TP's

JavaScript Module

ENI, Nicolas Hodicq

Version 1.2, 2018-05-15

Table des matières

Chapitre 1. JavaScript: les bases

1.1. Développer du JS dans une page web

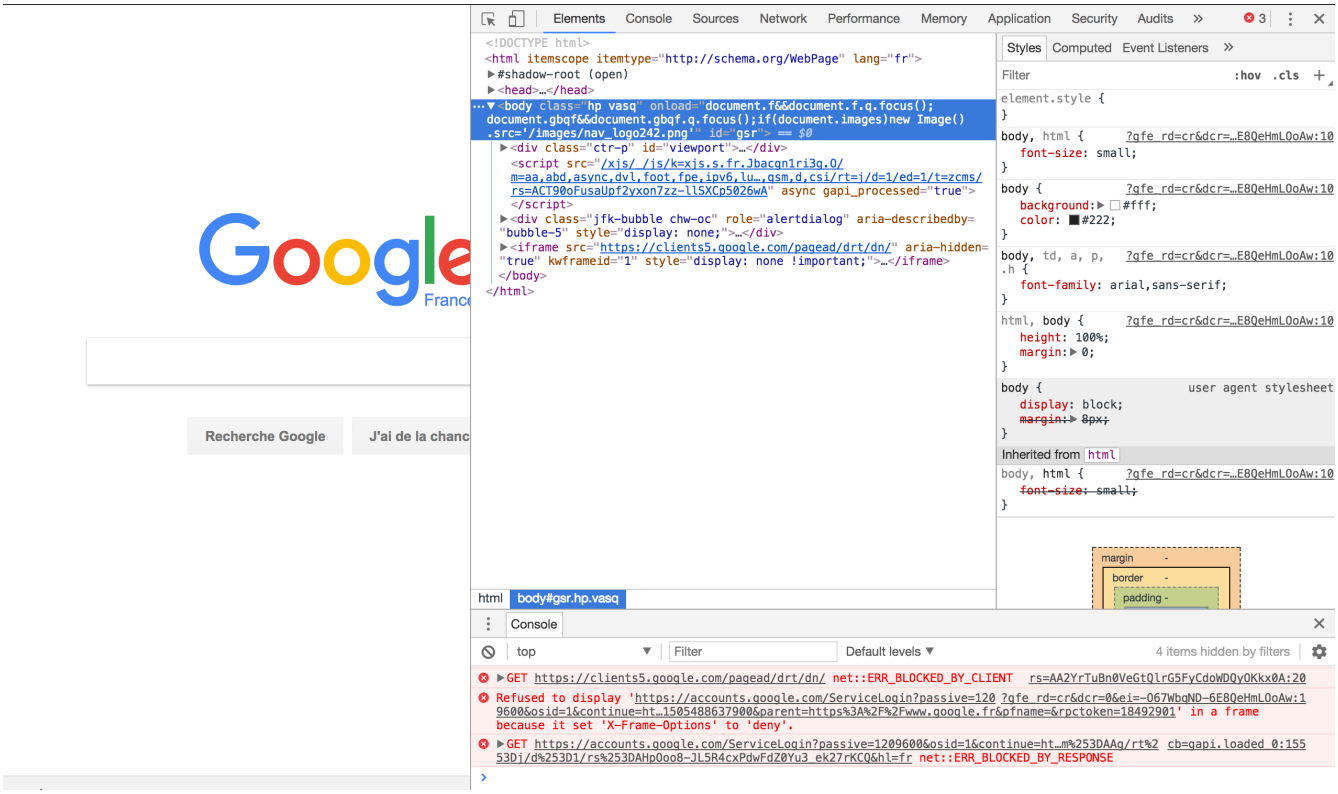
1.1.1. Outils de développement des navigateurs

Nous allons commencer par découvrir les outils de développement des navigateurs. Tous les navigateurs ont à ce jour des outils de développement, mais les fonctionnalités diffèrent en fonction de celui-sur lequel vous développez.

Les exemples que je vous fournis seront réalisés avec Chrome, je vous invite toutefois à tester les autres. Il est fréquent dans le développement web d'avoir des différences entre les navigateurs que ce soit au niveau CSS, JS, ou HTML. La bonne pratique est de toujours savoir avec quels navigateurs et quelles versions minimum nos applications devront-elles être compatibles.

Vous pouvez vous référer au site CanIUse pour connaitre la compabilité des API (HTML, JS, CSS) par navigateur.

- Ouvrez Chrome
- Sur la page d'accueil, ouvrez les outils de développements: MacOs (`Cmd+Alt+i`) et Windows (`F12`)
- Vous devriez obtenir un panneau de développement comme ceci:



- Faites un tour des outils, la console, les éléments du DOM, le Style et les sources

© 2018, ENI Services. Tous droits réservés.

1.1.2. Balise script

Nous allons déclarer une balise script dans une page HTML, afin de faire notre première ligne de JavaScript.

- Créer un fichier HTML

Sources: 1. samples/initial.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Base JavaScript</title>
</head>
<body>
    JavaScript Module: bases
</body>
</html>
```

- Dans le header, ajoutez une balise script comme ceci:

```
    <script> ①
        console.log('Hello World') ②
    </script>
```

① Balise script.

② Log dans la console la chaine de caractère "Hello Word".

- Vous devriez obtenir :

Sources: 2. samples/bases.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Base JavaScript</title>
    <script>
        console.log('Hello World')
    </script>
</head>
<body>
    JavaScript Module: bases
</body>
</html>
```

© 2018, ENI Services. Tous droits réservés.

- Ouvrez la page dans votre navigateur
- Ouvrez les outils de développement, et affichez la console. Vous devriez voir `Hello Word`

1.1.3. Externaliser dans un fichier JS

En pratique, nous allons plus souvent écrire le code JavaScript dans fichier externe afin d'avoir une meilleur architecture logicielle. Nous allons reprendre la page HTML précédemment créée.

- Créez un fichier `bases.js` dans le même dossier que la page HTML
- Dans le header, en dessous de la balise que vous avez créée précédemment, ajoutez une balise script, comme ceci :

```
    <script type="text/javascript" src="bases.js"></script>
```

- Vous devriez obtenir :

Sources: 3. samples/bases-external.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Base JavaScript</title>
    <script>
        console.log('Hello World')
    </script>
    <script type="text/javascript" src="bases.js"></script>
</head>
<body>
    JavaScript Module: bases
</body>
</html>
```

- Dans le fichier javascript que vous avez créé, ajoutez `console.log('External JacaScript file')`
- Rafaichissez la page dans votre navigateur
- Ouvrez les outils de développement, et affichez la console. Vous devriez voir `Hello Word` suivant de `External JacaScript file`

Tous les imports réalisés dans le header d'une page web sont chargés avant d'afficher le contenu de la page à l'utilisateur. C'est pourquoi dans le header, il est une bonne pratique de déclarer le moins d'import possible, idéalement aucun. Cela accroit la vitesse d'affichage de la page, l'expérience de l'utilisateur sera améliorée. Les imports doivent bien sûr être déclarés mais, en les positionnant en bas de fichier (avant la fin de la balise `body`).

- Créez un fichier `bases-latest.js.js` dans le même dossier que la page HTML

© 2018, ENI Services. Tous droits réservés.

- Avant la balise fermante body, `</body>`, ajoutez une balise script pour importer le fichier que vous venez de créer

```
    <script type="text/javascript" src="bases-latest.js"></script>
```

- Vous devriez obtenir :

Sources: 4. samples/bases-latest.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Base JavaScript</title>
    <script>
        console.log('Hello World')
    </script>
    <script type="text/javascript" src="bases.js"></script>
</head>
<body>
    JavaScript Module: bases
    <script type="text/javascript" src="bases-latest.js"></script>
</body>
</html>
```

- Dans le fichier javascript que vous avez créé, ajoutez `console.log('Latest external JacaScript file')`
- Rafaichissez la page dans votre navigateur
- Dans le fichier javascript que vous avez créé, ajoutez `console.log('Latest external JacaScript file')`
- Rafaichir la page dans votre navigateur
- Ouvrez les outils de développement, et affichez la console. Vous devriez voir dans l'ordre:
    - `Hello Word`
    - `External JacaScript file`
    - `Latest external JacaScript file`

© 2018, ENI Services. Tous droits réservés.

