur de l'élément script :

```HTML
<script type="text/javascript">
  // Mon code Javascript
  ...
</script>
```

Intégrer du code JS dans ma page

Remarques:

- Avant, il était d'usage de placer la balise <script> entre les tags <head> et </head>, cependant il est actuellement recommandé de la placer en fin de document juste avant </body> pour ne pas bloquer le chargement de la page, et exécuter les scripts uniquement lorsque le DOM est prêt.

- Attention, la balise "script" n'est pas auto-fermante, c'est-à-dire que l'on ne peut pas écrire <script type="text/javascript" ... />

- L'attribut type doit être renseigné avec la syntaxe XHTML, mais il peut être omis en HTML5.

Instruction pour démarrer

- Afficher une popup avec un message :

```
alert("Bonjour");
```

- Ecrire du texte dans la console :

```
console.log("Bonjour");
```

- Ecrire dans le document HTML :

```
document.getElementById("myId").innerHTML = "<p>Bonjour</p>"
```

Notion de variable

- Les Variables sont utilisées pour stocker des valeurs.
- Deux étapes pour les utiliser :
  - Déclarer la variable.
  - Initialiser la variable.
- Déclarer une variable : L'instruction var (avant ES6)
- Depuis ES6 il est recommandé d'utiliser let et const,

mon_fichier.html

```
...
<script>
  var a = null;
  ...
</script>
```

Notion de variable

Le hoisting est un comportement de JavaScript dans lequel les déclarations de variables sont déplacées vers le haut de leur contexte d'exécution avant que le code ne soit réellement exécuté. Cela signifie que vous pouvez utiliser une variable même avant de les avoir déclarées explicitement dans votre code.

Cependant, le hoisting peut entraîner des comportements inattendus et des erreurs difficiles à détecter lorsqu'il est utilisé avec la déclaration de variables var.

Pour éviter ces problèmes, il est recommandé d'utiliser let et const introduits dans ECMAScript 6 (ES6). Ces déclarations offrent une portée de bloc, n'autorisent pas la redéclaration, et les variables déclarées avec const ne peuvent pas être réassignées après leur initialisation.

Initialiser une variable

Il y a 2 possibilités :

- Déclarer la variable et l'initialiser en 2 étapes :

```
// déclaration d'une variable nb
let nb;

// affectation d'une valeur pour nb
nb = 3;
```

- Déclarer la variable et l'initialiser en même temps :

```
// déclaration et affectation en même temps
let nb = 3;
```

Dans le cas d'une constante il n'est possible d'affecter la valeur que lors de la déclaration :

```
// déclaration et affectation d'une constante
const minutesPerHour = 60;
```

Les contraintes de nommages

- Les variables sont « case sensitive »
- C’est à dire :

```
let prenom = "max";

let PRENOM = "MAX";
```

- prenom et PRENOM sont deux variables différentes !
- Il est recommandé d’utiliser le plus possible la touche de « complétion » de votre éditeur (touche TAB).
- Elles doivent commencer OBLIGATOIREMENT par une lettre
- La convention de nommage des variables est le camelCase ex : numeroRue
- L'usage des accents est interdit

Les opérateurs

- Les opérateurs arithmétiques
  - \+ Addition
  - \- Soustraction
  - * Multiplication
  - / Division
  - % Modulo
  - ++ Incrément
  - -- Décrément

Les contraintes de nommages

- En JavaScript les variables sont typées
- Il y a 5 types primitifs :
  - Number : entier ou flottant.
  - String : chaine de caractères.
  - Boolean : (true ou false).
  - Undefined : quand la variable que vous souhaitez utiliser n'existe pas.
  - Null : type particulier n'ayant qu'une seule valeur : null
- typeof permet connaître le type d'une variable :

  Ex: `console.log("type de la variable prenom : " + typeof prenom);`

Les tableaux

- Un tableau est une variable qui contient une liste de valeur
- Représentation d'un tableau :

0	Une valeur
1	Une autre valeur
2	Encore une autre valeur

- Déclaration d'un tableau vide : `let tab = [];` ou `let tab = new Array();`

- Déclaration d'un tableau avec des valeurs `let tab = [1,2,3];`

Accéder aux éléments d'un tableau

- Les éléments d'un tableau sont accessibles par un index.
- Cet index est un nombre entier avec pour valeur initiale 0.
- En reprenant cet exemple : `let tab = [1,2,3];`
- On en déduit la structure du tableau et la façon d'accéder aux éléments :

