ADRAR \o/ PÔLE NUMERIQUE
INFRASTRUCTURES SYSTÈMES & RÉSEAUX
CYBERSÉCURITÉ INFRA. & APPLICATIONS
DEVOPS / SCRIPTING AUTOMATISATION
DEVELOPPEMENT WEB & MOBILE

ADRAR FORMATION

JAVASCRIPT

Durée Totale du Module : 21H

Auteur : Mathieu Paris

Relu, validé & visé par :
- [x] Jérôme CHRETIENNE
- [x] Sophie POULAKOS
- [x] Mathieu PARIS

Date création : xx / xx / 20xx

Date révision : xx / xx / 20xx

ADRAR FORMATION

ADRAR PÔLE NUMERIQUE
INFRASTRUCTURES SYSTÈMES & RÉSEAUX
CYBERSÉCURITÉ INFRA. & APPLICATIONS
DEVOPS / SCRIPTING AUTOMATISATION
DEVELOPPEMENT WEB & MOBILE

ADRAR FORMATION

JAVASCRIPT

Durée Totale du Module : 21H

Auteur : Mathieu Paris

Relu, validé & visé par :
- ☒ Jérôme CHRETIENNE
- ☒ Sophie POULAKOS
- ☒ Mathieu PARIS

Date création : xx / xx / 20xx

Date révision : xx / xx / 20xx

TECH' CARE

ADRAR FORMATION

Présentation de Javascript

Voici Brendan Eich (une sorte d'humain)

Nous sommes à l'été 1995, dans les bureaux de la Netscape Communications Corporation, le web est en effervescence grâce aux langages HTML et CSS, en parallèle les applications de bureau sont très populaires pour les différents OS (Windows, MacOs, Linux …) et le langage de programmation Java propose une machine virtuelle (très populaire) permettant de s'affranchir des barrières liées au langages spécifique des OS.
Brendan va donc travailler sur la problématique de créer un nouveau langage qui devra répondre aux contraintes suivantes :
Un langage de programmation
Multi Environnement
Orienté Objet
Permettant de rendre des pages web dynamiques
10 Jours plus tard naissait Javascript…

Langage de programmation

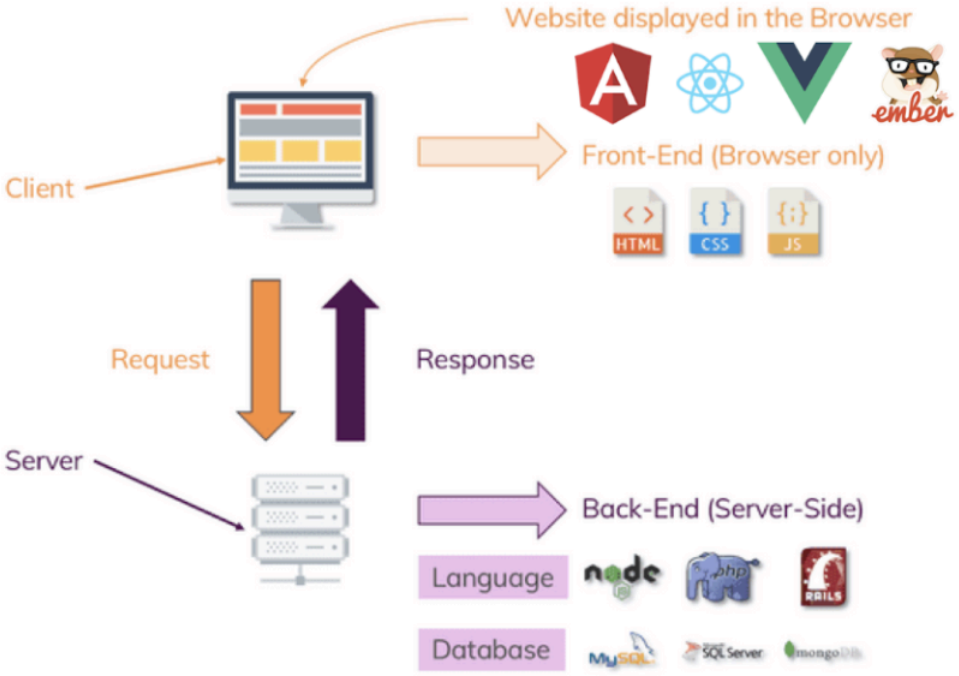
Brendan s'est inspiré de nombreux langages de programmation, notamment de Java mais en simplifiant la syntaxe pour les débutants.