1.2. JavaScript: le langage

1.2.1. Les types

Vous pouvez vous aider de cette annexe : types

- Créez une page html et un fichier JavaScript.
- Déclarez une variable avec la valeur 10 et loguez celle-ci dans la console.

```
var maVariable = 10;
console.log('Ma variable : ', maVariable);
```

- Assignez `Formation module javascript` à la variable et loguez celle-ci dans la console.
- Assignez `undefined` à la variable et loguez celle-ci dans la console.
- Assignez `null` à la variable et loguez celle-ci dans la console.
- Assignez un objet avec une propriété `nom` et une valeur `Module JavaScript` à la variable et loguez celle-ci dans la console.

1.2.2. Opérateurs

Vous pouvez vous aider de cette annexe : operators

Comparaison

- Créez une page html et un fichier JavaScript.
- Créez 2 variables x et y

```
var x;
var y;
```

- Comparez l'égalité des variables

```
console.log('Égalité faible : ', x == y);
console.log('Égalité forte', x === y);
```

- En vous basant sur l'exemple précédent, refaites l'exercice avec les valeurs suivantes:
  - x = 2 et y = 'Formation'
  - x = 2 et y = '2'
  - x = undefined et y = null

© 2018, ENI Services. Tous droits réservés.

Addition string et number

- Créez une page html et un fichier JavaScript.
- Additionnez `20 + 5` et assignez le résultat dans une variable. Loguez le résultat
- Additionnez `20 + 'Formation'` et assignez le résultat dans une variable. Loguez le résultat
- Additionnez `20 + 5 + 'Formation'` et assignez le résultat dans une variable. Loguez le résultat
- Additionnez `20 + 'Formation' + 5` et assignez le résultat dans une variable. Loguez le résultat
- Additionnez `'Formation' + 20 + 5` et assignez le résultat dans une variable. Loguez le résultat

© 2018, ENI Services. Tous droits réservés.

1.2.3. Les fonctions

Vous pouvez vous aider de cette annexe : fonctions

Multipliy

- Créez une page html et un fichier JavaScript.
- Créez une fonction multiply qui prend en argument 2 paramètres et retourne le résultat de la multiplication de ceux-ci.
- Appelez cette fonction avec mes valeurs 10 et 3 et loguez le résultat.
- Loguez le nombre d'arguments en entrée de la fonction.
- Ajoutez lors de l'appel de la fonction une 3eme valeur, par exemple 7. Que constatez vous en regardant les logs ?
- Remplacer les 3 valeurs d'appel avec les valeurs suivantes: 2, 'Formation', 5. Que constatez vous ?
- Ajoutez des contrôles sur les arguments en entrée de la fonction. Nous souhaitons garantir que les paramètres en entrées ont des valeurs de type numérique afin que le multiplication soit possible. Il est possible de jeter des erreurs et de les attraper comme ci-dessous.

```
function throwError () {
    throw new Error('Messagae');
}

try {
    throwError();
} catch (e) {
    console.log(e);
}
```

Auto-invoquée

- Reprendre les fichiers de l'exercice précédent
- Ajouter à la suite une fonction auto-invoquée qui logue un messsage `'Auto-invoquée'`
- Rechargez votre page, que constatez vous dans les logs ?

Fonction en paramètre d'une fonction

En Javascript, une fonction est un type défini. Il est donc possible d'affecter une fonction à une variable et passer un argumant de type fonction à une fonction.

Voici un exemple:

© 2018, ENI Services. Tous droits réservés.

```
var logger = function (a) {
  console.log('1er argument :', a);
};

function firstCallBack(callback) {
  if (callback && typeof callback === 'function') {
    callback('mom premier callback');
  }
}

firstCallBack(logger);
```

Cette possibilité technique nous permet de passer une fonction qui sera exécuter plus tard. Le terme utilisé pour ses fonctions passées en argument est **callback**. Ce mécanisme est très répandu dans la programmation JavaScript, cela permet une première approche de développement de traitements asynchrones.

- Créez une page html et un fichier JavaScript.
- Créez une fonction `hello`

```
    var hello = function (name) {
        return 'Hello ' + name;
    };
```

- Créez une fonction `asyncWithCallback`

```
    // Pour déclencher un traitement, dans un temps donné, nous pouvons utiliser la m
éthode setTimeout
    // qui prend en argumant un callback et un temps en milliseconde ainsi que le
paramètre que l'on
    // souhaite translettre au callback
    setTimeout(function () {
        // la fonction sera exécutée dans 5000 millisecondes
    }, 5000)

    function asyncSayHello (name, callback) {

        if (callback && typeof callback === 'function') {
           setTimeout(callback, 5000, name);
        }
    }
```

- Appelez la fonction asyncWithCallback

© 2018, ENI Services. Tous droits réservés.

```
asyncSayHello('Niko', hello);
```

© 2018, ENI Services. Tous droits réservés.

1.2.4. Les objets

Vous pouvez vous aider de cette annexe : objets

Littéral

- Créez une page html et un fichier JavaScript.
- Nous allons créer un objet qui décrit une formation:
  - Nom
  - Langage
  - dateDebut
  - dateFin
  - stagiares

```
var formationJS = {
    nom: 'Module JavaScript',
    langage: 'JavaScript',
    dateDebut: '01/09/2017',
    dateFin: '05/09/2017',
    stagiaires : [{
            nom: 'Nom Exemple',
            prenom: 'Prenom Exemple'
        }
    ]
}
```

- Loguez le nom, le langage, les dates de début et de fin de la formation
- Ajoutez dynamiquement une propiété `formateur` avec la valeur suivante

```
{
    nom: Hodicq,
    prenom: Nicolas
}
```

- Loguez le nom du formateur de la formation
- ajoutez dans l'objet littéral une propriété `description` qui prend en valeur une fonction qui retourne une chaine de caractère concaténant le nom, les dates de début et de fin de la formation.
- Loguez le résultat de la fonction `description`
- Supprimez dynamiquement la propiété stagiaires et loguez l'objet

© 2018, ENI Services. Tous droits réservés.

Objet via Function

Il existe plusieurs méthodes pour instancier un objet, littéral, via l'objet Object ou bien via une function. Nous venons de manipuler la méthode litérale, nous allons maintenant aborder celle via les functions.

Par convention, les fonctions représentant des objets commencent par une lettre en majuscule.

- Créez une page html et un fichier JavaScript.
- Nous allons créer un objet qui décrit une formation:
  - Nom
  - Langage
  - dateDebut
  - dateFin
  - stagiares

```
function Formation () {
    this.nom = 'Module JavaScript';
    this.langage = 'javaScript';
    ...
}

var formation = new Formation();
```

- Loguez le nom, le langage, les dates de début et de fin de la formation.
- Nous allons rendre générique la création d'une formation

```
function Formation (nom, langage) {
    this.nom = nom;
    this.langage = langage;
    ...
}

var formation = new Formation('Module JavaScript', 'JavaScript');
```

- Loguez le nom, le langage, les dates de début et de fin de la formation.

Héritage

- Reprendre le TP précédent
- Nous allons créer 2 objets:

© 2018, ENI Services. Tous droits réservés.

- Personne (nom, prénom, age)
- Stagiaire (nom, prenom, age, connaissances)

Le stagiaire héritera de l'objet Personne.

- Instancier un stagiaire et loguez l'objet dans la console.
- Tester dans la console, l'instance de l'objet que vous avez créé avec `instanceof`

N'hésitez pas à regarder l'annexe : objets

© 2018, ENI Services. Tous droits réservés.