0	1
1	2
2	3

- Accéder aux éléments d'un tableau

```
console.log(tab[0]);
console.log(tab[1]);
console.log(tab[2]);
```

Modifier ou Ajouter une valeur

- Modification de la valeur située à l'index 2.

```
tab[2] = "la valeur 3";
```

- Ajout d'une valeur à l'index 3.

```
tab[3] = "la valeur 4";
```

- Ajout d'une valeur à l'index 6

```
tab[6]= "la valeur 7";;
```

Méthodes pour compter, ajouter et supprimer les éléments d'un tableau

La propriété *length* d'un tableau indique le nombre d'éléments qu'il contient.

```
let people = ["Pierre", "Paul", "Jacques"];
let howManyPeople = people.length; // 3
```

Pour ajouter un élément à la fin d'un tableau, on utilise *push*.

```
people.push("Stéphanie"); // ajoute "Stéphanie" à la fin de notre tableau people
```

Pour supprimer le dernier élément d'un tableau, utilisez la méthode *pop* , sans passer aucun argument

```
people.pop(); // supprimer le dernier élément du tableau (Stéphanie)
```

Méthodes pour compter, ajouter et supprimer les éléments d'un tableau

Ajouter votre élément au début du tableau, utilisez la méthode *unshift*

```
people.unshift("Karim"); // ajoute "Karim" au début de notre tableau people
```

Pour supprimer le premier élément d'un tableau, utilisez la méthode *shift* , sans passer aucun argument

```
people.shift(); // supprime le premier élément du tableau (Karim)
```

Notion de comparaison

- Une comparaison renvoie toujours une valeur booléenne

- Elle est constituée :
  - d'un opérateur de comparaison
  - d'un opérateur logique
  - de n'importe quelle valeur pouvant être convertie en booléen

Les Opérateurs

- Logiques :
  - ! NON logique
  - && ET logique
  - || OU logique
- Comparaisons :
  - == EGAL à
  - === STRICTEMENT EGAL à (compare la valeur et le type)
  - < INFERIEUR à
  - > SUPERIEUR à
  - != DIFFERENT de

Les Structures conditionnelles : if

```
// Déclaration d'une variable 'nb' initialisée à 5
let nb = 5;

// Déclaration d'une variable 'resultat' initialisée à une chaîne vide
// Permet de stocker le résultat de la condition
let resultat = "";

// Utilisation d'une déclaration conditionnelle
// pour évaluer si 'nb' est inférieur à 5
if (nb < 5) {
  // Si 'nb' est inférieur à 5, cette condition est vraie,
  // donc ce bloc est exécuté
  resultat = "nb est plus petit que 5";
} else {
  // Si la condition précédente est fausse
  // (c'est-à-dire 'nb' n'est pas inférieur à 5), ce bloc est exécuté
  resultat = "nb est plus grand ou égal à 5";
}
// Affichage du résultat dans la console
console.log(resultat);
```

Les Structures conditionnelles if – else if

```
let nb = 1;

// Déclaration d'une variable 'resultat' initialisée à une chaîne vide
// permettant de stocker le résultat de la condition
let resultat = "";

// Utilisation de déclarations conditionnelles pour évaluer la valeur de 'nb'
if (nb == 1) {
  // Si 'nb' est égal à 1, cette condition est vraie, donc ce bloc est exécuté
  resultat = "nb est égale à 1";
} else if (nb == 2) {
  // Sinon si 'nb' est égal à 2, ce bloc est exécuté
  resultat = "nb est égale à 2";
} else if (nb == 3) {
  // Sinon si 'nb' est égal à 3, ce bloc est exécuté
  resultat = "nb est égale à 3";
} else {
  // Si aucune des conditions précédentes n'est vraie, ce bloc est exécuté
  resultat = "nb est différent de 1, 2 ou 3";
}

// Affichage du résultat dans la console
console.log(resultat);
```

Les Structures conditionnelles - Ternaire

```
let nb = 5;

let resultat = "";

if (nb < 5) {
  resultat = "nb est plus petit que 5";
} else {
  resultat = "nb est plus grand ou égal à 5";
}

console.log(resultat);
```

Le code Précédent peut être simplifié par :

```
let nb = 5;

// Utilisation de l'opérateur ternaire pour évaluer si 'nb' est inférieur à 5
// Si la condition est vraie, 'resultat' est défini comme "nb est plus petit que 5",
// sinon, il est défini comme "nb est plus grand ou égal à 5"
let resultat = (nb < 5) ? "nb est plus petit que 5" : "nb est plus grand ou égal à 5";

console.log(resultat);
```

