JAVASCRIPT : Les bases et Environnement

1. Environnement de développement :

Vite :

Accessible via NPM

Aide à la création d’application Javascript, n’est pas utilisé en production.

Il y a deux parties :

Met en place un environnement de développement Javascript.

Génère un code javascript et assure la compatibilité du code.

Fonctionnement :

Code source -------🡪 rollup.js (en JS) + ESBUILD (en GO) -------🡪 Code Final

Rollup :

* permet de faire du Tree shaking (permet de récupérer que le code utile
* permet de créer des bundles
* convertir les modules en ESModules

ESBUILD :

* minify le code
* compatibilité des navigateurs

On va pouvoir rajouter des plugins : ex sass / TypeScrit / react / vue.js / babel

Création des bundles.

Création d’un serveur web de dev.

(bundle les dépendances, caches, hot module replacement, plugins)

On utilise ESNext en développement (les dernières versions de Javascript)

Installation et configuration de Vite :

* npm init -y
* npm install -D vite (-D uniquement dispo pour le dev, devDepencies)
* rajoute une commande start : vite (npm start) (pour le serveur de développement)
* rajout « type » : « module » dans le package json
* ajouter un script dans le html : <script type=  « module » src=  « script.js »>
* ajouter dans « scripts » : «build » : «vite build » (pour avoir accès à npm run build
* ajouter dans « scripts » : « preview » : « vite preview »
* créer un fichier vite.config.js
  + import {defineConfig} from «vite » ;
  + export default defineConfig({
  + build : {
    - outDir : «dossier »,
    - } ,
    - Plugins : []
  + }) ;

1. JAVASCRIPT

* Rappels Important
* Javascript est langage de programmation Objet / prototypé.
* C’est un langage interprété existant pour des applications côtés client et serveur.
* C’est un langage SYNCHRONE, contrairement aux idées reçues.
* Il est monothread.

1. La console Javascript

* console.log(‘’) ;
* console.error(« ») //avec lien avec l’endroit de l’erreur, il y a un sourcemap pour le dev.
* Console.assert(assertion, message) //permet d’afficher un message et la trace si l’assertion est false
* Console.count(label) //pertme de compter le nombre de fois que la ligne a été executée
* Console.table(tableau) //affiche sous forme de tableau
* Console.time() et timeEnd()
* Console.trace() permet d’afficher la trace d’appel.
* Console.warn()
* Console.info()

1. Variables

var => déclaration de variable

let => (ES6)

* Evite la déclaration de variable de même nom
* Reconnait la portée de block

const => pour créer des constantes

* Elle doit obligatoirement être initialisée à la déclaration.
* Sur un Objet ne rend constante que la référence.

1. Hoisting

Hoisting c’est la remonté des variables avant l’execution (lecture du script avant l’execution)

1. Type

(type statique / type dynamique)

Type dynamique, pas besoin de déclarer le type cela est fait à l’execution et peut changer en cours de route.

**Primatives :**

Booleens

Number

String

Undefined // quand il n’y a pas d’initialisation

Null

Symbol (ES6)

BigEnt(ES2020)

**Objets :**

Litteral

Tableau

Fonction

Date

…. Tout le reste.

Mot clef typeof permet de connaitre le type d’une variable.

1. Les opérateurs et les notions de précédence et d’associativité

* Précédence et associativité :

Précédence (priorité représentée par un nombre voir MDN plus c’est grand plus c’est prioritaire) si on a une précédence identique on utilise la règle d’associativité.

1. Les conversions :

Conversion implicite (coercion)

1 + «hello» => convertie en string

1 + bool => convertie en bool ; (donne 1 si bool true)

« » + bool => convertie en string

Conversion explicite

Number(‘4’) par exemple renverra NaN si jamais il ne peut pas convertir.

String()

parseInt()

etc…

1. Approfondissement des opérateurs de comparaisons et logiques.

== peut créer de la conversion

Exemple true == «true » renvoie false ;

Il est préférable d’utiliser === qui empêche la conversion implicite

Attention à la convection implicite et les comparaisons :

Petit exemple d’erreur possible

3 < 2 < 1 renvoi true à cause de l’associativité et de la conversion explicite

* 3 < 2 = false
* false < 1 // l’interpréteur va convertir false en nombre donc zéro résultat 0 < 1 est vrai…
* point pratique d’utilisation de la conversion explicite :

plutôt que de faire Boolean(a) pour convertir un nombre en boolean on peut faire !!a

ex :

let a = 1 ;

console.log( !!a) ;

1 conversion en boolean sur l’inverse deux on réinverse pour revenir sur la valeur initiale mais en boolean. Donc true.

1. Référence et valeur

(let, const, var) Une variable : c’est une adresse et une valeur.

