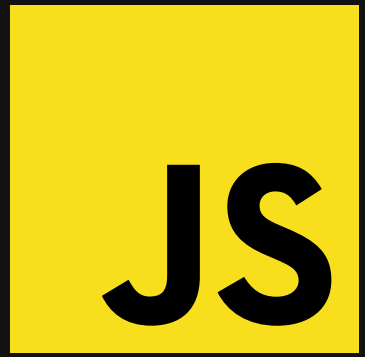
JS

JavaScript

Préambule

- Le JavaScript est créé en 1995
- Standardisé sous le nom d'*ECMAScript*
- Depuis 2015, une nouvelle version sort chaque année
- Parmi les languages les plus utilisés au monde
- Le web utilise ce language pour scripter ses pages, tous les navigateurs savent interpréter du JavaScript

Il existe un language de programmation nommé Java, qui n'a absolument rien à voir avec JavaScript. Il ne faut pas les confondre

JavaScript

Les ressources

W3Schools : https://www.w3schools.com/js/
Références et tutoriels facile d'accès. Excellent pour débuter

MDN : https://developer.mozilla.org/en-US/docs/Web/JavaScript
Documentation officielle de Mozilla. Référence la plus complète et la plus précise disponible

De nombreux tutoriels et documentations sont également disponibles sur presque tous les supports (livres, sites internet, vidéos, etc.)

N'hésitez pas à chercher dès que vous avez une questions. *Savoir utiliser une documentation doit faire partie de vos compétences*

JavaScript

Où l'écrire

Le javascript s'écrit toujours dans une balise `<script>`. Il peut s'écrire :

A même le HTML

```html
<!DOCTYPE html>
<html>
    <head>
        <title>JavaScript</title>
    </head>
    <body>
        <script>
            console.log('Hello World!');
        </script>
    </body>
</html>
```

Dans un fichier externe

```html
<!DOCTYPE html>
<html>
    <head>
        <title>JavaScript</title>
    </head>
    <body>
        <script src="script.js"></script>
    </body>
</html>
```

Ces deux codes sont équivalents si `script.js` contient `console.log('Hello World!');`

JavaScript

Ordre d'execution

Le JavaScript s'execute dans l'ordre où la page est lue par le navigateur. Cela implique qu'un script ne peut accéder qu'aux éléments déjà analysés.

Par défaut, l'execution du JavaScript est bloquante pour le chargement de la page. On placera généralement la balise `<script>` à la fin de la balise `<body>` pour ne pas ralentir le chargement.

```
<!DOCTYPE html>
<html>
    <head>
        <title>JavaScript</title>
        <script> ... </script>      -> S'exécutera avant le chargement du body, il ne pourra pas y accéder
    </head>
    <body>
        <script> ... </script>      -> S'exécutera après le chargement du body et avant celui du h1
        <h1> Titre </h1>
        <script> ... </script>      -> S'exécutera une fois tous les éléments chargés
    </body>
</html>
```

JavaScript

Les variables

JavaScript utilise un *typage implicite*. Comme en Python, on ne déclare pas le type des variables. Les variables sont déclarées avec `let`, ou avec `const` si leur valeur ne change pas.

```
let maVariable = 3;    // Déclare une variable, sa valeur pourra changer
const maVariable = 3;  // Déclare une constante, changer sa valeur provoquera une erreur
```

Selon la norme ECMAScript, toutes les instructions en JavaScript doivent se terminer par un point-virgule, mais l'interpréteur acceptera si vous ne les mettez pas. Prenez l'habitude de respecter le standard et de toujours les utiliser.

Il est possible de déclarer des variable avec `var` ou sans utiliser de préfixe. Cela déclare une variable globale au contexte courant. On ne l'utilise plus en pratique, cela créé des confusions.

```
maVariable = 3;       // Ces deux lignes font la même chose. maVariable sera globale à la fonction courante
var maVariable = 3;   // Ces syntaxes sont désuètes, à ne pas utiliser !
```

JavaScript

Les types

Il existe 7 types de base en JavaScript :

```
const maVariable = "JavaScript";    // Chaîne de caractères
const maVariable = 3.14;            // Nombre réel
const maVariable = true;            // Booléen
const maVariable = null;            // null, indique l'absence d'un objet
const maVariable;                   // undefined, indique qu'il n'y a pas de valeur, "non déclaré"
const maVariable = BigInt(3);       // BigInt, entier sans limites
const maVariable = Symbol("foo");   // Symbol, pas utile dans ce cours
```

L'instruction `typeof` retourne le type de la variable sous forme de chaîne de caractères

```
const maVariable = false;
const typeDeMaVariable = typeof maVariable; // typeDeMaVariable vaut "boolean"
```

Il existe également d'autres types non basiques, qui sont des structures stockant des types basiques (tableaux, objets, etc).

