**Le JavaScript**.

- Assigné une valeur

var num = 5; console.log(num); 5

var chiffre; console.log(chiffre) undefined (non-défini)

console.log(chiffre + num) NaN (not a number)

- Le camel Case

Pour les noms de variable.

- Les nombres decimaux

console.log(11, 5864+2, 541-4) 11 5866 537

La virgule sert à séparé plusieurs résultats. Si l'on veut mettre un chiffre à valeur decimale, on met un point.

- Les priorités de calcul

les multiplications et division en premier sauf en cas de parenthèses qui là, seront prioritaire, suivie de l'addition et la soustraction.

- le modulo

console.log(10 % 3) 1

c'est le reste de la division, Utile pour savoir si quelque chose est pair ou impair.

console.log(10 % 2 ) 0

Ici le résultat est pair car il retourne 0.

- L'incrémentation

L'incrémentation, c'est le fait d'ajouté 1.

La décrémentation, c'est le fait de soustraire 1.

var num = 0;

num++; (num = num +1)

console.log =(num); 1

var num = 5;

num--; (num = num -1)

console.log =(num); 4

On peut aussi l'écrire:

var num = 0;

num = num + 1;

console.log =(num); 1

ou bien

var num = 0;

num += 3; (num = num + 3)

console.log =(num); 1

- **Les chaines de caractères**

var texte = "hello";

var txt = 'hello';

On peut utiliser les doubles ou simple côte, mais par pratique on utilise les doubles côte.

Il est possible d'utilisé la concaténation.

- La valeur **.length**

- la valeur .length fonctionne avec les chaines de caractères et le nom des variables, mais ne fonctionne pas avec des valeurs numerique.

ex: var nom = "Coquoz";

console.log(nom.length); 6

- Pour acceder à une lettre dans un mot, il faut mettre des crochets(comme dans un tableau, la première lettre = 0).

ex:var nom = "Coquoz";

console.log(nom[3]); u

- Pour connaitre la valeur de la derniere lettre.

ex: var nom = "Coquoz";

console.log(nom[nom.length -1]);

- **modèle de libellé ou template String** :

Le langage JavaScript permet d'inclure des expressions dans une chaîne de caractères lorsque cette chaîne est délimitée par une paire d'accents graves(atl gr + 7 ou altd 9 et 6 en même temps au pavé numérique) seuls ou backticks ( `...` ). Une telle chaîne est appelée un modèle de libellé ou template literal. A l'intérieur, les expressions sont indiquées par la syntaxe ${expression}.

On utilise souvent cette possibilité pour créer des chaînes intégrant des valeurs de variables.

**const** country = "France";

**console.log**(`Je vis en ${country}`); // "Je vis en France"

**const** x = 3;

**const** y = 7;

**console.log**(`${x} + ${y} = ${x + y}`); // "3 + 7 = 10"

Il est possible de mettre une condition ternaire: (lesson 54 de Antho)

ex :

**const** personnage = {

nom : "Jon Snow",

force : 15,

arme : "épée"

};

**const** text = `${personnage.nom? personnage.nom:"Yan"} est mon nom.`;

Si le nom de mon objet n'est pas renseigné alors ce sera Yan qui sera dans la console.

Il est aussi possible d'aller à la ligne pour déclarer par exemple du html.

- **Les boites de dialogue** :

il y a **comfirm()** afficher une boîte de dialogue qui demande une confirmation.

il y a **alert()** faire apparaître une boîte de dialogue qui affiche un message.

et il y a **prompt()** faire apparaître une boîte de dialogue qui demande une information à l'utilisateur.La valeur enregistrer est un String.