Les Structures conditionnelles : Le switch

```
// Déclaration d'une variable 'nb' initialisée à 3
let nb = 3;

// Déclaration d'une variable 'resultat' initialisée à une chaîne vide
let resultat = "";

// Utilisation de l'instruction switch pour évaluer la valeur de 'nb'
switch (nb) {
  case 1:
    // Si 'nb' est égal à 1, exécute ce bloc
    resultat = "nb est égal à 1";
    break; // Sort du switch après avoir exécuté le bloc correspondant

  case 2:
    // Si 'nb' est égal à 2, exécute ce bloc
    resultat = "nb est égal à 2";
    break; // Sort du switch après avoir exécuté le bloc correspondant

  case 3:
    // Si 'nb' est égal à 3, exécute ce bloc
    resultat = "nb est égal à 3";
    break; // Sort du switch après avoir exécuté le bloc correspondant

  default:
    // Si 'nb' ne correspond à aucun des cas précédents, exécute ce bloc
    resultat = "nb est différent de 1, 2 ou 3";
    break; // Sort du switch après avoir exécuté le bloc correspondant
}

// La variable 'resultat' contient maintenant le résultat approprié en fonction de la valeur de 'nb'
```

Les boucles : while

```
// Déclaration d'une variable 'nb' initialisée à 0
let nb = 0;

// Utilisation d'une boucle 'while' pour incrémenter 'nb'
// jusqu'à ce qu'il atteigne 10
while (nb < 10) {
  // Tant que le 'nb' est inférieur à 10, ce bloc de code sera exécuté
  nb++; // Incrémente 'nb' de 1 à chaque itération de la boucle
}

// À la fin de la boucle, la variable 'nb' aura une valeur égale à 10
```

Les boucles : do-while

```
// Déclaration d'une variable 'nb' initialisée à 0
let nb = 0;

// Utilisation d'une boucle 'do-while' pour incrémenter 'nb' jusqu'à ce qu'il atteigne 10
do {
  // ce bloc sera exécuté au moins une fois même si 'nb' est supérieur ou égal à 10
  nb++; // Incrémente 'nb' de 1 à chaque itération de la boucle
} while (nb < 10);

// À la fin de la boucle, la variable 'nb' aura une valeur égale à 10
```

Contrairement à la boucle 'while', la boucle 'do-while' garantit que le bloc de code est exécuté au moins une fois, même si la condition est initialement fausse. Cela peut être utile dans certaines situations où vous souhaitez vous assurer qu'un bloc de code est exécuté au moins une fois, indépendamment de la condition initiale. Dans votre exemple, la variable nb atteindra une valeur égale à 10 à la fin de la boucle.

Les boucles : for

```
// Utilisation d'une boucle 'for' pour itérer de 0 à 9
for (let i = 0; i < 10; i++) {
  // Affiche la valeur actuelle de 'i' dans la console à chaque itération
  console.log(i);
}
```

La boucle 'for' est souvent utilisée lorsque vous connaissez à l'avance le nombre d'itérations que vous souhaitez effectuer. Dans cet exemple, la boucle itère de 0 à 9, et la variable i prend les valeurs successives de 0 à 9, qui sont ensuite affichées dans la console à chaque itération.

Les boucles : for…in

```
// Déclaration d'un tableau de chaînes représentant des marques de voitures
const marquesVoitures = ["Toyota", "Ford", "Honda", "BMW", "Mercedes-Benz"];

// Utilisation de la boucle 'for...in' pour itérer sur les indices du tableau
for (const index in marquesVoitures) {
  // Affiche l'indice et la marque de voiture correspondante à chaque itération
  console.log(`Indice : ${index}, Marque de voiture : ${marquesVoitures[index]}`);
}
```

'for…in' est une façon plus synthétique de faire une boucle 'for'.
Cependant La boucle 'for...in' est plus adaptée aux objets et est conçue pour itérer sur les propriétés énumérables. Elle n'est généralement pas recommandée pour les tableaux, car elle pourrait inclure des propriétés indésirables.

Les boucles : for…of

```
// Déclaration d'un tableau de chaînes représentant des marques de voitures
const marquesVoitures = ["Toyota", "Ford", "Honda", "BMW", "Mercedes-Benz"];

// Utilisation de la boucle 'for...of' pour itérer sur les éléments du tableau
for (const marque of marquesVoitures) {
  // Affiche chaque marque de voiture à chaque itération
  console.log(`Marque de voiture : ${marque}`);
}
```

L'utilisation de 'for...of' est plus appropriée pour itérer sur les valeurs d'un tableau, car elle évite la nécessité de manipuler les indices et offre une syntaxe plus concise que 'for' et 'for…in' .