1.2.5. Les scopes

Vous pouvez vous aider de cette annexe : scopes

La notion de scope est surement la plus difficiile à appréhender lors qu'on vient d'autres langages, notamments les langages objets. Le scope est défini par fonction. Une variable définie dans une fonction ne sera accessible que dans cette fonction.

Le scope est l'ensemble des variables, des objets et des fonctions auxquels vous avez accès.

La notion du this est aussi différente des autres langages. Son comportement varie qu'on soit en mode strict ou non. Pour plus de détails sur le this, call, apply et bind, je vous laisse vous référer à cette documentation

Attention aux scopes !!

- Créez une page html et un fichier JavaScript.
- Créez une variable `nom` avec votre nom en valeur
- Créer une fonction `sayHello` comme suit:

```
    function sayHello() {
        var nom = 'Niko';
        console.log(nom);
    }
```

- Appelez la fonction sayHello
- Faites un console.log de nom après avoir appelé `sayHello`
- Vous devriez obtenir ceci:

```
var nom = 'Nom global';

function sayHello() {
  var nom = 'Niko';
  console.log('Nom : ',nom);
}

sayHello();
console.log('Nom : ', nom);
```

Vous pouvez constater que la variable nom, définie dans la méthode sayHello, n'affecte pas la valeur de la variable globale nom.

© 2018, ENI Services. Tous droits réservés.

- Créez un second ficher JavaScript
- Dans ce fichier, faites un console.log de nom. Que constatez vous ?
- Toujours dans ce second fichier, ajouter une autre fonction sayHello

```
function sayHello () {
  var nom = 'Niko 2';
  console.log('sayHello fichier js 2', nom);
}
```

- Changer la valeur de la variable nom sans la rédéfinir

```
nom = 'Nom global à changer';
```

- Enfin, appelez la méthode sayHello et loguez la variable nom

```
sayHello();

console.log('Nom : :', nom);
```

- Voici ce que vous devriez obtenir au final

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Scopes</title>
</head>
<body>
    Formation JavaScript: les scopes

    <script type="text/javascript" src="scopes-1.js"></script>
    <script type="text/javascript" src="scopes-2.js"></script>
</body>
</html>
```

© 2018, ENI Services. Tous droits réservés.

Sources: 5. Fichier 1

```
var nom = 'Nom global';

function sayHello() {
  var nom = 'Niko';
  console.log('Nom : ',nom);
}

sayHello();
console.log('Nom : ', nom);
```

Sources: 6. Fichier 2

```
console.log('Nom fichier js 2', nom);

function sayHello () {
  var nom = 'Niko 2';
  console.log('sayHello fichier js 2', nom);
}

nom = 'Nom global à changer';

sayHello();

console.log('Nom : :', nom);
```

- Vous devriez constatez que la variable globale nom est accessible dans le second fichier JS. Vous avez même réussi à changer sa valeur.
- Que se passe-t-il maintenant si nous changeons l'ordre d'import des scripts JS. Faites comme ceci:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Scopes</title>
</head>
<body>
    Formation JavaScript: les scopes

    <script type="text/javascript" src="scopes-2.js"></script>
    <script type="text/javascript" src="scopes-1.js"></script>
</body>
</html>
```

© 2018, ENI Services. Tous droits réservés.

- Dans la console, vous devez voir cette erreur normalement : `scopes-2.js:1 Uncaught ReferenceError: nom is not defined at scopes-2.js:1`.

Il faut être attentif, à l'ordre des imports en JavaScript. Vous ne pouvez pas utiliser, une fonction, un objet, ... avant qu'ils ne soient chargés.

En JavaScript, tous les fichiers js chargés dans une page web partagent le même espace. On l'appel le `Global Namespace`. Il est primordial de porter une attention particulière aux namespaces. En effet, vous pouvez changer, un objet de votre application, tout comme un objet d'une librairie que vous avez inclus dans votre application.

Namespaces

Il existe plusieurs techniques utilisées pour définir son namespace. Nous allons manipuler la création de namespace avec une fonction auto invoqué (Self Invoking Functions).

- Créez une page html et un fichier JavaScript.
- Créez une fonnction auto-invoquée

```
(function () {
  // code
})();
```

- Passez l'objet window en argument
- En vous appuyant sur l'exemple ci-dessous:
  - Définissez un namespace avec votre nom ou pseudo, il faut trouver un nom qui soit le plus possible unique
  - Créez une fonction qui récupère une liste de contacts (retourner un tablean codé en dur). un contact est défini par un nom et un prénom
  - Attachez cette fonction à votre namespace
  - Rendez votre namespace accessible de manière global en le déclarant en tant que propriété de l'objet window
  - Appelez votre fonction de récupération de contacts depuis un second fichier JS

© 2018, ENI Services. Tous droits réservés.

```
(function (window) {
  var LEGAL_AGE = 18;
  var beerService = {};         // Variable privée non accessible en dehors de la
fonction

  beerService.servingBeer = function (age) {
    age = age || LEGAL_AGE;
    if (age < LEGAL_AGE) {
      return;
    }

    // bloc de code
  };

  window.beerService = beerService;

})(window);

beerService.servingBeer(39);
```

© 2018, ENI Services. Tous droits réservés.

1.2.6. Les closures

Uniquement pour ceux qui veulent aller plus loin: Closures

© 2018, ENI Services. Tous droits réservés.

1.3. JavaScript: Interagir avec le DOM

1.3.1. Document

La principale, en tout cas la première historiquement, utilisation du JavaScript est de permettre la manipulation du DOM (Document Object Model). Pour cela, l'objet `document` est disponible dans le namespace globale quand le JavaScript est exécuté dans une page web.

Cet objet via son API permet d'interagir avec les différents éléments, paramètres de la page courante.

Changer le titre de la page

- Créez une page html, définissez un titre dans le header et ajoutez du contenu comme dans l'exemple ce-dessous :

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title to change</title>
    <link rel="stylesheet"
          href="document.css">
</head>
<body>
    <h1>
        Formation JavaScript: DOM-document
    </h1>

    <h1 id="title"
        class="text-red">
        Formation JavaScript: DOM-document
    </h1>

    <p class="text-red">
        Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis
nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis
aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat
nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui
officia deserunt mollit anim id est laborum.
    </p>

    <script type="text/javascript" src="document.js"></script>
</body>
</html>
```

© 2018, ENI Services. Tous droits réservés.

- ajouter un fichier CSS et définir les classes suivantes :

```
.text-red {
    color: red;
}
```

- Créez un fichier JavaScript, loguer le titre de la page avec la méthode `document.title`
- Changer le titre `document.title = 'une nouvelle valeur'`. Vous devriez voir le nom de la page dans le navigateur
- Voici ce que vous devriez obtenir :

```
var app = {
  init: function () {
    console.log('Document title: ', document.title);
    document.title = 'Nouveau titre';
    console.log('Document title: ', document.title);
  }
};

app.init();
```

Récupérer un noeud du DOM

L'objet document permet de récupérer un ou plusieurs éléments du DOM que ce soit par le nom de balise, une classe ou un ID. Le résultat est toujours un tableau d'éléments car le critère de recherche peut correspondre à 0 ou n éléments.

par un tag (document.getElementsByTagName)

- Récuperez et loguez le ou les éléments correspondant à la balise `h1`

par une classe (document.getElementsByClassName)

- Récuperez et loguez le ou les éléments correspondant à la classe `text-red`

par son ID (document.getElementById)

- Récuperez et loguez le ou les éléments correspondant à l'id `title`

1.3.2. Les événements

Un événement permet de déclencher une fonction.

Chaque élément du DOM peut déclencher des événements, certains sont disponibles pour tous les éléments, d'autres sont spécifiques. Par exemple l'événement `click` sera transverse et `submit` ne

© 2018, ENI Services. Tous droits réservés.

sera disponible que sur une balise form.

Il existe 2 possibilités pour définir une fonction sur un événement:

- HTML first

```
<h1 id="title-2"
    onclick="alert('Hello title-2')"
    class="text-red">
    Formation JavaScript: DOM-events
</h1>
```

- JavaScript first

```
    <h1 id="title-1"
        class="text-red">
        Formation JavaScript: DOM-events
    </h1>
```

```
var title1  = document.getElementById('title-1');

// Choix 1
title1.addEventListener('click', function () {
  alert('Hello title-1');
})

// Choix 2
title1.onclick = function () {
  alert('Hello title 1');
}
```

Vous pouvez remarquer qu'il y a une légère différence dans le nom de l'événement. En HTML et dans le choix 2, il vous faut déclarer `on + <NOM_DE_L_EVENT>` alors qu'en JavaScript choix 1 c'est uniquement le `NOM_DE_L_EVENT`.

Dans la réalité, vous trouverez souvent un mix de ces approches dans une application.

- Nous avons également la possiblité de récupérer un objet `event` dans la fonction de callback.