- **Saisie d'un nombre** :

(OpenClassRooms => jouer avec les variables)

Quel que soit le texte saisi, l'instruction **prompt**() renvoie toujours une valeur de type chaîne. Il faudra penser à convertir cette valeur avec l'instruction **Number**()(retourne un décimal), **parseInt()**(retourne un entier) ou **parseFloat()**(retourne un nombre décimale) si vous souhaitez ensuite la comparer à d'autres nombres ou l'utiliser dans des expressions mathématiques. (ne fonctionne pas avec node)

**const** saisie = **prompt**("Entrez un nombre : "); // La valeur saisie est de type chaîne

**const** nb = **Number**(saisie); // nb est de type nombre

// ...

Il est possible de combiner les deux opérations (saisie et conversion) en une seule ligne de code, pour un résultat identique :

**const** nb = **Number**(**prompt**("Entrez un nombre : ")); // nb est de type nombre

// ...

Ici, le résultat de la saisie utilisateur est directement converti en une valeur de type nombre par l'instruction Number() et affecté à la variable nb.

- **Les tableaux : array**

Dans un tableau, il est possible de stocker des entiers, des varibles de tous types, des strings et aussi des tableaux.

ex: var nom = "Coquoz";

var age = 34;

var stats = ["attaque", "défense"];

var tab = [nom , age, stats];

console.log(tab);

Pour accéder à la valeur défence:

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

Pour changer la valueur défence:

var nom = "Coquoz";

var age = 34;

var stats = ["attaque","defense"];

var tab = [nom , age, stats];

tab[0] = "John";

tab[1]= 12;

tab[2][1]="riposte";

console.log(tab);

**Le push, le pop, le splice, le shift et le unshift** :

**var** arr = ["un",2,"trois"];

arr.**push** = 4;

push met dans un tableau le 4 en derniere position.

**console.log**(arr);

**var** retrait = arr.**pop**()**;**

le pop retire le dernier élément du tableau.

**console.log**(arr)**;**

**console.log**(retrait);

Le **splice** s'utilse avec des parametres dans lesquels vont se positionné l'*indice* à partir duquel il faut supprimé les éléments puis *le nombre d'éléments à enlever*.

ex :

**const** films = ["Le loup de Wall Street", "Vice-Versa", "Babysitting"];

films.**splice**(0,1);

Il restera dans le tableau vice-versa et babysitting.

Pour les tableaux simple et multidimansion

**tableau simple**

monTableau.***splice***(0,2); (indice, nbr élément à supprimé)

monTableau.***splice***(0,0,"chouette", "loup", "beaucoup"); (indice, nbr élément à supprimé, élément à rajouté)

console.log(monTableau);

monTableau.**splice**(*[début], [nbASupprimer], [élément(s)]*) : retire, remplace ou ajoute des éléments.

Début : l'index à partir duquel commencer le changement, si négatif, part de la fin du tableau

nbASupprimer : un entier indiquant le nombre d'éléments à retirer ou remplacer

Element(s) : les éléments à ajouter à partir du début précisé. Si aucun élément n'est spécifié, alors n'en ajoutera pas.

**tableau multidimention**

monTableau2D.*splice*(0,1); supprime le premier tableau

monTableau2D[0].*splice*(0,1); supprime le 1er élément de mon 1er tableau

console.log(monTableau2D);

Rajouter un tableau dans mon tableau2D ex : avec des ages.

monTableau2D.***splice***(2,0,["30","45","55"]);

console.log(monTableau2D);

Pour le shift et le unshift c'est pareil mais c'est pour le premier élément du tableau.

Pour parcourir un tableau, il y a 2 solutions le **forEach** et **for of** :

ex :

**const** films = ["Le loup de Wall Street", "Vice-Versa", "Babysitting"];

films.**forEach**(film => {

console.log(film);

});

**---------------**

**const** films = ["Le loup de Wall Street", "Vice-Versa", "Babysitting"];

**for** (const film **of** films) {

console.log(film);

}

- **La concatenation** :

exemple spécial:

var a 2;

var b "Yan";

console.log("le nombre est : %d",a); d pour decimal.

console.log("le prénom est : %s",b); s pour string.

- **la fonction** :

une fonction se déclare comme suit :

function nomDeLaFonction(){

corps de la fonction.

