# Chapitre 1 : Intro

## Que permet le Javascript ?

* Il permet d’écrire des scripts clients
  + **Script** : Programme destiné à être exécuté
  + **Client** : Tout se déroule sur l’ordinateur de l’internaute

## Les caractéristiques de JS

|  |  |
| --- | --- |
| 1. Un langage de programmation | 1. Impératif |
| 1. Orienté objet | 1. Evènementiel |
| 1. Faiblement / non typé   ***Dynamiquement Typé*** | 1. Où les fonctions sont des objets de premier ordre |
| 1. Qui évolue (ES6) | 1. Sensible à la casse |

*Cf : p17-22 du cours*

## Les commandes de bases

* confirm(txt)
  + Demande une confirmation et renvoi True ou False dépendant de si l’utilisateur a utilisé Enter ou Cancel
* prompt(txt [, valDefaut])
  + Demande une entré (en STRING) à l’utilisateur.
  + Peut éventuellement renvoyer une valeur par défaut
* document.write(txt)
  + Ecrit directement dans le document HTML
* console.log(txt)
  + Ecrit dans la console du navigateur
* alert(txt)
  + Ecrit dans une fenêtre pop-up

## Les manières d’insérer du script

### 1.4.1 : Code interne

* Dans des balises script
  + Il peut y avoir plusieurs balises
  + Elles peuvent se trouver n’importe où
  + <script>...</script>

### 1.4.2 : Code inline

* <button onclick= "codeJS">Cliquez moi !</button>
* <a href= "javascript :*codeJS*">text</a>

### 1.4.3 : code externe

* <script src= "nom\_fichier\_js"></script>
* Exécution synchrone (immédiate)

# Chapitre 2 : Variables et valeurs

## Les types primitifs

|  |  |
| --- | --- |
| Number | String |
| Boolean | Function |
| Object | Undefined |

### 2.1.1 : Les nombres

* Valeurs acceptées :

|  |  |  |
| --- | --- | --- |
| Entiers | 42  -17 | |
| Réels | 342.17 | |
| Autres bases | 037 (octal)  0x3BFF (hexa)  0b011101 (binaire) | |
| Valeurs spéciales | Infinity | 5 / 0 |
| -Infinity |  |
| Nan | 100 / "pas un chiffre" |

* Opérations sur les nombres :

|  |  |  |
| --- | --- | --- |
| Opérateurs usuels | + - \* / % \*\*(exposant) | |
| Opérateurs unaires | * ++ -- | |
| Opérations de comparaison | Plus petit [ou égal] | < <= |
| Plus grand [ou égal] | > >= |
| Est égal | == |
| Est strictement égal  (vérifie le type, ne fait aucune conversion implicite) | === |
| Opérateurs bit à bit | & | ~ ^ << >> >>> | |

### 2.1.2 : Les booléens

* Littéraux pour les booléens :
  + true / false
* Opérations sur les booléens
  + ! && ||
  + Ternaire 🡪 ? :

### 2.1.3 : Les strings

* Encadré par des guillemets ou des apostrophes
  + "pomme", 'Clément'
* Caractères échappés :
  + \n \t \' \" \\
* Les Gabarits / template literals:
  + Encadré par des backtips (`...`)
  + Peut être réparti sur plusieurs lignes
  + Permet d'inclure des expressions régulière
    - ${*nom\_variables*}
* Operations standards

|  |  |
| --- | --- |
| Concaténation  + | "Bonjour" + *nom*   * "Bonjour Clément" |
| Longueur | Nom.length |
| Extraction | Nom[0]  Nom.charAt(0) |