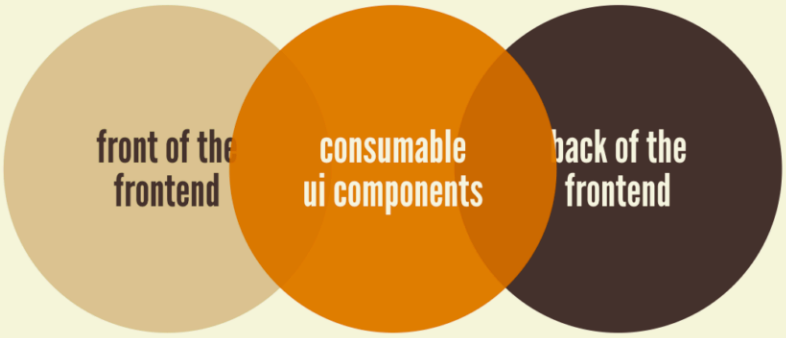
Multi Environnement

L'aspect multi environnement réside dans le fait que JS s'exécute au sein du navigateur web de l'utilisateur, et tout le monde utilise un navigateur (encore + aujourd'hui) et on retrouve les navigateurs partout (Desktop, Mobile,Tablettes, TV, Montres …)
(*Il existe des frameworks JS comme Electron permettant de créer des apps desktop à partir d'un app web.)

En effet les navigateurs sont composés de 2 moteurs : un moteur de rendu (Render Engine) ainsi qu'un moteur JS (Javascript Engine).

JS s'exécute dans le navigateur certes mais coté client, Front-End, sur la page HTML qui est déjà chargée.
Par la suite après la version ES6 de JS, on peut exécuter du JS côté serveur (Back-End) via des technologies comme Node.js ou encore Deno.



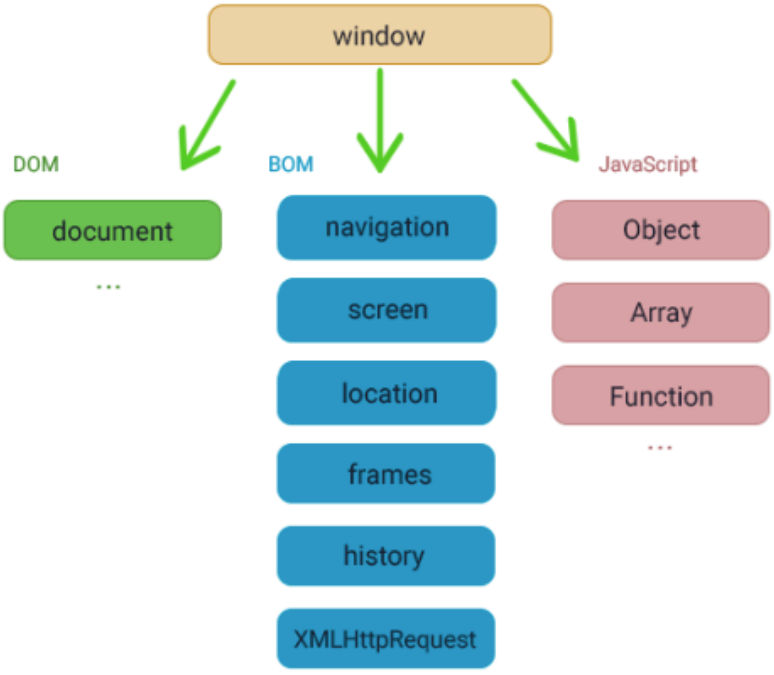


Orienté Objet

Javascript est un langage conçu pour être orienté objet, à tel point que l'on peut entendre dire « en JS tout est objet », c'est en ce point que JS diffère par rapport à d'autres langage comme Java, pour être plus précis, JS est un langage de programmation orienté objet à prototype, cela signifie que JS va fournir les outils de base du langage (créer une chaine de caractères, une fonction etc..) via un système d'objet là ou Java utilise des classes (en JS une chaine de caractères, une fonction et même une classe sont des objets).