```
title1.addEventListener('click', function (e) {
  console.log(e);
  alert('Hello title-1');
})
```

© 2018, ENI Services. Tous droits réservés.

- En vous basant sur l'exemple suivant :

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Events</title>
    <link rel="stylesheet"
          href="events.css">
</head>
<body onload="app.init()">
    <h1 id="title-1">
        Formation JavaScript: DOM-events
    </h1>

    <h1 id="title-2"
        class="text-red">
        Formation JavaScript: DOM-events
    </h1>

    <p class="text-red">
        Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis
nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis
aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat
nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui
officia deserunt mollit anim id est laborum.
    </p>

<script type="text/javascript" src="events.js"></script>
</body>
</html>
```

- Mettez des écouteurs 'mouseover' et 'mouseout' sur l'élément du DOM `<p>`.
- Ces événements sont de type MouseEvent, vous pouvez donc y trouver les propriétés x et y. Loguez ces valeurs pour chacun des événements.
- Dans une page web vous avez aussi accès à l'objet `window` qui met à disposition une propriété `innerWidth`. Cette propriété vous donne la taille du viewport.
    - Avec cette donnée et l'événement que vous récupérez dans le callback `mouseover`, déterminez si la souris survole le coté gauche ou droit du texte. Affichez GAUCHE ou DROITE dans la console.
    - Si on augmente ou si on réduit la taille de la fenètre, le traitement doit être valide sans recharger la page.

© 2018, ENI Services. Tous droits réservés.

1.3.3. Modifier le DOM

Nous savons maintenant comment récupérer des éléments du DOM, comment réagir aux événements. Il nous reste à être capable de modifier le DOM. Cela signifie, modifier, ajouter ou supprimer un élément.

Modifier un élément

- Nous allons continer l'exercice précédent.

A l'aide de la propriété `classList` d'un élément, vous pouvez ajouter, supprimer, tester si une classe CSS est présente.

- Voici les rèles de gestion à implémenter:
  - Quand le texte est survolé à gauche, supprimer la classe (texte en noir)
  - Quand le texte est survolé à droite, la classe doit être présente (texte en rouge)
  - Quand le texte n'est pas survolé, la classe doit être présente (texte en rouge)

Ajouter un élément

- Nous allons continer l'exercice précédent.
- Pour ajouter un élément, l'objet `document` met à disposition une propriété `createElement`

```
var p = document.createElement("p"); ①
document.body.appendChild(p); ②
```

① Création d'un élément de paragraphe.

② Ajout de l'élément au body du document

- Au click sur le titre possédant l'id `title-1` ajouter un élément de paragraphe avec la classe CSS `text-red`

Pour ajouter du texte à un élément pagraphe, la propriété de l'élément est `innnerHtml`.

- Si vous cliquez plusieurs fois, un élément sera ajouté à chaque click.
- Ajoutez maintenant le comportement du TP précédent sur les events (mouseover et mouseout) à chaque nouvel élément créé.

Supprimer un élément

- Nous allons continer l'exercice précédent.
- Ajouter un bouton dans la page pour supprimer les paragraphes

© 2018, ENI Services. Tous droits réservés.

- Créez une fonction en javascript qui sera appelée sur le `onclick` du button.
- Récupérez tous les paragraphes de la page et supprimez les.

© 2018, ENI Services. Tous droits réservés.

1.4. JavaScript: Les appels réseaux

1.4.1. XMLHttpRequest

Il est souvent nécessaire pour une application cliente (front-end) de devoir récupérer des données sur un serveur afin les afficher dans la page, ou bien mettre à jour une donnée dans le système d'information.

Aujourd'hui Les appels réseaux se réalisent avec l'objet standartisé XMLHttpRequest. Les différents verbes HTTP (OPTIONS, GET, POST, PUT, DELETE, ...) sont supportés, ainsi que différent protocole, par exemple **file**.

Voici un exemple d'appel HTTP GET issue de la documentation :

```
    // Ici, la requête sera émise de façon synchrone.
    const req = new XMLHttpRequest();
    req.open('GET', 'http://www.mozilla.org/', false);
    req.send(null);

    if (req.status === 200) {
        console.log("Réponse reçue: %s", req.responseText);
    } else {
        console.log("Status de la réponse: %d (%s)", req.status, req.statusText);
    }

    // Ici, la requête sera émise de façon asynchrone.
    const req = new XMLHttpRequest();

    req.onreadystatechange = function(event) {
        // XMLHttpRequest.DONE === 4
        if (this.readyState === XMLHttpRequest.DONE) {
            if (this.status === 200) {
                console.log("Réponse reçu: %s", this.responseText);
            } else {
                console.log("Status de la réponse: %d (%s)", this.status, this
.statusText);
            }
        }
    };

    req.open('GET', 'http://www.mozilla.org/', true);
    req.send(null);
```

Dans une application cliente, il faut toujours privilégier une approche asynchrone. L'interface n'est pas bloquée en attendant la réponse d'un traitement.

© 2018, ENI Services. Tous droits réservés.

- Créez une page html. Au niveau JavaScript, je vous laisse créer le ou les fichiers nécessaires. L'objectif est de commencer à structurer le code de manière pérenne et de mettre en oeuvre tout ce que nous avons déjà vu. Pensez au découpage des fichiers !!
- Ajouter un conteneur de type `div` avec un id
- Au chargement de la page, faites un appel réseau vers l'API User de Github. Vous allez récupérer les détails de votre compte github. Si vous n'en avez pas, il vous faut en créer un, cela vous servira plus tard dans la formation. au format JSON.

A l'aide de l'objet JSON, vous pouvez transformer vos données reçues en objet JS, par exemple `JSON.parse(this.responseText)`

- Voici l'URL a appelé `https://api.github.com/users/VOTRE_USER`
- Voici les données reçues avec mon user

© 2018, ENI Services. Tous droits réservés.

```
{
    login: "nartawak",
    id: 2499853,
    avatar_url: "https://avatars0.githubusercontent.com/u/2499853?v=4",
    gravatar_id: "",
    url: "https://api.github.com/users/nartawak",
    html_url: "https://github.com/nartawak",
    followers_url: "https://api.github.com/users/nartawak/followers",
    following_url: "https://api.github.com/users/nartawak/following{/other_user}",
    gists_url: "https://api.github.com/users/nartawak/gists{/gist_id}",
    starred_url: "https://api.github.com/users/nartawak/starred{/owner}{/repo}",
    subscriptions_url: "https://api.github.com/users/nartawak/subscriptions",
    organizations_url: "https://api.github.com/users/nartawak/orgs",
    repos_url: "https://api.github.com/users/nartawak/repos",
    events_url: "https://api.github.com/users/nartawak/events{/privacy}",
    received_events_url: "https://api.github.com/users/nartawak/received_events",
    type: "User",
    site_admin: false,
    name: "Nicolas Hodicq",
    company: "@bewizyu ",
    blog: "",
    location: "Lyon",
    email: null,
    hireable: null,
    bio: "Full stack web & mobile developer",
    public_repos: 11,
    public_gists: 0,
    followers: 7,
    following: 13,
    created_at: "2012-10-06T10:53:46Z",
    updated_at: "2017-08-30T10:34:58Z"
}
```


- Affichez dans la page (avec un minimum de design qd même), l'image de votre profil, votre pseudo, et votre nom

1.4.2. Pour aller plus loin ...

- Reprenez le TP précédent.
- Créez une nouvelle page HTML
- Au click sur l'image de profil, naviguer vers la seconde page.
- Nous allons afficher dans cette seconde page, la liste des repositories du user Github. Pour ce faire, récupérez la propriété `login` et faites la passer en paramètre à votre seconde page (paramètre dans l'URL).