JavaScript

La console

Pour afficher un texte dans la console :

```
console.log("Bonjour"); // Affichera "Bonjour" dans la console
```

La fonction print() existe en JavaScript, mais il s'agit de la commande pour imprimer la page web !

Il existe les fonctions pour afficher des messages d'avertissement et d'erreur :

```
console.warn("Attention");
console.error("Erreur");
```

Ces fonctions peuvent afficher n'importe quelle variable de n'importe quel type, profitez-en !

JavaScript

Les opérations booléennes

Les comparaisons classiques sont utilisées :

```
console.log(3 < 2);    // false
console.log(3 >= 2);    // true
console.log(3 == 2);   // false
console.log(3 != 2);   // false
```

Les opérateurs `===` et `!==` permettent de tester la valeur et le type :

```
console.log(2 == "2");    // true      car les valeurs sont identiques, peu importe le type
console.log(2 === "2");   // false     car les types sont différents
console.log(2 != "2");    // false
console.log(2 !== "2");   // true
```

Les opérateurs `&&`, `||` et `!` fonctionnent respectivement comme `and`, `or` et `not` en Python

```
console.log( !(2 != "2" || 3 < 2) && 0 < 4); // true
```

JavaScript

Les opérations mathématiques

En JavaScript on retrouve les opérations des base : addition `+`, soustraction `-`, multiplication `*`, division `/`, modulo `%` (reste d'une division entière).

Il existe également les opérateurs d'assignement : `+=`, `-=`, `*=`, `/=`
`x += y` est équivalent à `x = x + y`, idem pour les autres opérateurs

Il existe aussi les opérateurs d'incrémentation `++` et de décrémentation `--`
`x++` est équivalent à `x = x + 1` et `x--` est équivalent à `x = x - 1`

L'opérateur `+` sert de concaténation pour les chaînes de caractères.
`"Hello " + "World"` donne `"Hello World"`

Pour tous les détails, voir https://www.w3schools.com/js/js_operators.asp

JavaScript

Les fonctions mathématiques

Toutes les fonctions de math avancées se font grâce à l'objet `Math`

- `Math.abs(x)` : valeur absolue
- `Math.log(x)`, `Math.exp(x)` : logarithme et exponentielle
- `Math.min(x, y, z, ...)`, `Math.max(x, y, z, ...)` : minimum et maximum de plusieurs valeurs
- `Math.random()` : nombre aléatoire dans l'intervalle `[0, 1[`
- `Math.sqrt(x)`, `Math.pow(x, y)` : racine carrée et puissance
- `Math.round(x)`, `Math.floor(x)`, `Math.ceil(x)` : Arrondi resp. au plus proche, inférieur, supérieur
- `Math.PI` : π

Référence complète : https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/Math

JavaScript

Coercition

Les opérateurs mathématiques peuvent être utilisés entre tout types, mais attention aux comportements surprenants ! L'addition est particulière.

```
console.log(3 + 5);       // 8        Addition standard
console.log("3" + "5");   // "35"     Concaténation de texte
console.log(3 + "5");     // "35"     Dès qu'un opérateur est un texte, c'est une concaténation
console.log(3 - 5);       // -2       Soustraction standard
console.log("3" - "5");   // -2     La soustraction converti automatique tout texte en nombre
console.log("3" - "A");   // NaN      Si un texte n'est pas convertible, la valeur NaN (Not A Number est retournée)
console.log("3" * "5");   // 15       Idem pour tous les autres opérateurs
console.log("3" * true);  // 3        true est converti en la valeur 1, false est converti en la valeur 0
console.log("3" - null);  // 3        null est converti en la valeur 0
console.log("3" - []);    // 3        Les tableau vides valent 0
console.log("3" - undefined); // NaN  Une opération avec "undefined" provoquera toujours un Nan
```

Il existe beaucoup d'autres cas. On apelle cette "conversion automatique" entre types la *coercition*.

Retenez : Éviter au maximum les opérations entres types différents !

JavaScript

Cas particuliers

Pas convaincu de ne pas mélanger les types ?

```
console.log(('b' + 'a' + + 'lancer' + 'a').toLowerCase()); // Qu'affiche ceci ?
console.log(+ 'lancer');                                   // NaN      car le + converti le 'lancer' en nombre
console.log('a' + (+'lancer'));                            // aNaN     car un + avec du texte converti en texte
console.log('a' + + 'lancer');                             // aNaN     car JavaScript suit la priorité des opérations
console.log('a' + + 'lancer' + 'a');                       // aNaNa
console.log(('b' + 'a' + + 'lancer' + 'a'));               // baNaNa
console.log(('b' + 'a' + + 'lancer' + 'a').toLowerCase()); // banana   toLowerCase met tout en minuscules
```