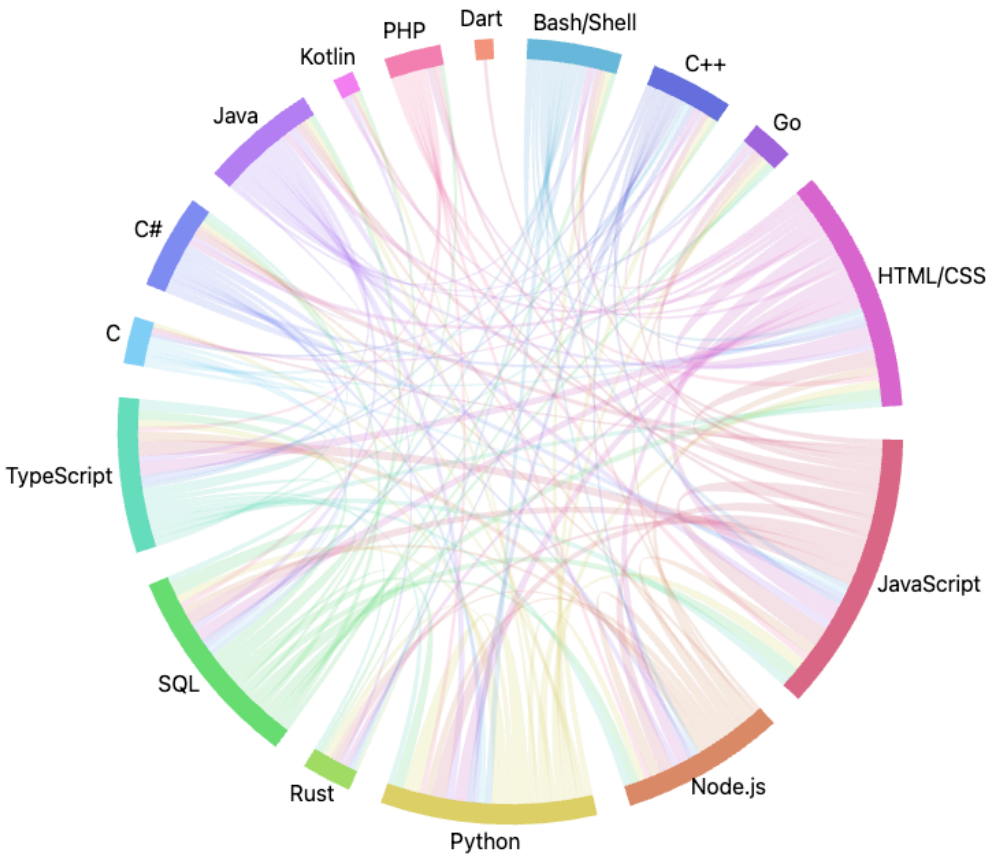
Web Dynamique

JS va apporter un ensemble d'outils pour manipuler ce qu'il se passe dans la page web, au sein du navigateur de l'utilisateur.
Techniquement JS contient déjà des objets (avec des propriétés et des fonctions) pour tous les éléments affichés dans la fenêtre de l'utilisateur, l'objet window dans lequel va être exécuté notre code JavaScript (nos variables, nos fonctions, nos classes, nos tableaux, etc.) qui pourra manipuler les objets du BOM (Browser Object Model) pour contrôler l'historique ou la navigation et manipuler les objets du DOM (Document Object Model), les éléments HTML et CSS chargés sur la page.



JS : Actuellement
Très vite JS s'est imposé comme un langage incontournable du Web, en combinaison à HTML et CSS dans un premier temps pour améliorer le design des site web, puis au travers de librairies populaires comme JQuery et les évolutions majeures de ES6 qui ont amené de nouvelles technologies côté serveur (Node / Deno) mais aussi les Frameworks (React, Angular, Vue, Svelte…)

https://stackoverflow.blog/2021/08/02/2021-stack-overflow-developer-survey-results/



L'évolution fulgurante de JS (un incontournable du web ?)