}

ON peut mettre un parametre par defaut au cas ou la valeur attendu n'est pas rentrer.

ex :

function monNom (nom = "no name") {

console.log(nom);

}

monNom(); retourne "no name" si a l'appele de la fonction le nom n'est pas renter en parametre.

- **La condition ternaire** :

var a=1;

var b=2;

console.log(a==b?"Vrai":"Faux"); Faux

console.log(a!=b?"Vrai":"Faux"); Vrai

**==** : Renvoie true si les opérandes sont égaux après conversion en valeurs de mêmes types.

**!=** : Renvoie true si les opérandes sont différents.

**===** : Renvoie true si les opérandes sont égaux et de même type. Voir Object.is() et égalité de type en JavaScript.

**!==** : Renvoie true si les opérandes ne sont pas égaux ou s'ils ne sont pas de même type.

**>** : Renvoie true si l'opérande gauche est supérieur (strictement) à l'opérande droit.

**>=** : Renvoie true si l'opérande gauche est supérieur ou égal à l'opérande droit.

**<** : Renvoie true si l'opérande gauche est inférieur (strictement) à l'opérande droit.

**<=** : Renvoie true si l'opérande gauche est inférieur ou égal à l'opérande droit.

ATTENTION : => ceci n'est pas un opérateur, il sagit de la fonction flechée.

- **Les différantes condition** :

Il y a le if, else if , else, qui se déclare comme ca :

**if**(condition booleen true){

action;

}

**else if** (condition booleen true){

action;

}

**else**{

action;

}

et le **switch** :

**switch** (expression) {

case valeur1:

// instructions exécutées quand expression vaut valeur1

break;

case valeur2:

// instructions exécutées quand expression vaut valeur2

break;

...

default:

// instructions exécutées quand aucune des valeurs ne correspond

}

- **Les Objets** :

var chevalier = {

"force":15,

"vitesse":5,

"faiblesse":["mage","ninja"]

};

Pour accèder au propriétés de l'objet :

console.log(chevalier);

puis avec la notion de pointage, on peut avoir accès aux propriétés :

console.log(chevalier.force); 15

console.log(chevalier.faiblesse[0]); mage

Comment ajouter, modifier et supprimer des paramètres d'un objet ?

Modifier :

chevalier.force = 25;

**console.log**(chevalier.force); 25

ajouter :

chevalier.arme = "épée";

**console.log**(chevalier); force:25, vitesse:5,faiblesse['mage','ninja'],arme:épée

supprimer :

delete chevalier.arme;

Pour savoir si l'objet a une propriété voulue :

console.log(chevalier.hasOwnProperty("arme")); false

il est possible d'avoir un objet dans l'objet :

**var** chevalier = {

"force":15,

"vitesse":5,

"faiblesse":["mage","ninja"]

"compagnon"{

"animal" : chien,

"aide" : "écuyer"

}

};

- **Le while** :

var i=0;

while(i<10){ // entre parenthese, la condition.

i++;

console.log(i);

}

- **Le random** :

La fonction **Math.random()** génère un nombre flotant qui va de 0.000000000001 à 0.9999999999999.

Pour aller au delà de 0 il faut le multiplier.

ex :

**var** random = **Math.random**() \* 6;

console.log(random); 5.5451198494943; //la valeur sera inferieur à 6.

Si l'on veux un nombre entier, il faut mettre la fonction **Math.floor()**.

ex :

**var** random = Math.floor(**Math.random**() \* 6);

console.log(random); 5

Mettons que l'on souhaite faire un jeu aléatoire avec un dé, il suffit de rajouté +1.