* Quelques principes :
  + Les chaines de caractères sont immuables
    - On ne peut changer juste une valeur du tableau
    - On ne peut pas non plus changer sa valeur *length*
* Opérations prédéfinies

|  |  |  |
| --- | --- | --- |
| Test | s.startsWith(début) | Renvoie vrai ou faux |
| s.endsWith(fin) |
| s.includes(partie) |
| Extraction | s.substr(début, longueur) | Renvoie une nouvelle chaine de caractère |
| s.substring(début, fin) |
| Recherche | s.indexOf(partie) | Renvoie l'index correspondant. -1 si introuvé |
| s.lastIndexOf(partie) |
| Décomposition | s.split(separateur) | Renvoie un tableau décomposant les éléments |

* Javascript est typé dynamiquement
  + Le contenu d’une variable peut changer de type au cours de l’éxécution
  + Même si à chaque moment, une variable à un type précis
  + Ce qui engendre :
    - Aucun type à la déclaration
    - De nombreuses conversions implicites sont effectuées
    - Certaines erreurs logique ne sont pas facilement détectable
* Donner le type d’une variable à un moment précis :
  + typeof ...
    - typeof 12 🡪 "number"
* Les réponses que peuvent donner typeof :
  + number
    - 12
    - parseint("21")
    - NaN
    - -78
    - 342.17
    - 037 (octal) 0x3BFF(hexa) 0110(binaire)
    - Infinity (5/0)
    - -Infinity
  + string
    - "coucou"
  + object
    - new Date()
    - null
  + function
    - function test(){}

## Les variables

### 2.2.1 : Le scope

* Par définition, un scope représente une portion de code (un bloc) qui est comprise entre 2 accolades.
* Exemples :
  + une fonction
  + Une condition
  + une boucle
  + ...
* Une variable n'est pas accessible en dehors de son scope

### 2.2.2 : Les 4 déclarations de variables

1. Pas de déclaration
   1. msg = "hello";
   2. Déclaré au niveau global
2. let
   1. let msg = "hello";
   2. Déclaré localement suivant les règles du scope
3. const
   1. const msg = "hello";
   2. Déclaré localement suivant les règles du scope
   3. Doit obligatoirement être initialisé
   4. Ne peut être modifié par la suite
4. var
   1. var msg = "hello"
   2. Déclaré au niveau de la fonction englobante du scope
   3. Peut être redéclaré.

## Le hoisting

* Dans chaque scope fonctionnel :
  + les délcarations var et function sont hissées vers le début.
  + Pour let et const :
    - Hissage vers le début du bloc
    - Mais on ne peut utiliser la variable avant sa déclaration
      * On parle alors de TDE (Temporal Dead Zone)
  + /!\ Cependant, seules les déclarations sont hissés /!\

## Les conversions explicites

* Tableau des conversions :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A transformer  En ça | Nombre | Booléen | String | Undefined | Object |
| Nombre |  | true / false  1 / 0 | "42" 🡪 42  Vide ou blancs 🡪 0  "mot" 🡪 NaN | NaN | null 🡪 0 |
| Booléen | 0, NaN / autre  False / true |  | Vide "" / "autre…"  false / true | false | null / autre  false / true |
| String | NaN 🡪 "NaN"  Infinity 🡪 "Infinity"  Autre 🡪 "chiffre" | "true", "false" |  | "undefined" | "null" ou via toString() |

* A RETENIR CONCERNANT LE TABLEAU :
  + Les valeurs falsy :
    - false, 0, null, undefined, "", NaN
  + 2 autres méthodes pour convertir en nombre :
    - parseFloat(s)
    - parseInt(s, base)
    - Dans les 2 cas, on s'arrête au premier caractère illégal

## Les conversions implicites

|  |  |
| --- | --- |
| Expression | Priorité |
| + | S'il y a un string : concaténation  Sinon : addition |
| == | Si valeurs de même types :  Comparaison simple. Sauf NaN qui n'est pas égale à lui-même  S'il y a *null* ou *undefined* : alors  null == undefined 🡪 true  sinon 🡪 false  Sinon, conversion en nombre |
| === | Aucune conversion. Donne automatiquement faux si les types sont différents ! |
| < | Si 2 strings, alors c'est l'ordre lexicographique  Sinon, conversion en nombre et comparaison |