2008

2021

Un site pour faire l'état de l'art de Javascript :
https://2022.stateofjs.com/en-US/libraries/

Algo / Syntaxe JS Moderne (ES6 et +)

Dans cette section nous allons nous intéresser aux notions de base de la syntaxe de Javascript.

Amélioration de la DX

Visual Studio Code : super paramétrable, gratuit, plein d'extensions
Visual Studio Code .dev : la version web de vscode (c'est cool on peut le connecter direct à GitHub, ya aussi la plupart des extensions) (vs code depuis une tablette ? Smartphone ? À tester)

Alternatives :
Webstorm
Fleet

En ligne
codepen
Stackblitz

Extensions vscode

LiveShare : Pour partager / streamer son code (travail collaboratif / débug)

GitGraph : Pour mieux visualiser les repositories git sur vscode

Better Comments : Pour des commentaire en couleur
(ça bug un peu avec des frameworks mais pour JS ça va)

Vite : Un ensemble d'outils très utiles pour facilement et 'vite ' créer puis initialiser des projets (très utilisé avec les principaux frameworks qui peuvent de base comporter beaucoup de fichiers)
Le fonctionnement se fait via des CLI (des lignes de commandes dans le terminal).
Il est nécéssaire d'installer node.js sur votre serveur ou votre ordinateur (cela va créer un serveur de développement web)

Setup de Javascript

Il nous faut un éditeur de code, par exemple Visual Studio Code et pour plus de confort une extension permettant de simuler un serveur de développement Live Server
Et bien entendu nous aurons besoin d'un navigateur internet.(basé sur chromium, Firefox, Safari, etc…)

Setup Classique

Dans l'approche la plus commune on va bien faire attention de relier le fichier JS à la fin du fichier HTML (juste avant la fermeture de la balise <body>)

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Intro JS</title>
</head>
<body>

    <script src="app.js"></script>
</body>
</html>
```

Et un petit script dans le fichier Javascript

```javascript
console.log('Bienvenue dans Javascript');
```

Côté Template

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Intro JS</title>
</head>
<body>
      <!-- On peut aussi faire du JS directement ici  -->
    <script>console.log('Hello World')</script>
</body>
</html>
```

Async ou Defer

On peut aussi placer la balise script pour relier le fichier JS dans le haut du HTML, dans la balise <head> MAIS il faut ajouter l'attribut async ou defer

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Intro JS</title>
    <script src="app.js" async></script>
</head>
<body>