**var** random = **Math.floor**(**Math.random**() \* 6 + 1);

console.log(random);

- **la fonction fléchée** :

(leçon 55 antho, leçon 80 LNL)

fonction fléchée simple, sans parmètres

**let** monTest = () => console.log("Mon test de fonction ");

monTest();

**const** arr = [1,2,3,4,5];

**const** arrPlusUn = arr.**map**(nombre=>`${nombre}+1=${nombre +1}`);

console.log(arrPlusUn); affiche +1 au tableau

**const** arrPlusUn = arr.**map**(() => `5`);

console.log(arrPlusUn); le tableau retourne 5, cinq fois.

Dans une methode, la fonction fléchée est forcément annonyme et doit avoir des parenthèses. Pour quelle ne le soit plus, il faut la mettre dans une variable:

ex :

**const** maFonction= () => console.log("coucou");

- **Les closures (fermeture) :**

(leçon 81 LNL)

Les closures désigne une fonction dans une fonction.

Elles permettent de garder en memoire les variables locale dans la fonction dans laquel elle sont, même si la fonction dans laquel elles sont est terminer.

Ex 1:

function bonjour(prenom){

let resultat = “Bonjour ”+prenom; // variable locale

// closure

let maClosure = () => console.log(resultat);

return maClosure;

}

let maFonction = bonjour(“Yan”);

maFonction();

Ex 2:

function timer(){

let seconde = 0;

let maClosure = () => {

return ++seconde;

}

return maClosure;

}

let monTimer = timer();

comme il n'y a pas de return, il faut faire un console log

console.log(monTimer()); // renoie 1

console.log(monTimer()); // renoie 2

Démontre bien que la closure sauvegarde la variable seconde.

- **le destructuring avec un objet** :

(leçon 57 antho)

C'est faire appel au divers propriétées de l'objet, voir les modifiées à travers une variable commune.

création d'un objet:

**const** joueur = {

nom : "Jon",

type : "Chevalier",

arme : "épée"

}

la variable :

const {nom, type: genre, arme = "lance"} = joueur;

La propriete type est changer en genre (:), et la valeur épée si elle n'existe pas en lance (=).

console.log(nom, genre, arme);

- **le destructuring avec un tableau** :

c'est comme pour les objets mais avec de [] pour la variable.

**const** stats = [154, 12, 78, 28];

**const** [attaque , defense, vitesse, magie]= stats;

console.log(attaque); 154

- **le destructuring avec échange de valeurs** :

**let** maCarte = "As de trèfle";

**let** taCarte = "Roi de pique";

[maCarte, taCarte] = [taCarte, maCarte];

console.log(maCarte, taCarte);

Ici l'échange se fait en une ligne et évite de créer un nouvelle variable.

- **Le spread Operator** :

le spread Operator s'utilise sur tous ce qui est itérable(de 1 en 1).

ex :

**const** listeAmis = ["Ken","Toki","Raoul"];

**const** listeFamille = ["Saroeun","Kean","Tuko"];

console.log(listeAmis, ...listeFamille);

Ici dans la console, listeAmis est entre crochet tandis que listeFamille ne l'est pas.

**const** liste = [...listeAmis, "kawai", ...listeFamille];

Ajout possible dans la liste de Kawai.

il est possible de faire une copie de notre liste dans une variable, mais il faut bien mettre les crochets au cas ou l'on souhaite rajouté des valeurs.

**const** copieListe = [...liste];

copieListe.push = "Tortue";

Destructuration et spread.

**const** listeBis = ["Jacques", "Pierre", "Bob", "Nokia","HTC"];

**const** [ami1, ami2, ami3, ...telephones] = listeBis;

console.log(ami1+" , "+ami2,telephones);

- **L'objet Set** :

Permet de stocker des valeurs uniques (il n'ya pas 2 fois la même valeur) de tout type. Il s'agit d'une sécurité comparé à un simple tableau. Je vous recommande d'utiliser cet objet lorsque vous voulez stocker des informations sensibles qui ne doivent pas être dupliquées. L'exécution d'un objet Set est également plus rapide qu'un objet Array (donc qu'un tableau), même si sur de petits tableaux, cela ne se remarque pas.