Liste d'opérations WTF en JavaScript: https://github.com/denysdovhan/wtfjs

JavaScript

Les conditions

Les conditions s'écrivent de la manière suivante :

```
if(a < b){
    console.log("Do A");
}else if(a < c){
    console.log("Do B");
}else{
    console.log("Do C");
}
```

Comme en Python, les blocs `else if` et `else` sont facultatifs

Il est aussi possible de faire des ternaires avec la syntaxe `<condition> ? <si_vrai> : <si_faux>`

```
const a = 5;
const value = a > 3 ? "Plus grand" : "Plus petit ou égal";
console.log(value); // "Plus grand"
```

JavaScript

Les boucles – la boucle while

Pour répéter des instructions plusieurs fois, il existe les boucles :

```
let text = "";
while(text != "Bonjour"){
    text = prompt("Dites 'Bonjour' !");  // prompt() Demande à l'utilisateur d'entrer un texte
}
console.log("Vous avez dit bonjour !");
```

La boucle `while (condition) {instructions}` répète les `instructions` tant que `condition` est vraie.

La boucle `while` est utile quand *on ne connait pas le nombre de répétitions à faire*.

Attention à *toujours* vous assurer que la boucle se terminera, sinon c'est une boucle infinie

L'évaluation de la condition est faite au début de chaque itération

JavaScript

Les boucles – la boucle for

Pour répéter des instructions plusieurs fois, il existe les boucles :

```
for(let i=0; i<10; i++){
    console.log(i); // 0 1 2 3 4 5 6 7 8 9
}
```

La boucle `for (initialisateur; condition; iteration) {instructions}` :

- Execute `initialisateur` avant d'entrer dans la boucle
- Execute `iteration` après chaque tour de boucle
- S'execute tant que `condition` est vraie

La boucle `for` est utile quand *on connait le nombre de répétitions à faire*. Elle permet d'éviter des erreurs qui créent des bouclent infinies, car il est plus simple de voir que la boucle s'arrêter quoi qu'il arrive.

JavaScript

Les boucles – break & continue

Dans une boucle, l'instruction `break` sort immédiatement de la boucle, sans condition.

```
const a = [/*Quelque chose*/];
for(let i=0; i<a.length; i++){
    if (a[i] == 0) {            // Si a[i] vaut zéro, sort de la boucle
        break;
    }
    console.log(`${1/a[i]}`);   // Ceci affichera l'inverse de tous les éléments du tableau, jusqu'au premier zéro
}
```

Dans une boucle, l'instruction `continue` passe immédiatement à l'itération suivante.

```
const a = [/*Quelque chose*/];
for(let i=0; i<a.length; i++){
    if (a[i] == 0) {            // Si a[i] vaut zéro, ignore l'itération actuelle et continue la boucle
        continue;
    }
    console.log(`${1/a[i]}`);   // Ceci affichera l'inverse de tous les éléments du tableau, sauf ceux valant zéro
}
```

JavaScript

Les tableaux

Les tableaux sont l'équivalent des listes en Python

```
const a = [];                          //  Tableau vide
const b = [1, 2, 3];                   //  Tableau de nombres
const c = [1, "a", null, 12.4, [1,2] ]; //  Les tableau peuvent contenir n'importe quel mélange de types
console.log(c[2]);                     //  Accès aux éléments : affiche null. La numérotation commence toujours à 0
c[58] = true;                          // Il est possible d'assigner n'importe quel élément d'un tableau.
                                       // Tous les élément non assignés valent 'undefined'


b.push(5);                             //  "push" pour ajouter un élément, b contient alors [ 1, 2, 3, 5 ]
b.pop();                               //  "pop" supprime et renvoie le dernier élément, b contient alors  [1, 2, 3]

console.log(b.length);                 // Affiche 3. La propriété "length" retourne la taille du tableau

const d = [[1,2,3], [4,5,6], [7,8,9]];  // Il est possible d'avoir des tableaux de tableaux
console.log(d[1][2]);                  // Affiche 6
```

Techniquement, en JavaScript les tableaux sont des objets. `typeof []` retourne `"object"`

JavaScript

Les fonctions des tableaux

- Taille d'un tableau : `array.length`
- Ajouter un élément à la fin: `array.push(2)`
- Retirer le dernier élément : `array.pop()` Renvoi l'élément retiré
- Trier le tableau : `array.sort()` Trie le tableau et le renvoie
- Inverser l'ordre des éléments : `array.reverse()` Inverse l'ordre des éléments et renvoie le tableau
- Appartenance : `array.includes('value')` Retourne `true` si le tableau contient 'value', `false` sinon

Et bien d'autres : https://www.w3schools.com/js/js_array_methods.asp

JavaScript

Les objets

Un objet en JavaScript est équivalent à un dictionnaire en Python

```
const object = {
    'key1' : 3,
    'key2' : "Value",
    'key3' : {
        'subkey1' : true,
        'subkey2' : 5
    },
    'key4' : [4,5,6]
};
```