</body>
</html>
```

https://www.alsacreations.com/astuce/lire/1562-script-attribut-async-defer.html

Defer : Antérieur à la vague HTML5, l'attribut defer existait déjà dans les "anciennes" versions d'Internet Explorer. Il signifie que le navigateur peut charger le(s) script(s) en parallèle, sans stopper le rendu de la page HTML. Contrairement à async, l'ordre d'exécution des scripts est préservé, en fonction de leur apparition dans le code source HTML. Il est par ailleurs reporté à la fin du parsing du DOM (avant l'événement DOMContentLoaded). De nos jours, cet attribut présente moins d'intérêt car les navigateurs disposent par défaut de techniques internes pour télécharger les ressources en parallèle sans nécessairement attendre les autres.

Async : Nouveau venu dans HTML5, async signifie que le script pourra être exécuté de façon asynchrone, dès qu'il sera disponible (téléchargé). Cela signifie aussi que le navigateur n'attendra pas de suivre un ordre particulier si plusieurs balises <script> sont présentes, et ne bloquera pas le chargement du reste des ressources, notamment la page HTML. L'exécution aura lieu avant l'événement load lancé sur window et ne sera valable que pour les scripts externes au document, c'est-à-dire ceux dont l'attribut src mentionne l'adresse.

Ce comportement est bien pratique pour gagner en temps de chargement, il faut cependant l'utiliser avec prudence : si l'ordre n'est pas respecté, un fichier exécuté de façon asynchrone ne pourra attendre le chargement d'un précédent, par exemple s'il en utilise des fonctions voire un framework. Il ne faudra pas non plus compter appeler document.write() pour écrire dans le document HTML puisqu'il sera impossible de savoir à quel moment les actions seront déclenchées.

Les Variables

Pour rendre une application web dynamique, il nous faut manipuler, interagir avec des données, par exemple le nombre de billets restant pour un concert, une commande avec un numéro de produit, un prix et une date, une adresse mail, etc. Un langage de programmation va donc utiliser des variables pour enregistrer ces données (pour l'ordinateur c'est un espace mémoire).

Anatomie d'une variable

Une variable va se définir par 3 aspects :
- Un NOM (pour identifier ce que confient la variable)
- Un TYPE (un nombre, une chaîne de caractère, etc…)
- Une VALEUR (c'est le contenu de la variable)

JavaScript est un langage dont le typage est **faible** et **dynamique**. Cela signifie qu'il n'est pas nécessaire de déclarer le type d'une variable avant de l'utiliser. Le type de la variable sera automatiquement déterminé lorsque le programme sera exécuté. Cela signifie également que la même variable pourra avoir différents types au cours de son existence

https://developer.mozilla.org/fr/docs/Web/JavaScript/Data_structures

Déclarer une variable et assigner une valeur

⚠️ : Pour assigner un VALEUR à une variable il faut au préalable que cette variable soit initialisée (avec le mot clé let ou const suivi du NOM de la variable).
PS : Prendre aussi l'habitude de finir une instruction JS avec ;

```
let maVariable;
```

Ensuite nous pouvons lui assigner une **VALEUR.**

```
maVariable = 'Hello World';
```

C'est crucial car cela permet à cette variable d'être utilisée dans le programme, auquel cas notre code va provoquer des erreurs et donc entrainer le crash de notre application.
99% du temps nous allons déclarer une variable et lui assigner une valeur directement.

```
let maVariable = 'Hello World';
```

Plusieurs types de variable

Le dernier standard ECMAScript définit 8 types de données :
Sept types de données primitifs:

- Booléen (true / false)
- Null
- Undefined
- Number
- BigInt (proposition pour ES2020)
- String (chaine de caractère)
- Symbole (type introduit avec ECMAScript 6)
- Objet

Exercice :

Déclarer, initialiser et afficher en console plusieurs variables de chaque type.
(chaines de caractères, nombre, nombre a virgule, tableaux, objets)

Bonus : Faire une variable qui contient une fonction dans laquelle on fait un log console « Hello World »

Solution possible

https://github.com/jefff404/cours-js/tree/2-variables

```
/**
 * ************************************
 * 2-VARIABLES
 * ************************************
 */
//! On déclare une variable avec let ou const (ou var dans les anciennes versions de JS)
let maVariable;
//! On assigne une valeur à une variable avec le signe =
maVariable = 'Hello World';
console.log(maVariable);

//? Les types de variables (JS utilise un typage dynamique)
let uneString = 'MDR';
let unNombre = 11;
let unBooleen = true;
let unTableau = [1,2,3,'Hello'];
let unObjet = {
    propriete1 :  22,
    propriete2:'LOL'
};
//! Spoiler : on déclare une fonction comme ceci 👇
function testFunction (){
    console.log('Fonction de Test ?');
}
//? Bonus 💡 : on peut placer une fonction dans une variable
let uneFonction = function maFonction (){
    console.log('Fonction qui affiche HelloWorld');
}

console.log(uneString);
console.log(unNombre);
console.log(unBooleen);
console.log(unTableau);
console.log(unObjet);
console.log(uneFonction);
```

Nombres / Calculs /Modulo ☺/ Incrémenter / Assignement Composé

Bien entendu, une des utilités principales des langage de programmation c'est de pouvoir faire des calculs (+ ou - complexes) pour cela nous allons utiliser des opérateurs de calculs (+ , - , *, /) comme en Mathématiques.

Une autre technique très largement utilisée dans les applications web consiste à cumuler des chiffres.
Imaginons une variable qui nous sert de compteur de (vues, likes, lectures, etc…) que nous allons incrémenter (c'est-à-dire rajouter une valeur à ce compteur).

Pour cet exemple de compteur dans le cas ou l'on veut toujours ajouter 1