**let** monTableau = [45,23,78,11,99,45,62,78];

**let** monSet = new Set(monTableau);

**console.log**(monSet); // Set { 45, 23, 78, 11, 99, 62 }

*Pour ajouter une valeur* :

monSet.**add**(15);

monSet.**add**("Hello");

monSet.**add**(["Mon","nom","est","personne"]);

**console.log**(monSet); // Set { 45, 23, 78, 11, 99, 62, 15, "Hello", ["Mon","nom","est","personne"]}

*Pour supprimer une valeurs* :

monSet**.delete**(23);

**console.log**(monSet); // Set { 45, 78, 11, 99, 62, 15}

Pour voir combien d'éléments il y dans le Set:

**console.log**(monSet**.size**);

- **L'objet Map** :

Permet de stocker des paires clé-valeur et de mémoriser leur ordre d'insertion.

**let** monMap = **new** **Map**([

["prénom","Yan"],

["nom","Coquoz"]

]);

**console.log**(monMap);// Map { 'prénom' => 'Yan', 'nom' => 'Coquoz' }

Pour ajouter une valeur :

monMap.**set**("poste","Chef de la tribu");

*Pour supprimer une valeurs (on met simplement la clé)*:

monMap.**delete**("nom");

*Autre* *façon* :

let utilisateur = new **Map**();

utilisateur.**set**("Wolverine",{

email : "logan@marvel.com",

poste : "X-men"

});

utilisateur.**set** ("Deadpool",{

email : "licorne.rose@marvel.com",

poste : "anarchiste"

});

**console.log**(utilisateur);

// Map {

'Wolverine' => { email: 'logan@marvel.com', poste: 'X-men' },

'Deadpool' => { email: 'licorne.rose@marvel.com', poste: 'anarchiste' }

- **L'objet WeakSet** :

Offre une flexibilité supplémentaire en comparaison de l'objet Set. Il n'accepte que des objets. Une fois que l'objet stocké dans le WeakSet passe à null, il est automatiquement retiré.

// objet a

**let** voitureA = {

constructeur : "Tesla",

modele : "CyberTruck"

}

// objet b

**let** voitureB = {

constructeur : "Machin",

modele : "Bidulle"

}

**let** voiture = **new WeakSet();**

// ajouter un objet au WeakSet

voiture.**add**(voitureA);

voiture.**add**(voitureB);

**console.log**(**"**ajout d'objet : ");

**console.log**(voiture);

// supprimer un objet WeakSet

voiture.**delete**(voitureB);

**console.log**("suppression d'objet : ");

**console.log**(voiture);

// ou bien une fois les objets créer on les met directement dans la création du weakSet

let voitures = **new WeakSet**([voitureA,voitureB]);

**console.log**(voitures);

- **L'objet WeakMap** :

Offre une flexibilité supplémentaire en comparaison de l'objet Map. Il n'accepte que des objets en clé. On utilise souvent cet objet pour stocker des données. Une fois que l'objet en clé n'existe plus (par exemple, si sa valeur passe à null), la valeur est automatiquement supprimée également.

//

**let** car = new **WeakMap()**;

// objet index

**let** index = {

id : 1

}

// objet index2

**let** index2 = {

id : 2

}

// objet carA

**let** carA = {

constructeur : "tesla",

modele : "Cybertruck"

}

// en param, une clé qui est un objet.

car.**set**(index , "5");

// en param, une clé et une valeur qui sont des objets.

car.**set**(index2, carA)

**console.log**(car);

**En conclusion** :

Vous n'utiliserez pas souvent ces nouveaux objets que JavaScript a ajouté récemment, néanmoins, si vous devez un jour en utiliser un, vous aurez tout ce qu'il vous faut dans cette formation pour le faire. Si vous ne devez retenir qu'une chose, c'est que ces objets servent surtout à stocker des données liées à un objet. Par exemple, si vous avez une application de messagerie, et un objet "utilisateur", vous pouvez utiliser un WeakMap pour stocker ses messages.