© 2018, ENI Services. Tous droits réservés.

Pour changer de page, utilisez `window.location`.

```
// Changer de page
window.location = 'mapage.html'

// Récupération d'un paramètre dans une URL
var url = new URL(window.location);
var name = url.searchParams.get('name');
```

- Dans la seconde page, récupérez le paramètre et faites l'appel réseaux pour fetch les données nécessaires.

```
// Par exemple
'https://api.github.com/users/' + name + '/repos'
```

- Afficher la liste des repositories en mettant le nom du repo et description si il y en a une.

© 2018, ENI Services. Tous droits réservés.

Chapitre 2. Node.js

2.1. Installation

Node.js en version 8+ doit être installé sur la machine locale. Pour cela, il existe 2 façons de l'installer :

2.1.1. Site officiel

- téléchargez la version correspondant à votre système sur le Site officiel

2.1.2. Node Version Manager

Installer un version manager permet d'avoir une ou n version de Node.js différentes sur la machine.

Mac

- Installation via HomeBrew

```
brew install nvm
```

- Ajoutez ensuite à votre profile (~/.bash_profile, ~/.zshrc, ~/.profile, or ~/.bashrc), le contexte pour nvm

```
export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && . "$NVM_DIR/nvm.sh" # This loads nvm
```

- Vous pouvez ensuite voir toutes les versions de node.js disponibles. Installez la version v8.1.0

```
nvm ls-remote

nvm install v8.1.0

nvm ls
```

Windows

- Installation de Nodist
- Page de release

© 2018, ENI Services. Tous droits réservés.

2.1.3. Test de l'installation

- Ouvrez un terminal et lancez les commandes suivantes pour vérifier les versions :

```
node -v

npm -v
```

© 2018, ENI Services. Tous droits réservés.

2.2. Node.js: les bases

2.2.1. Premier script

Run JavaScript

- Créer un fichier index.js
- Écrire dans la console `Formation JavaScript Node.js`
- Dans un terminal, jouer le script avec node

```
node index.js
```

Debug JavaScript

- Reprendre le même fichier que pour le TP précédent. Chrome doit être installé et démarré sur la machine.
- Installer l'extension suivante dans chrome Noeud Inspector Manager

```
node --inspect index.js
node --inspect-brk index.js
```

- Qu'observe-t-on ?

© 2018, ENI Services. Tous droits réservés.

2.2.2. Module loader: CommonJS

Dans chaque fichier Javascript interprété par Node.js, certaines variables sont accessibles dans le scope. Le code que vous avez produit dans un fichier est inséré dans une fonction qui vous amène des arguments. Cela permet entre autre, comme nous l'avons vu précédemment d'isoler notre code du namespace global.

Si nous lançons la commande que vous venez de jouer

```
node --index-brk index.js
```

Dans le navigateur, vous pouvez voir ce code :

```
(function (exports, require, module, __filename, __dirname)
    { console.log('Formation JavaScript Node.js');
});
```

Nous pouvons constater que notre code est bien isolé dans une fonction, pour lui définir un scope propre. Je vais vous décrire les différents paramètres.

exports

Exports est une référence à module.exports. Il existe certaines différences importantes que vous pouvez retrouver dans la documentation.

Je vous conseille de privilégier `module.exports` dans les TP's.

require

Require permet de charger les modules. Ces modules sont chargés de manière synchrone.

module

Cet objet est une référence au module courant. `module.export` est utilisé pour définir ce que le module exporte. Dit autrement, quels sont les objets, ou function ou variables ou ... que le module courant met à disposition des autres modules.

__filename

C'est le module courant. C'est le fichier JavaScript.

© 2018, ENI Services. Tous droits réservés.

__dirname

Répertoire du module courant. C'est le répertoire du fichier JavaScript.

- Voici un module speak.js qui possèdent une fonction sayHello et qui l'exporte :

```
// Nom du fichier speak.js

function sayHello (name) {
  console.log('Hello ' + name);
}

module.exports = sayHello;
```

- et maintenant nous l'importons dans un fichier index.js

```
var speak = require('./speak');

speak('Niko');
```

- En jouant la commande suivante `node index.js`, j'obtiens `Hello Niko` dans la terminal
- Une meilleure approche serait de se dire que lorsque notre module "parle", nous voulons pas forcément qu'il ne sache que dire Hello. On voudrait qu'il sache dire Bye aussi. Nous allons donc l'implémenter et devoir changer l'export.

```
function sayHello (name) {
  console.log('Hello ' + name);
}

function sayBye(name) {
  console.log('Bye ' + name);
}

module.exports = {
  sayHello: sayHello,
  sayBye: sayBye
};
```

```
var speak = require('./speak');

speak.sayHello('Niko');
speak.sayBye('Niko');
```

© 2018, ENI Services. Tous droits réservés.

Module math.js

- Créez un module Math.js
- Créez et exportez les fonctions suivantes:
  - addition
  - soustraction
  - multiplication
  - division
- Créez un module index.js
- Appelez toutes les fonctions et loguez le résultat de chacune

Package.json

Le package.json est la carte d'indentité d'un projet Node.js. Nous allons donc créer celui pour votre projet.

- Dans un terminal, lancer la commande `npm init -y`
- Voici le package.json généré:

```
{
  "name": "modules",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

N'hésitez pas à consulter la description complète de la documentation du package.json

- Ajoutez une propriété `start` avec la valeur `node index.js`.

```
"scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node index.js"
  },
```

© 2018, ENI Services. Tous droits réservés.

- Dans le terminal, lancer la commande `npm run start` ou `npm start`. La balise script du package.json permet de définir des raccourcis.
- au fur et à mesure des TP's nous ajouterons les configurations nécessaires

Module externe ou dépendance

Un des avantages du développement en JavaScript avec Node.js est NPM (Node Package Manager). NPM et son registry mettent à disposition un nombre très important de dépendances. Pour chercher une dépendance, vous pouvez soit le faire en ligne de commande `npm search lodash` ou vous rendre sur le site npmjs.com.

Le modules peuvent être local ou global.

local

Un module local est installé dans le dossier node_modules du projet. Celui-ci est crée si il n'est pas déjà existant. Nous ne pourrons utiliser ce module que dans le cadre de notre projet.

global

Un module global est installé au niveau de Node.js. Il est possible de l'utiliser directement en ligne de commande une fois installé. Tous les modules ne peuvent pas être installé en global. Ils doivent définir une propriété bin dans leur package.json

local

Nous allons installé un module utilitaire qui fait parti des références à connaitre en JavaScript et l'utilisez dans le projet précédent.

lodash est une librairie qui permet de bénéficier de fonctions pour les objets, les collections, les tableaux. Elle permet aussi une écriture très fonctionnelle mais c'est un autre sujet.

- Reprendre le TP précédent
- Ajouter la dépendance au projet `npm install lodash --save`. Le paramètre save permet de l'ajouter dans le package.json
- Vous devriez voir:
  - Un dossier node_modules avec `lodash` dedans
  - Votre package.json doit avoir une nouvelle propriété `dependencies` avec lodash et sa version.
- Utilisons lodash dans l'application. Voici un exemple basé sur le sayHello que j'ai utilisé tout à l'heure

© 2018, ENI Services. Tous droits réservés.