```
unCompteur = 0;
//* unCompteur + 1
unCompteur ++;
```

On peut également faire avec l'opérateur moins
```
unCompteur --;
```

Si l'on souhaite ajouter de 10 en 10
```
//* unCompteur = unCompteur + 10
unCompteur +=10;
```

Exercice Calculs - Nombres

Mettre en place un programme qui affiche en console le résultat de différents calculs (en utilisant tous les opérateurs de base et des nombres à virgule)
En plus faire un console log d'un calcul ultra complexe.

Mettre en place un compteur et utiliser tous les opérateur d'assignement composé.

Solution possible

https://github.com/jefff404/cours-js/tree/3-calculs

```
/**
 * ************************************
 * 3-CALCULS
 * ************************************
 */
console.log(42*675);
let unChiffre = 9;
let unNombre = 33;
console.log(unChiffre*unNombre);
console.log(2,9+1,3);
console.log(2.9+1.3);
console.log((1+1)-(2*3)/2);
console.log(10%2);
let unCompteur = 0;
console.log(unCompteur+1);
unCompteur = 0;
unCompteur = unCompteur+1;
console.log(unCompteur);
unCompteur = 0;
//* unCompteur = unCompteur + 1
unCompteur +=1;
console.log(unCompteur);
unCompteur = 0;
//* unCompteur + 1
unCompteur ++;
console.log(unCompteur);
//* unCompteur - 1
unCompteur --;
console.log(unCompteur);
//* unCompteur = unCompteur + 10
unCompteur +=10;
console.log(unCompteur);
//* unCompteur = unCompteur x 10
unCompteur *=10;
console.log(unCompteur);
//* unCompteur = unCompteur / 10
unCompteur /=10;
console.log(unCompteur);
//* unChiffre puissance 9
console.log(unChiffre**9);
```

Chaînes de caractères

En javascript nous pouvons tout aussi bien gérer des chaines de caractères, cela peut constituer juste un mot, une phrase ou tout un paragraphe par exemple pour cela ci-dessous nous allons voir les différentes syntaxes permettant de gérer des strings (chaines de caractères).

Nous allons donc voir les simple quote, les double quote, (guillemets simple ou double) ainsi que les littéraux de gabarits (ou template strings) qui facilite l'affiche d'une variable au sein d'un texte.

```
let bonjour = 'Bonjour';
```

```
let unMessage = "Bienvenue";
```

```
let welcome = `Bienvenue`;
```

Dans certains cas il peut être utile d'assembler plusieurs chaines de caractères, c'est le principe de la **concaténation**, pour cela on utilise l'opérateur + (à tester 🫠)

Exercice Phrase

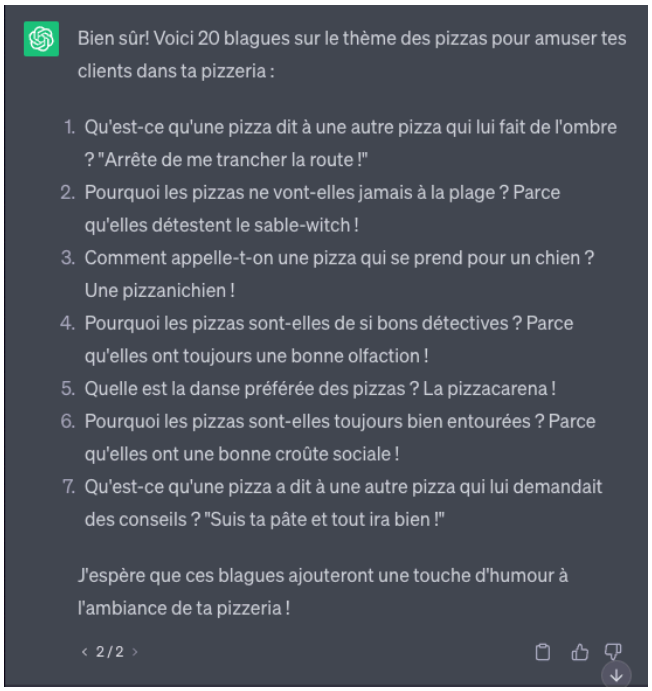
Le client, le restaurant **"**La Pizzeria Raffinata**"** (**le client insiste sur les guillemets**)nous a choisi pour réaliser son application mobile dans laquelle on peut directement commander en livraison.

Définissez selon vous toutes les variables pertinentes qui résume la commande d'un utilisateur chez **"**La Pizzeria Raffinata**"**

Vous devez faciliter le travail pour l'équipe du Template et ranger toutes ces variables dans une variable qui se nommera **SumUpOrderPhrase**, cette phrase devra contenir
(on utilise les variables précédentes pour former une phrase) :

Merci d'avoir commandé chez **"**La Pizzeria Raffinata**"**

Ps :
Le client nous a envoyé ce message ultérieurement



« Pour le message de la commande j'aimerais aussi que cela intègre soit le message 1 ou le 7 mais en respectant le saut de ligne.

Par avance Merci. »

Solution possible

https://github.com/jefff404/cours-js/tree/4-strings