Les Fonctions

Les fonctions en JavaScript sont des éléments fondamentaux pour structurer et organiser le code. Elles permettent la réutilisation du code, la modularité et contribuent à rendre le code plus lisible et maintenable.

Déclaration et Appel d'une fonction :

```
// Déclaration de la fonction afficherMessage
function afficherMessage(nom) {
  // Le paramètre 'nom' représente le nom de la personne à saluer
  // Il s'agit d'une variable locale à la fonction

  // Corps de la fonction : affiche un message de bienvenue dans la console
  console.log(`Bienvenue, ${nom} !`);
}

// Appel de la fonction avec un exemple d'argument concret
afficherMessage("Alice");
```

Les Fonctions avec une valeur de retour

Déclaration et Appel d'une fonction avec une valeur de retour:

```
// Déclaration de la fonction calculerMoyenne avec deux paramètres
function calculerMoyenne(nombre1, nombre2) {
  // Le paramètre 'nombre1' représente le premier nombre à prendre en compte
  // Le paramètre 'nombre2' représente le deuxième nombre à prendre en compte

  // Corps de la fonction : calcul de la moyenne des deux nombres
  let moyenne = (nombre1 + nombre2) / 2;

  // Renvoie la moyenne calculée
  return moyenne;
}

// Appel de la fonction avec des exemples d'arguments concrets
let resultatMoyenne = calculerMoyenne(10, 20);

// Affichage du résultat
console.log(resultatMoyenne); // Affiche 15
```

Les Fonctions fléchées

Les fonctions fléchées, introduites avec ECMAScript 6 (ES6), offrent une syntaxe plus concise pour déclarer des fonctions en JavaScript. Elles sont particulièrement utiles pour les fonctions anonymes et pour réduire le code boilerplate.

Déclaration et Appel d'une fonction:

```
// Fonction traditionnelle
function multiplierTraditionnelle(a, b) {
  return a * b;
}

// Appel de la fonction
let resultatTraditionnelle = multiplierTraditionnelle(5, 3);
console.log(resultatTraditionnelle); // Affiche 15


// Fonction fléchée
const multiplierFleche = (a, b) => a * b;

// Appel de la fonction
let resultatFleche = multiplierFleche(5, 3);
console.log(resultatFleche); // Affiche 15
```

Quelques Fonctions du langage

Fonctions de conversion à partir d'une chaine de caractère :

- **parseInt()** : renvoie un entier à partir d'une chaine de caractères
- **parseFloat()** : renvoie un flottant à partir d'une chaine de caractères

Fonctions de test :

- **isNaN()** : teste si le paramètre n'est pas un nombre
- **isFinite()** : teste si le paramètre n'est pas infini

Quelques Fonctions du langage

Méthode 'forEach':

La méthode 'forEach' exécute une fonction pour chaque élément du tableau. Elle est utilisée pour effectuer des actions sans créer un nouveau tableau.

```
const fruits = ['pomme', 'banane', 'orange'];

fruits.forEach(fruit => console.log(fruit));
// Affiche :
// pomme
// banane
// orange
```

Quelques Fonctions du langage

Méthode 'map':

La méthode 'map' crée un nouveau tableau en appliquant une fonction à chaque élément du tableau d'origine. Elle ne modifie pas le tableau initial.

```
const nombres = [1, 2, 3, 4];

const carres = nombres.map(nombre => nombre * nombre);

console.log(carres); // Affiche [1, 4, 9, 16]
```

Quelques Fonctions du langage

Méthode 'find':

La méthode 'find' renvoie la première valeur qui satisfait une condition spécifique dans le tableau. Elle s'arrête dès qu'un élément correspondant est trouvé.

```
const personnes = [
  { nom: 'Alice', age: 25 },
  { nom: 'Bob', age: 30 },
  { nom: 'Charlie', age: 22 }
];

const personneTrouvee = personnes.find(personne => personne.age === 30);

console.log(personneTrouvee); // Affiche { nom: 'Bob', age: 30 }
```

Quelques Fonctions du langage

Méthode 'filter':

La méthode 'filter' crée un nouveau tableau contenant uniquement les éléments qui satisfont une condition spécifique.

```
const nombres = [10, 5, 8, 15, 3];

const nombresPairs = nombres.filter(nombre => nombre % 2 === 0);

console.log(nombresPairs); // Affiche [10, 8]
```

Exceptions

Les exceptions en JavaScript sont des événements qui se produisent pendant l'exécution d'un programme et qui interrompent le flux normal d'instructions. La gestion des exceptions est cruciale pour écrire un code robuste et prévenir les erreurs imprévues.