```
const _ = require('lodash');

function sayHello (name) {
  if (_.isString(name) && !_.isEmpty(name)) {
    console.log('Hello ' + name);
  }
}
```

- Utilisez lodash dans votre fichier math.js afin de vérifier ques tous les paramètres de fonction sont bien des numbers

global

Nous allons commencer à tester notre code métier. Mocha est une librairie fortement utilisée pour faire des tests en node.js.

Beaucoup d'autre librairies de test existent. Cela dépendra des choix des équipes de développement sur un projet.

- Installez mocha en global sur la machine `npm install -g mocha`

Pour installer, supprimer des modules globaux, il faut ajouter le paramètre `-g` ou `--global` à la commande node.

- Vérifiez que vous avez bien accès à mocha

```
mocha --help
```

- Créons notre premier test
- Dans votre projet, créez un dossier test avec un fichier math.spec.js
- Ajoutez dans votre fichier ce bout de code issu de la documentation

```
var assert = require('assert'); ①

describe('Array', function() {  ②
  describe('#indexOf()', function() {
    it('should return -1 when the value is not present', function() {   ③
      assert.equal(-1, [1,2,3].indexOf(4));     ④
    });
  });
});
```

① import de l'API assert présente par défaut dans Node.js

© 2018, ENI Services. Tous droits réservés.

② describe permet de définir une section de tests.

③ it permet de définir un test

④ assert pour test notre traitement

L'API assert de Node.js est suffisante pour des tests simples. Nous préférons généralement utiliser des outils nous permettant d'avoir une API plus complète. Je vous conseille donc d'écrire vos tests avec chai

```
expect(function () {}).to.not.throw();
expect({a: 1}).to.not.have.property('b');
expect([1, 2]).to.be.an('array').that.does.not.include(3);
```

-Installer chai en dépendance de développement

```
npm install chai --save-dev
```

- Dans votre terminal, lancez la commande `mocha`, pour voir les résultats des tests

```
modules git:(formation-modis)    mocha


  Array
    #indexOf()
      ✓ should return -1 when the value is not present
```

- En vous basant sur cet exemple et la documentation de chai, réalisez les tests de vos fonctions dans le fichier math.js

Modules globaux en local

Vous avez installé mocha en global, c'est très pratique. Il existe cependant, une approche plus fine. Quelle est la problématique ? Mocha doit être installé en global sur la machine pour réaliser les tests. L'application n'est pas 'autonome' avec ses dépendances. Le second problème, la version de mocha installée sur la machine n'est pas forcément celle pour utilisée pour écrire les tests.

Il est possible avec Node.js d'installer le module en dépendance locale, dans le dossier node_modules. Les dépendances définissant un bin dans leur package.json auront leur lanceur dans le dossier `node_modules/.bin`

- Supprimez mocha installé en global `npm uninstall -g mocha`
- Installez mocha en tant que dépendance local `npm install mocha --save-dev`

© 2018, ENI Services. Tous droits réservés.

Nous avons vu précédemment le --save, `--save-dev` installe la dépendance en locale et inscrit la dépendance dans le package.json, mais en dépendance de développement. Nous ne les installerons pas en production.

- Pour lancer les tests

```
node_modules/.bin/mocha
```

- Nous pouvons faire un raccouci dans le package.json

```
"scripts": {
    "test": "mocha",
    "start": "node index.js"
  },
```

Vous pouvez remarquer que nous ne précisons pas 'node_modules/.bin'. Node.js sait aller chercher le module dans le .bin

© 2018, ENI Services. Tous droits réservés.

2.3. Node.js: TP ⇒ pub-services

Dans ce TP nous allons créer deux modules nodes. Le premier portera le métier de notre application, et le second sera un module global qui nous mettra a disposition des fonctions en lignes de commandes.

Le périmètre fonctionnel est simple, Nous allons réaliser deux services, qui donne une liste de pub et un autre qui nous donne ceux ouverts à la date courante.

2.3.1. pub-services

Création d'un repository git

Pour ce projet nous allons utiliser un repository git. Nous nous servirons de ce projet tout au long de la fin de formation.

- Créez un repo github que vous initialisez avec un README.md.
- Clonez le repo et positionnez vous dedans.

Initialisation du projet pub-services

- Créez une branche pub-services sur git
- Créez un dossier pub-services
- Initialisez le package.json
- Voici un exemple de ce que vous pourriez faire :

```
+--test
|   + ...
+--services
|   +-pubs.services.js  // Les services sont asynchrones (callback)
+--mocks
|   +-pubs.json
+--index.js             // Fichier de mapping des services
+--package.json
```

Jeux de données

Nous allons utiliser des données mockées. Cela veux dire que vous allez générer votre propre jeux de données. Voici un template :

© 2018, ENI Services. Tous droits réservés.

```
[
  {
    "name": "Arawak",
    "owner": {
      "firstName": "Nicolas",
      "lastName": "Hodicq",
      "mail": "nhodicq@bewizyu.com"
    },
    "openDays": [
      "Tuesday",
      "Wednesday",
      "Thursday",
      "Friday",
      "Saturday"
    ],
    "openHours": {
      "start": 10,
      "end": 1
    },
    "beers": [
      {
        "type": "Blonde",
        "name": "Triple Karmeliet"
      }
    ]
  }
]
```

- Créez un fichier pub.json avec vos données

Services métiers

- Dans le fichier pub-services.js, créez deux fonctions que vous exporterez, une qui liste tous les pub et une qui liste les pub ouverts en se basant sur les jours. Vous utiliserez le jeux de données mockées

Vous pouvez regarder la librairie moments.js pour la manipulation de date.

Fichier de mapping

- Dans le fichier index.js qui sera le fichier importé dans notre prochain module node, il vous faut mapper et exporter tous les services que vous souhaitez mettre à disposition des modules qui installeront ce module en dépendance.

Par exemple:

© 2018, ENI Services. Tous droits réservés.

```
var pubService = require('./services/pubs.service');

module.exports = {
  services: {
    pubService : pubService
  }
}
```

- Poussez votre branche git sur le serveur distant (github) et faites une pull request que vous pouvez merger après.
- Mettez vous sur la branche master et faire un pull de la branche master distante

2.3.2. pub-cli

Création d'un module node qui peut être installé en global. Ce module exposera via un client shell, 2 options: - Lister les pubs - Lister les pubs ouverts

```
pub-cli --list
pub-cli --opened
```

Initialisation du projet pub-cli

- Créez une branche pub-cli sur git
- Créez un dossier pub-cli
- Initialisez le package.json
- Voici un exemple de ce que vous pourriez faire :

```
+--bin
|   +-index.js          // Parse les arguments passés à la commande shell
+--src
|   +-main.js         // Appel les services métier des pubs
+--package.json
```

- Ajoutez en dépendance le projet pub-services précédemment créé. Dans le package.json

```
"dependencies": {
    "pub-services": "../pub-services"
  }
```

```
npm install
```

© 2018, ENI Services. Tous droits réservés.

Parser les arguments

- Pour parser les arguments, ajoutez la librairie commander
- Voici un exemple du fichier bin/index.js

```
#!/usr/bin/env node
var program = require('commander');

program
  .version('1.0.0')
  .option('-l, --list', 'Get the pubs list')
  .parse(process.argv);

if (program.list) {
  require('../src/main').getListPub();
}
```

Installer notre module

- Vous devez ajouter les propiétés suivantes dans le package.json

```
"bin": {
    "pub-cli": "bin/index.js"
  },
  "preferGlobal": "true",
```

- Pour installer votre module courant globalement sur la machine et ainsi faciliter vos tests `npm install -g .`
- Le nom dans la package.json est le nom de l'exécutable dans le terminal

```
pub-cli --help
```

- Lorsque votre application fonctionne totalement
  - Poussez votre branche git sur le serveur distant (github) et faites une pull request que vous pouvez merger après.
  - Mettez vous sur la branche master et faire un pull de la branche master distante

2.3.3. pub-rest-api

Création d'un module permettant de lancer un serveur web via express.js.

- Créez une branche pub-rest-api sur git
- Créez un dossier pub-rest-api

© 2018, ENI Services. Tous droits réservés.

- Initialisez le package.json
- Vous devrez ajouter en dépendances le projet pub-services précédemment créé.

Structure du projet :