```
/**
 * ************************************
 * 4-STRINGS
 * ************************************
 */
let bonjour = 'Bonjour';
let unMessage = "Bienvenue";
let welcome = `Bienvenue`;
console.log(bonjour,unMessage);
let unTexte = "Bonjour \"MR Gingle\"";
let unTxt = 'Aujourd\'hui';
console.log(unTexte,unTxt);
let concatenation = bonjour + " et " + unMessage +', il fait beau temps' + unTxt;
console.log(concatenation);
let uneTemplateString = `Hello ! ${bonjour} et ${unMessage}
on retourne à la "ligne" plus besoin des 'antislash'`;
console.log(uneTemplateString);
```

Les tableaux

Dans des applications qui gèrent plusieurs données il peut s'avérer judicieux de les ranger dans un tableau (dans une seule donnée on aura plusieurs informations).
Pour la syntaxe des tableaux nous assignons à une variable, des crochets [ ], c'est à l'intérieur de ces [ ] que l'on peut ranger des données (des strings, des numbers, des variables … bref tout type de données)

Ci-dessous un exemple d'un tableau qui contient des nombres :

```
let mesNombres = [100,200,300];
```

Les tableaux on cet aspect pratique qu'ils ont un système d'indexation (on peut accéder à chaque case du tableau) en utilisant cette syntaxe :

```
mesNombres[2]
```

Ci-dessus on accède au contenu de la case numéro 2 du tableau qui contient le nombre…300

(⚠️ le système d'indexation des cases d'un tableau commence à 0, donc la case numéro 0 est en fait la première case du tableau).

Exercice : Arrays

```
//! EXO 4 ARRAYS
//TODO: Créer 1 variable pour un nom,
//TODO: Créer une variable pour un âge,
//TODO: Créer une variable passions qui est un tableau qui contient 2 chaînes de caractères (au choix)
//TODO: Puis créer une variable tabUser qui est un tableau qui contient les variable du nom, de l'âge et passions
//TODO: en Console on affiche le tabUser
//TODO : en passant par tabUser on veut afficher en console uniquement les passions
//TODO : en passant par tabUser on veut afficher en console uniquement la 2ème passion
```

Solution possible

```
/**
 * ************************************
 * 5-ARRAYS
 * ************************************
 */
let prenom = 'JoSé';
let age = 45;
//! On déclare un tableau avec cette syntaxe : []
//! On peut placer ce que l'on veut dans un tableau
let unTableau = [12,'Salut ca va bien?',prenom,age];
console.log(unTableau);
console.log(unTableau[2]);
//! Exemple avec un tableau dans un tableau
let mesPassions = ["Boxe","Jonquilles"];
let monPerso = [prenom, age, mesPassions];
//! Avec le système d'index / indice on peut accèder
//! au contenu d'une case du tableau.
//! La 1ère case du tableau est à l'indice 0.
console.log(monPerso[2]);
console.log(monPerso[2][1]);
monPerso[0] = 23;
monPerso[1] = 'YOLO';
monPerso[2][1] = 'COOL';
console.log(monPerso);
//! On peut utiliser la propriété length,
//! .length sur le tableau en lui même
//! nous renvoi le nombre de case du tableau
console.log(monPerso.length);
//! .length sur une case précise du tableau
console.log(monPerso[2].length);
let mesNombres = [100,200,300];
```

Exercice array 2 les fonctions pour les tableaux