Structure :

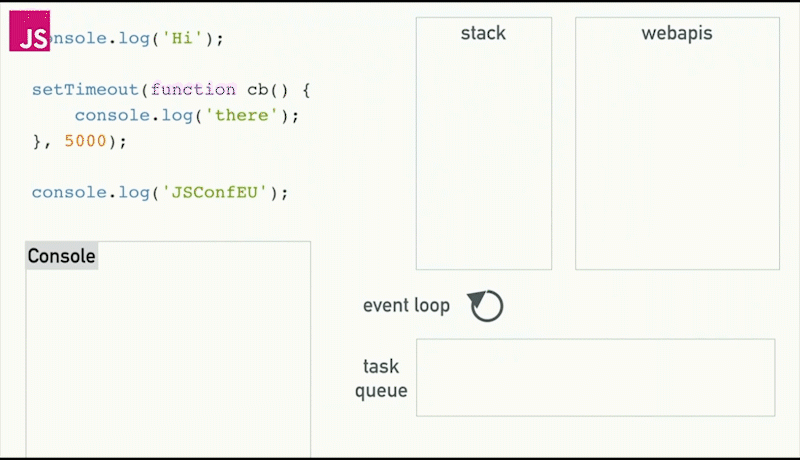
WEB APIs

Event loop

V8

Task queue

Stack + Heap



* Quelques rappelles

Stack : (plus rapide d’accès)

Primitives

Heap : (plus importante en taille)

Objets

Ex :

Let a = 1 ;

Let b= a ;

B += 1 ;

Résultat :

A=> 1

B=> 2

* Car il y a une copie par valeur c’est une primitive.
* Les valeurs sont comparées par valeur

Donc a === b

Let a = {b :1} ;

Let c = a ;

c.b = 2 ;

Resultat :

a.b => 2

c.b => 2

* Car c’est de la référence.

Les références sont comparées sur leur référence :

Let a = {b :1} ;

Let c = a ;

A === c;

Mais si :

C= {b:1}

Alors a != c;

1. Opérateur de coalescence des nuls

let usernarme = user.name || ‘John Doe’; (ici agit aussi sur les true false (aussi apres conversion))

let username = user.name ?? “john doe”; si c’est null ou undefined uniquement

1. Rappels : Structures de contrôle

* if else if et else.

Attention :

* chaine de caracteres vide est convertie en false
* chaine de caracteres est convertie en true
* On n’oublie l’utilisation de !!variable.
* Les ternaires
* Switch :
  + Ne pas oublier les breaks ;
  + Le default
  + switch(){case 1 : break ; default :}
  + On peut utiliser des blocks dans les cases, cela permet de créer des variables locales, ne dispense pas du break.
* for
  + break pour couper
  + continue pour couper l’iteration actuelle.
* while et do while. 0/n ou 1/n

1. les nombres et Math
2. Bases

3 types d’initialisation :

expression literal : 4 par exemple;

ou avec Number : Number(4) // qui permet aussi convertir

ou Fonction constructor : new Number(10) //objet

le format number permet d’encoder les nombres sur 64bits. Il respecte le format double précision.

* Limite de ce format : la perte de précision par overflow ou d’arrondi :

La ram utilise des fractions pour stocker en RAM ça peut donner des comportements étranges.

Ex :

console.log(0.1+0.2) // 0.30000000000000004, pour resoudre ce probleme on peut faire des opération sur des entiers et ensuite effectuer une division pour revenir sur un décimal (\* 10 et / par 100)

console.log(0.1\*0.2) // 0.020000000000000004

console.log(0.11/0.1) // 1.0999999999999999

console.log(123456789012345678) // 123456789012345680

* NaN est une valeur spéciale (not a number)

1. Décimaux, exposant et notation scientifique

const dec = 15.45

const dec = 1e6 => 1000000 //notation scientifique.

ES7 (ECMAScript 2016) introduit l’opérateur des puissances 2 \*\* 3 //8

1. Méthodes disponibles en global

parseInt() // un peu plus puissant que Number()

parseInt(nombre, base)

ex :

parseInt(«11», 2) => ca donnerait 3

parseInt(«5», 2)=> ca donnerait NaN car 5 n’est pas en base de 2

parseInt(«11.5»)=> donnera 11 car il va tronquer pour renvoyer que l’entier.

parseFloat() ;

+ «44»  // le plus avant la chaine de caracteres cast en int.

Number.isNaN(…) // attention undefined n’est pas un NaN.

1. Number

Attention : new Number(1) // donne un objet et non une primitive ce qui est pratique car ca donne acces à des méthodes.

Quand on essaye d’invoquer des méthodes de Number sur une primitive, il convertira automatiquement la primitive en Objet Number.

Par exemple :

.toFixed(2) //fixe le nombre apres la virgule. (au format string attention ! il faudra parseInt ou Float)

.toString()

.toExponential() //Mettre au format exponential.

.toPrecision() // permet de transformer le nombre en gardant une précision donnée mais sans changer sa notation sauf pour les entiers si la précision est inférieur au nombre de chiffres nécessaire pour l’écrire.