```
+--index.js             // Serveur express
+--package.json
```

- Exemple de code pour lancer un serveur express :

```
const express = require('express');
const pubService = require('pub-services').services.pubService;
const app = express();

app.get('/pubs', function (req, res) {
  const pubs = pubService......
  res.status(200).json();
})

app.listen(3000);
```

- Démarrez le serveur web

```
node index.js
```

© 2018, ENI Services. Tous droits réservés.

Chapitre 3. JavaScript: ES6+

3.1. pub-services

Nous allons reprendre le projet pub-services et le passer en version ES6.

Le système de chargement de module est celui que vous avez déjà manipuler. La version que je vous ai présenté dans les slides n'est pas encore implémenté dans Node.js. Nous pouvons l'utiliser par contre lorsque l'on utilise des transpileurs tel que Babel et Traceur. Nous reviendrons sur ce point dès lors que nous développerons sur des applications clientes.

3.1.1. Modèliser le modèle avec des classes

- Pour rappel, un objet pub est définit comme suit:

```
[
  {
    "name": "Arawak",
    "owner": {
      "firstName": "Nicolas",
      "lastName": "Hodicq",
      "mail": "nhodicq@bewizyu.com"
    },
    "openDays": [
      "Tuesday",
      "Wednesday",
      "Thursday",
      "Friday",
      "Saturday"
    ],
    "openHours": {
      "start": 10,
      "end": 1
    },
    "beers": [
      {
        "type": "Blonde",
        "name": "Triple Karmeliet"
      }
    ]
  }
]
```

- créez les classes pour modéliser cette structure de données et instanciez les objets lors de la

© 2018, ENI Services. Tous droits réservés.

récupération des données mockées.

ES6+ syntaxe

- destructuring, object shorthand, promises, ... faites passer le code pub-services et pub-cli à la sauce ES6+

© 2018, ENI Services. Tous droits réservés.

3.2. Promises

Vous allez manipuler les promises et les chaines de promises via l'API fs et la dépendance fs-extra

- Créez un nouveau projet Node.js avec un package.json et un fichier index.js
- Ajoutez la dépendance `fs-extra`
- Voici les règles de gestions du traitement:
  - Au démarrage de votre traitement, vérifiez que vous avez ou pas un dossier `temp` à la racine de votre projet
  - Si vous en avez un supprimez le
  - Ensuite créez le dossier temp
  - Créez un fichier `pubs.json` en insérant un jeu de données mockées. N'hesitez pas à reprendre les données que vous avez déjà créées dans les TP's précédents
  - Mettez un watcher sur le fichier, afin que lorsque vous sauvegardez le fichier après l'avoir édité, vous ayez une notification dans la console
  - Tous les traitements doivent être effectué de manière asynchrone

© 2018, ENI Services. Tous droits réservés.

Chapitre 4. Annexes

4.1. JavaScript Types

```
// ******************************** //
// ********* Variable ************* //
// ******************************** //

/**
 * déclaration d'une variable
 */
var premiereVariable;
premiereVariable = 1;

// une variable déclarée sans le mot clé var est une variable globale, accessible
depuis tous le namespace global

/**
 * Déclaration de plusieurs variables
 */
var premiereVariable, secondeVariable;
premiereVariable = 1;
secondeVariable = 'Formation JS';

/**
 * Variable dynamique
 */
var maVariable;
console.log(maVariable);        // result: undefined
maVariable = 1;
console.log(maVariable);        // result: 1
maVariable = .1;
console.log(maVariable);        // result: .1
maVariable = 'test';
console.log(maVariable);        // result: test
maVariable = false;
console.log(maVariable);        // result: false

/**
 * valeur par défaut d'une variable
 */
var option = option || {name: 'default'};

// ******************************** //
// *********** Types ************** //
// ******************************** //
```

© 2018, ENI Services. Tous droits réservés.

```
// ****** Primitives data types ***** //

/**
 * Number
 */
10;                     // Type: Number
10.05;                  // Type: Number
5e2;                    // Type: Number

/**
 * Boolean
 */
true;                   // Type: Boolean
false;                  // Type: Boolean

/**
 * String
 */
'Bewizyu'               // Type: String

/**
 * En JavaScript null is "nothing"
 * mais null est un objet
 * "You can consider it a bug in JavaScript that typeof null is an object. It should
be null."
 */
null                    // Type: object

/**
 * A variable sans valeur a la valeur undefined.
 * il est possible de réaffecter la valeur undefined a une variable
 */
undefined               // Type: undefined
var x = undefined;

// Différence entre undefined et null
typeof undefined           // undefined
typeof null                // object
null === undefined         // false
null == undefined          // true


// ******* Complexe data types ******** //

// Type: Object
var obj = {
  type: 'Formation',
  langage: 'JavaScript'
};
```

© 2018, ENI Services. Tous droits réservés.

```
['Formation', 'javascript', 'Types'] // Array => Type: object
typeof ['Formation', 'javascript', 'Types'] // => Object

// Function
var fn = function () {
  console.log('ma première function');
}
```

© 2018, ENI Services. Tous droits réservés.

4.2. JavaScript Operators

```
// ************************************************ //
// *********** Opérateurs arithmétiques ************* //
// ************************************************ //

/**
 * Addition
 */
10 + 5              // result: => 15

/**
 * Soustraction
 */
10 - 5              // result: => 5

/**
 * Multiplication
 */
10 * 5              // result: => 50

/**
 * Division
 */
10 / 5              // result: => 2

/**
 * Modulus
 */
11 % 5              // result: => 1

/**
 * Increment
 */
var i = 10;
++i;                // result: => 11

/**
 * Décément
 */
var d = 10;
--d;                // result: => 9

// ************************************************ //
// *********** Addition String & Number ************* //
// ************************************************ //

5 + 5;              // result: => 10
```

© 2018, ENI Services. Tous droits réservés.

```
'5' + 5;           // result: => '55'

'Hello' + 5;       // result: => 'Hello5'

5 + 10 + 'Hello'   // result: => '15Hello'

'Hello' + 5 + 10   // result: => 'Hello510'


// ************************************************ //
// *********** Opérateurs assignement ************** //
// ************************************************ //

var x = 5;         // assigne la valeur 5 to x

var y = 2;         // assigne la valeur 2 to y

var z = x + y;     // assigne la valeur 7 to z (x + y)

x += y             // Équivalent x = x + y

x -= y             // Équivalent x = x - y

x *= y             // Équivalent x = x * y

x /= y             // Équivalent x = x / y

x %= y             // Équivalent x = x % y

// ****************************************** //
// *********** Opérateur String ************** //
// ****************************************** //

'Formation' + ' ' + 'JavaScript'    // result: => 'Formation JavaScript'

// ************************************************* //
// *********** Opérateur de comparaison ************** //
// ************************************************* //

// Opérateur permettent d'obtenir une valeur booléenne d'une condition

var x = 10;

/**
 * == => Equal to ou égalité faible
 *
 * Le test d'égalité faible compare deux valeurs après les avoir converties en valeurs
d'un même type.
```

© 2018, ENI Services. Tous droits réservés.

```
 * Une fois converties (la conversion peut s'effectuer pour l'une ou les deux
valeurs),
 * la comparaison finale est la même que celle effectuée par ===
 *
 */
x == 5                // => false
x == 10               // => true
x == '10'             // => true

/**
 * != => Not equal to
 */
x != 5                // => true
x != 10               // => false
x != '10'             // => false

/**
 * === => Equal value and equal type ou égalité forte
 */
x === 5               // => false
x === 10              // => true
x === '10'            // => false

/**
 * != => Not equal value and not equal type
 */
x !== 5               // => true
x !== 10              // => false
x !== '10'            // => true

/**
 * > => Plus grand que
 */
x > 5                 // => true
x > 12                // => false
x > 10                // => false

/**
 * > => Plus grand que ou égal à
 */
x >= 5                // => true
x >= 12               // => false
x >= 10               // => true

/**
 * > => Plus petit que
 */
x < 5                 // => false
x < 12                // => true
```

© 2018, ENI Services. Tous droits réservés.