```
//!EXO 4.3 Ajout f° .push()
//TODO : créer un nouveau tableau qui contient des trucs
//TODO : allez se renseigner la f° push()
//TODO : utiliser push pour ajouter un nouveau truc au tableau
//TODO: On affiche en console ce tableau
```

Solution possible

(la fonction push( ),peut accepter plusieurs paramètres en les séparant par des virgule pour ajouter plusieurs données en utilisant une seule fois la fonction.

```
let testTabAjout = [10,10,10];
console.log('Avant Ajout :',testTabAjout);
testTabAjout.push('Cortex',92,'Les Pyramides');
console.log('Après Ajout : ',testTabAjout);
```

Pour les tableaux il existe également la fonction .pop(), qui va permettre de supprimer le dernier élément du tableau :

```
// pop pour suppr le dernier element du tableau
testTabAjout.pop();
console.log('suppression : ',testTabAjout);
```

```
Après Ajout :                                    app.js:37
▶ (6) [10, 10, 10, 'Cortex', 92, 'Les Pyramides']
suppression :                                    app.js:42
▶ (5) [10, 10, 10, 'Cortex', 92]
```

Exercice arrays 3

```
//! EXO 4.4 ARRAYS Récap
// TODO:  Créer 2 variables leNom et lePrénom
//TODO: Créer un tableau laPhrase et on y ajoute via (push)Le nom ,Le prénom Les initiales
//TODO: Afficher le tableau dans la console le nom le prénom et les initiales
```

Solution possible

```
let leNom = 'Garcia';
let lePrenom = 'José';
let initiales = lePrenom[0] + leNom[0];
let laPhrase = [];
laPhrase.push(leNom,lePrenom,initiales)
console.log(laPhrase);
```

Fonctions / Fonctions Fléchée / return / param / param par defaut / scope

Dans tous les langages de programmation on retrouve le concept de fonctions, il s'agit d'un bloc de code (un sous-programme dans notre programme si l'on veut) qui va contenir plusieurs instructions, de cette manière plutôt que d'écrire plusieurs fois certaines lignes de code, on va les regrouper au sein d'une fonction, de cette manière nous pourrons exécuter ailleurs dans le programme toutes les lignes de code de la fonction

Function

Syntaxe de base pour déclarer une fonction

```
function maSuperFonction(){
    console.log('Hello World');
    console.log(22+33);
}
```

Ensuite quelque part dans le programme il va falloir exécuter la fonction :

```
//! Détailler la fonction OK, mais ne pas oublier
//! d'exécuter au moins une fois dans le programme cette fonction
maSuperFonction();
```

Paramètre

Les fonctions ont aussi un concept de paramètre, dans le cas où les instructions au sein de la fonction ont besoin d'une variable extérieure. Ci-dessous une fonction qui va afficher en console une variable unNom

```
//! Certaines fonction ont besoin de prendre un paramètre ici num
//! Pas besoin de déclarer le paramètre, il sera défini à l'utilisation de
//! la fonction
function fonctionAvecParametre(num){
    console.log('Hello World');
    console.log(22+num);
}
//! Ici notre paramètre num aura pour valeur 9
fonctionAvecParametre(9);
```

La variable num est définie directement lors de l'utilisation de la fonction.

Exercice Fonction 1

```
//! EXO 5 : Function
// TODO : créer une fonction qui prend un nombre en paramètre
// TODO : La f° doit afficher en console: 33 + le nombre reçu en paramètre
// TODO : créer une autre fonction qui prend 2 nombres en paramètre
// TODO : Cette seconde f° doit afficher en console l'ADDITION des 2 nombres reçus en paramètre
```