Sur Number on a des méthodes de class :

Number.isNaN()

Number.isInteger() // est ce que le nombre est un entier.

Number.isFinite()

Number.parseFloat()

Number.parseInt()

Number.isSafeInteger()

1. L’objet natif Math.

Math.pow(x,x) //puissance

Math.PI //donne PI

Math.ceil(2.55) //arrondi à l’entier

Math.floor(1.9) //arrondi à l’entier inférieur.

Math.round(1.4) //tronque à l’entier.

Math.max(1,2,5,8) // donne le max

Math.random() // donne un nombre aléatoire entre 0 et 1

Ex : Math.round(Math.random() \* 100)) ; va donner un nombre entre 1 et 100 entier (grace au round)

Avec un interval : Math.random() \* (max - min) + min;

1. Les chaines de caractères
2. Introduction

Une chaine de caractere est une primitive en javascript.

Attention une chaine n’est pas modifiable.

On utilise ‘’ ou « » pour la partie littéral.

Const a = ‘a’ ;

Const a = String(55)

Const a = new String(‘ddd’) // version Objet donne acces des méthodes.

Si on utilise une méthode sur une primitive il convertie automatiquement.

Operateur de concaténation : +

1. Les littéraux de modèle

Depuis ES6 :

Permet d’écrire sur plusieurs lignes et faire de l’interpolation :

`je suis

${expression}

Une chaine`

1. Index et length

Index : On peut utiliser un chaine comme un tableau pour récuperer un caractere depuis sont index.

Length est une propriété pour récupérer la taille d’une chaine : chaine.length.

Cela vient de l’Objet length.

1. Méthodes disponibles utiles sur les chaines de caractères

a.endsWith(‘chaine’) // est ce que la chaine fini par cette chaine.

a.startsWith(‘’)//idem pour le debut

a.indexOf(chaine, optionnel index de depart) //recupere l’index du debut de la chaine dans la chaine. (retourne -1 si il ne trouve pas)

a.lastIndexOf() // le dernier index de la recurrence.

a.search(chaine) //fonctionne de manière assez similaire.

a.replace(chainearemplacer, remplacement) // remplace une chaine

Attention une chaine n’est pas modifiable il faudra recuperer la nouvelle valeur.

a.slice(index de debut, index de fin) // decoupe une chaine et renvoie la valeur decoupé

a.trim() // enleve tous les espaces au début et fin.

a.toUpperCase() toLowerCase()

a.split(caractere) // retourne un tableau splité.

a.charAt(index)

a.substring() // un peu comme slice mais ne peut utiliser que des valeurs d’index positives

si index de debut est superieur a index de fin il affiche en inversé.

a.repeat(5) // permet de concatener la meme chaine plusieurs fois.

a.padStart(taille, caractere de remplicage)

a.padEnd()

.includes(chaine) //retourne true ou false si la chaine est présente.

1. Expression rationnelles (regex)

Expression particuliere qui permet de matcher avec une chaine de caracteres.

Déclaration ;

Const regex = /regex/flag

Ou

Const regex = new RegExp(‘exp’, ‘flag’) ;

Evaluer un regex ;

const evaluate = /regex/.test(chaine) ; //renvoi true ou false;

const evaluate = regex/.exec(chaine) ; //extrait la chaine

* Quelques règles :

\d n’importe quel type de nombre ex : /\ds/ => ok pour 4s ou 1s

\d\* entre 0 est un certain NOMBRE ex : /\d\*s/ => ok pour 4s ou 1s ou 1555s

\d+ entre 1 est un certain NOMBRE

\d{1,3} de 1 à 3 nombres

\D tout sauf des nombres

. n’importe quel type de caractere ex : / .\d{1,3}/ va retourner les patterns avec un espace un succession de nombre allant de 1 à 3 caracteres

\. Selectionne le caractere . spécifique (par rapport au .)

[chaine] une chaine précise

[^chaine] sauf la chaine

[chaine]+ de 1 à x fois la chaine.

[chaine]{1,3} entre 1 à 3 itérations.

\w n’importe quel type de digit

[a-z]

[A-Z]

[a-z] ? optionnel ca peut etre la ou non

Ex : /\d{3}[A-Z] ?/ forcement 3 digits et eventuellement une lettre majuscule

Ex : /\d{3}[A-Z] ?[a-z]/ idem et suivi d’une lettre minuscule

() // permet de grouper

Ex : /(.\*)\./ sur la chaine matruc.exe matchera avec matruc et renverra la chaine si c’est en exec.

Quelques méthodes utiles :

.match(regex) // permet de directement atcher une regex sur une chaine sans passer par test

.replaceAll(regEx, remplacement) ; (mettre le flag g pour effectuer l’action sur toute les occurrences ex : ‘/\d/g’

Fonctionne aussi avec .split()

* Les flags

. g pour toutes les occurrences

. i non sensible à la casse