Un objet est un ensemble de valeurs où chacune possède une clé. On accède à une valeur à l'aide de la clé, soit à la manière d'un tableau soit avec un point :

```
console.log(object['key2']); // "Value"
console.log(object.key2); // "Value"
```

JavaScript

Les objets

Il est possible de créer une clé directement en l'assignant

```
object['newkey'] = "newValue";
object.otherNewKey = "OtherNewValue";
```

Il est possible de supprimer une clé avec l'instruction `delete`

```
delete object['newkey'];
delete object.otherNewKey;
```

JavaScript

Les objets et les classes

TODO

JavaScript

Traverser des tableaux et des objets

Il existe la boucle `for(... of ...){}` pour itérer dans des tableau :

```
const table = ["a", "c", "e", "g"];
for(const value of table){
    console.log(value) // a c e g
}
```

Il existe la boucle `for(... in ...){}` pour itérer dans des objets :

```
const obj = {
    'k1' : "v1",
    'k2' : 3,
    'k3' : true
};
for(const key in obj){
    console.log(key + " : " + obj[key]); // k1 : v1    k2 : 3    k3 : true
}
```

JavaScript

Les fonctions

Les fonctions se déclarent de la manière suivante

```
function add(a,b,c){    // Déclare la fonction "add", disponible dans le contexte courant
    return a + b + c;
}

const result = add(4,6,8); // result = 18
console.log(typeof add) // Affiche "function"
```

En Javascript les fonctions sont des valeurs comme les autres, il est possible de les assigner à des variables :

```
const maFonction = function foo(a, b) { // Déclare une fonction "foo" assignée à la variable "maFonction"
    return a * b;
}
console.log(maFonction(3,4)) // Affiche 12
console.log(foo(3,4)) // ERREUR : foo n'est pas définit
```

Dans cet exemple, on a nommé notre fonction "foo". Ce nom est inutile car cette fonction est stockée dans "maFonction".

JavaScript

Les fonctions

Stocker des fonctions dans des variables est très courant en JavaScript. Il existe une syntaxe alternative sans donner de nom à une fonction. On appelle cela une fonction anonyme.

Si la fonction n'est composée que d'un `return`, il est même possible de l'enlever ainsi que les accolades. Les trois lignes ci-dessous sont équivalentes.

```
const operation1 = function nom_inutile(a,b){ return 1 + a * b } // Syntaxe lourde
const operation2 = (a,b) => { return 1 + a * b;} // Syntaxe plus légère, avec une fonction anonyme
const operation3 = (a,b) => 1 + a * b; // Dans ce cas, on peut omettre les accolades et le return
```

Exemple où une fonction est passée en paramètre d'une fonction :

```
function apply_to_table(table, func){       // Le paramètre func est une fonction appliquée au tableau
    for(let i=0; i<table.length; i++){      // On itère pour chaque élément du tableau
        table[i] = func(table[i]);          // Appel de la fonction avec l'élément du tableau en paramètre
    }
    return table;
}
console.log(apply_to_table([2, 4, 6], x => 3*x+1)); // Affiche [7, 13, 19]
```

JavaScript

Le DOM

Comment lire et écrire le contenu de notre document HTML avec du JavaScript ? Grâce au Document Object Model (DOM) !

Le DOM est l'interface permettant d'accéder à la page web. Il se caractérise par l'ajout de deux objets, accessibles partout dans le code : `document` et `window`.

Pour récupérer des éléments HTML :

```
const e1 = document.getElementById('monId');           // Retourne l'élément qui porte l'id "monId"
const e2 = document.getElementsByClassName('maClasse'); // Retourne un tableau avec les éléments de classe "maClasse"
const e3 = document.querySelector('p .large');          // Retourne le premier élément correspondant au sélecteur CSS
```

JavaScript

Le DOM

Une fois un élément récupéré, toutes ses propriétés sont modifiables

```
const elem = document.querySelector('p');            // Sélectionne le premier paragraphe de la page
elem.textContent = "Nouveau contenu du paragraphe";  // Assigne un nouveau texte au paragraphe
elem.style.fontSize = "20pt";                        // Assigne une nouvelle taille de police
```

Référence du DOM et ses fonctions : https://www.w3schools.com/js/js_htmldom.asp

Il est aussi possible d'attacher des événements aux éléments :

```
const elem = document.getElementById('MyId');                  // Récupère l'élément à modifier
elem.addEventListener("click", ()=>elem.textContent="Clic!");  // Au clic, modifie le texte de l'élément à "Clic!"
```

```
// document.body retourne toujours la balise body de la page actuelle
// Quand la page est complètement chargée ("load"), change le texte de l'élément info en "Loaded!"
document.body.addEventListener("load", ()=>document.getElementById('info').textContent="Loaded!");
```