# Javascript

## Valeurs et variables

# Définition

Une variable est un symbole qui associe un nom (l'identifiant) à une valeur.

La valeur peut être une valeur primitive ou un objet.

# Valeurs primitives

- booléen
- nombre
- chaîne de caractère
- undefined
- null
- Symbol

# Booléen

`true` ou `false`

# Nombre

Nombres à virgule flottante sur 64 bits.
Deux valeurs spéciales : `NaN` (not a number) et `Infinity`.

```
3/0; //Infinity

parseFloat("a"); // NaN
```

# Chaîne de caractère

Délimitée indiféremment par des apostrophes (simple quote) ou des guillemets (double quote).

```
"toto";
'toto';
```

Délimitée par des guillements obliques, la chaîne peut être écrite sur plusieurs lignes et intégrer des expressions.

```
let nom = "Toto";

alert(`Bonjour ${nom},
comment vas-tu ?`);
```

# Chaîne de caractère

## Caractères spéciaux

| | |
|---|---|
| `\b` | Retour arrière |
| `\f` | Saut de page |
| `\n` | Nouvelle ligne |
| `\r` | Retour chariot |
| `\t` | Tabulation |
| `\v` | Tabulation verticale |
| `\'` | Apostrophe ou guillemet droit simple |
| `\"` | Guillemet droit double |
| `\\` | Barre oblique inversée |
| `\`*XXX* | Caractère Latin-1 spécifié par, au plus, 3 chiffres octaux entre 0 et 377. |
| `\x`*XX* | Caractère Latin-1 spécifié par 2 chiffres hexadécimaux entre 00 et FF. |
| `\u`*XXXX* | Caractère Unicode spécifié par 4 chiffres hexadécimaux. |

# Undefined

Valeur utilisée lorsqu'aucune valeur n'a été attribuée à une variable.

```
let toto;

toto; //undefined
```

# null

Valeur qui représente l'absence intentionnelle de valeur.

```
let toto = null;
```

# Symbol

Valeur unique et immuable qui peut être utilisée comme clé d'une propriété d'un objet.

```
let sym = Symbol("toto");
let obj = {};
obj[sym] = "titi";
```

=> Utilisation avancée de javascript.

# Déclaration

## Obligatoire en mode strict

```
"use strict";

let tata; //déclaration
let toto = "titi"; //déclaration + affectation
let titi, tutu; //déclaration de 2 variables avec un seul mot clé let.

toto; // "titi"
titi; // undefined
tutu; // undefined
tyty; // ReferenceError: tyty is not defined
```

Les variables se déclarent avec les mots clés let

# Typage dynamique

On ne spécifie pas le type de données.

```
let reponse = 42;

reponse = "toto";
```

# Identifiants

## Règles

- commencent par une lettre, un tiret bas (_) ou un dollar ($)
- les caractères qui suivent peuvent être des chiffres (`0` à `9`)
- sensibles à la casse
- ne sont pas des mots réservés

## Habitudes

- camelCase : `maVariable` plutôt que `ma_variable`
- les constructeurs (et uniquement eux) commencent par une majuscule

```
let maVariable = new MonConstructeur();
```

# Mots réservés

- `break`
- `case`
- `catch`
- `class`
- `const`
- `continue`
- `debugger`
- `default`
- `delete`
- `do`
- `else`
- `enum`*
- `export`
- `extends`
- `finally`
- `for`
- `function`
- `if`
- `implements`*
- `import`
- `in`
- `instanceof`
- `interface`*
- `let`
- `new`
- `package`*
- `private`*
- `protected`*
- `public`*
- `return`
- `static`
- `super`
- `switch`
- `this`
- `throw`
- `try`
- `typeof`
- `var`
- `void`
- `while`
- `with`
- `yield`

* réservés en vue de l'évolution de javascript.

# Portée des variables

## Variables globales

En dehors des fonctions, on parle d'espace global.
Les variables définies dans cet espace sont appelées variables globales. Elles sont disponibles pour tout le code contenu dans le document.

## Variables locales

Lorsqu'une variable est déclarée dans une fonction, elle est appelée variable locale car elle n'est disponible qu'au sein de cette fonction.

# Variables globales

## Dans un navigateur web

Différents fichiers javascript partagent le même espace global.

```
<!doctype html>
  <html>
     <head></head>
     <body>
        <script src="script1.js"></script>
        <script src="script2.js"></script>
        <script>
           alert(maVariable); //tata
        </script>
     </body>
  </html>
```

script1.js

```
let maVariable = "toto";
```

script2.js

```
let maVariable = "tata";
```

# Portée des variables

## Avant ES6

```
var x = 3;

if (true) {
  var x = 2;
  x; //2
}

x; //2, pas d'erreur générée
```

La portée de bloc n'existait pas.

L'écrasement des variables est possible.

# Portée des variables

## Aujourd'hui

```
let x = 3;

if (true) {
  let x = 2;
  x; //2
}

x; //3
```

Le mot clé `let` permet la portée de bloc.

# Constantes

```
const x = 2;

x = 3; //TypeError: invalid assignment to const `x'
```

# Constantes

Les constantes ont une portée de bloc.

```
const x = 2;

if (true) {
  const x = 3;
}

x; //2
```

# Constantes

Si la valeur est un objet, celui-ci peut être modifié, c'est la variable qui ne peut pas être réaffectée.

```
const obj = { nom : "Toto", age : 26 };

obj.age = 27; //ok

obj = "toto"; //TypeError: invalid assignment to const 'obj'
```

# Conversions implicites

Si une instruction attend un type autre que celui de l'opérande, ce dernier est automatiquement converti. Attention aux pièges !

```
"La réponse est " + 37; // "La réponse est 37"
"37" - 7; // 30
"37" + 7; // "377"
```