Une exception est une anomalie qui se produit pendant l'exécution d'un programme. Cela peut être dû à des erreurs de logique, des entrées incorrectes, des problèmes réseau, ou d'autres circonstances imprévues.

En JavaScript, les exceptions sont gérées à l'aide de blocs 'try...catch'. Le code potentiellement problématique est placé dans le bloc try, et les erreurs éventuelles sont capturées et gérées dans le bloc 'catch'.

Exceptions : try...catch

La structure try...catch en JavaScript est utilisée pour gérer les exceptions et les erreurs qui peuvent survenir pendant l'exécution d'un programme. Elle permet de définir un bloc de code à tester (le bloc 'try') et de spécifier comment gérer les erreurs éventuelles (le bloc 'catch'). Voici comment elle fonctionne :

```
try {
  // Bloc try : code potentiellement problématique
  // Si une exception se produit ici, elle est "attrapée" et
  // gérée par le bloc catch
} catch (erreur) {
  // Bloc catch : code pour gérer l'exception
  // L'objet erreur contient des informations sur l'exception
} finally {
  // Bloc finally : code qui s'exécute toujours,
  // que l'exception soit survenue ou non
}
```

Exceptions : throw

L'instruction throw en JavaScript est utilisée pour générer manuellement une exception. Elle peut être utilisée dans n'importe quelle partie de votre code où vous souhaitez signaler une condition d'erreur ou générer une exception spécifique.

```
try {
  let age = prompt('Entrez votre âge :');

  if (isNaN(age)) {
    throw new Error('L\'âge doit être un nombre.');
  }

  console.log('Votre âge est :', age);
} catch (erreur) {
  console.error('Une erreur s\'est produite :', erreur.message);
}
```

Les Objets : tableaux Associatifs

Un objet représente un concept, une idée ou toute entité du monde réel, comme une voiture, une personne ou encore une page d'un livre. La structure ressemble au Tableaux.

Rappel des tableaux :

```
let tab = [1,2,3];
```

0	1
1	2
2	3

Il s'agit d'un tableau indexé
(les clés sont des entiers)

Objet en JS correspondant à du JSON :

```
let personne = {
  nom : "DUPONT",
  prenom : "Jeanne"
};
```

Key	Value
nom	DUPONT
prenom	Jeanne

Il s'agit d'un tableau associatif
(les clés sont des chaines de caractères)

Les Objets : Notions de POO

La programmation orientée objet (POO) en JavaScript repose sur la création et la manipulation d'objets. Les objets peuvent être créés à partir de classes en utilisant le concept de constructeurs et d'instances.

```
// Définition de la classe Personne
class Personne {
  // Constructeur
  constructor(nom, age) {
    this.nom = nom;
    this.age = age;
  }

  // Méthode
  afficherDetails() {
    console.log(`Nom: ${this.nom}, Age: ${this.age}`);
  }
}

// Création d'instances de la classe Personne
const personne1 = new Personne('Alice', 25);
const personne2 = new Personne('Bob', 30);

// Appel de la méthode sur les instances
personne1.afficherDetails(); // Affiche "Nom: Alice, Age: 25"
personne2.afficherDetails(); // Affiche "Nom: Bob, Age: 30"
```

Les Objets : Héritage

La programmation orientée objet (POO) en JavaScript repose sur la création et la manipulation d'objets. Les objets peuvent être créés à partir de classes en utilisant le concept de constructeurs et d'instances.

```
// Définition de la classe Etudiant qui hérite de Personne
class Etudiant extends Personne {
  constructor(nom, age, niveau) {
    // Appel du constructeur de la classe parente
    super(nom, age);
    this.niveau = niveau;
  }

  // Méthode spécifique à la classe Etudiant
  afficherDetailsEtudiant() {
    console.log(`Niveau: ${this.niveau}`);
  }
}

// Création d'une instance de la classe Etudiant
const etudiant1 = new Etudiant('Charlie', 22, 'Première année');

// Appel des méthodes héritées de la classe Personne et spécifiques à la classe Etudiant
etudiant1.afficherDetails(); // Affiche "Nom: Charlie, Age: 22"
etudiant1.afficherDetailsEtudiant(); // Affiche "Niveau: Première année"
```