```
x < 10                // => false

/**
 * > => Plus petit que ou égal à
 */
x <= 5                // => false
x <= 12               // => true
x <= 10               // => true

// Comparaison de différents types
// Lorsque l'on compare des String avec des Numbers, le langage convertit les String
en Numbers avant de les évaluer.
// Un String empty est convertit en 0 et un string non numérique est casté en NaN

2 < 12                // => true
2 < "12"                // => true
2 < "John"            // => false
2 > "John"            // => false
2 == "John"           // => false
"2" < "12"            // => false
"2" > "12"            // => true
"2" == "12"           // => false


// ********************************************** //
// *********** Opérateurs Logiques ************** //
// ********************************************** //

/**
 * AND / &&
 * condition1 && condition2
 * https://fr.wikipedia.org/wiki/Fonction_ET
 */
var x=5, y=10;
x === 5 && y > 10       // => false

/**
 * OR / ||
 * condition1 || condition2
 * https://fr.wikipedia.org/wiki/Fonction_OU
 */
var x=5, y=10;
x === 5 || y > 10       // => true

/**
 * NOT / !
 */
var x = 17;
!(x === 17)            // => false
```

© 2018, ENI Services. Tous droits réservés.

```
// ********************************************** //
// *********** Opérateur ternaire *************** //
// ********************************************** //

var x = 5;

/**
 * (condition) ? value1:value2
 * Si la condition est remplie, le resultat est value1 sinon value2
 */
x === 5 ? 'Formation' : 'Javascript';    // => 'Formation'


// ********************************************** //
// *********** Opérateur type ******************* //
// ********************************************** //

/**
 * typeof
 * retourne le type d'une variable
 */

typeof "John"                 // Returns "string"
typeof 3.14                   // Returns "number"
typeof NaN                    // Returns "number"
typeof false                  // Returns "boolean"
typeof [1,2,3,4]              // Returns "object"
typeof {name:'John', age:34}  // Returns "object"
typeof new Date()             // Returns "object"
typeof function () {}         // Returns "function"
typeof myCar                  // Returns "undefined"
typeof null                   // Returns "object"

/**
 * instanceof
 * retourne true si un objet est une instance d'un objet type
 */

function Car () {}
var t = new Car();
t instanceof Car        // => true

// ************************************************ //
// *********** Opérateur Bitwise ****************** //
// ************************************************ //

// Documentation
```

© 2018, ENI Services. Tous droits réservés.

```
// https://www.w3schools.com/js/js_bitwise.asp
```

© 2018, ENI Services. Tous droits réservés.

4.3. JavaScript Function

```
// ********************************* //
// ********** Functions ********** //
// ********************************* //

// Exemple syntaxe

function myFunction() {
  // Code à exécuter
}

function multiply(p1, p2) {
  return p1 * p2;
}

// Exemple arguments

function foo (parm1) {
  console.log('Nombre d arguments reçus: '+ arguments.length);
}

foo('test');              //Nombre d arguments reçus 1
foo('test', 1);           //Nombre d arguments reçus 2
foo('test', null, .3);    //Nombre d arguments reçus 3
foo();                    //Nombre d arguments reçus 0


// Function auto invoquée
// Ce système, très largement utilisé par les framework et librairie JS, permet de d
éfinir un espace de nommage.
// Cela permet surtout de prévenir des conflits entre les noms de variables dans le
namespace globale avec des librairies tierces
// Aucune autre fonction n'a accès aux variables privées de cette fonction

// exemple fonction auto-invoquée

(function autoInvoquee() {
  // code
})();

// Exemple isolation

// contexte: Nous sommes dans une application web client (navigateur). L'objet window
est donc défini.
// Nous voulons ajouter des fonctionnalités liées au service d'une bière ;)

(function (window) {
```

© 2018, ENI Services. Tous droits réservés.

```
  var LEGAL_AGE = 18;
  var beerService = {};          // Variable privée non accessible en dehors de la
fonction

  beerService.servingBeer = function (age) {
    age = age || LEGAL_AGE;
    if (age < LEGAL_AGE) {
      return;
    }

    // bloc de code
  };

  window.beerService = beerService;

})(window);

beerService.servingBeer(39);
```

© 2018, ENI Services. Tous droits réservés.

4.4. JavaScript Objects

```
// ************************** //
// ******** Objects ********** //
// ************************** //

//
https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Le_mod%C3%A8le_objet_JavaScript_en_d%C3%A9tails
//
https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Op%C3%A9rateurs/Initialisateur_objet

// ******* Création d'objet litteral ******** //

var car = {type:"Fiat", model:"500"};
car.type;
car['type']

// ajout d'une property dynamiquement
car.color = 'white';
car['color'] = 'white';

// Supression d'une property
delete car['color'];

// Méthode
var person = {
  firstName:"John",
  lastName:"Doe",
  age:50,
  eyeColor:"blue",
  fullName: function () {
    return this.firstName + ' ' + this.lastName;
  }
};

person.fullName();

// ****** Création d'objet via Object ****** //

var o;

// on crée un objet avec null
// comme prototype
o = Object.create(null);
```

© 2018, ENI Services. Tous droits réservés.

```
o = {};
// est équivalent à :
o = Object.create(Object.prototype);


// Exemple où on crée un objet avec quelques propriétés
// (On voit ici que le second paramètres fait correspondre les clés
// avec des descripteurs de propriétés.)
o = Object.create(Object.prototype, {
  // toto est une propriété de donnée
  toto: { writable: true, configurable: true, value: 'hello' },
  // truc est une propriété d'accesseur/mutateur
  truc: {
    configurable: false,
    get: function() { return 10; },
    set: function(value) { console.log('Définir `o.truc` à', value); }
    /* avec les accesseurs ES5n on aura :
     get function() { return 10; },
     set function(value) { console.log('Définir `o.truc` à', value); } */
  }
});


function Constructeur() {}
o = new Constructeur();

// est équivalent à :

o = Object.create(Constructeur.prototype);
// Bien entendu, si la fonction Constructeur
// possède des instructions pour l'initialisation,
// Object.create() ne pourra pas le reproduire


// on crée un nouvel objet dont le prototype est
// un nouvel objet vide et on y ajoute une propriété
// 'p' qui vaut 42
o = Object.create({}, { p: { value: 42 } });

// par défaut, les propriétés ne sont PAS
// écrivables, énumérables ou configurables
o.p = 24;
o.p;
// 42

o.q = 12;
for (var prop in o) {
  console.log(prop);
}
```

© 2018, ENI Services. Tous droits réservés.

```
// 'q'

delete o.p;
// false

// Pour définir une propriété selon ES3
o2 = Object.create({}, {
  p: {
    value: 42,
    writable: true,
    enumerable: true,
    configurable: true
  }
});


// ****************************** //
// ********* Héritage *********** //
// ****************************** //

// Dès lors qu'on aborde l'héritage, JavaScript n'utilise qu'un seul concept : les objets. Chaque objet possède un lien,
// interne, vers un autre objet, appelé prototype. Cet objet prototype possède lui aussi un prototype et ainsi de suite,
// jusqu'à ce que l'on aboutisse à un prototype null. null, n'a par définition, aucun prototype et forme donc le dernier
// maillon de la chaîne des prototypes.

// https://developer.mozilla.org/fr/docs/Web/JavaScript/H%C3%A9ritage_et_cha%C3%AEne_de_prototypes

// Un constructeur JavaScript est « juste » une fonction appelée avec l'opérateur new.

function Employe () {
  this.nom = "";
  this.branche = "commun";
}

function Manager () {
  this.rapports = [];
}
Manager.prototype = new Employe();

function Travailleur () {
  this.projets = [];
}
Travailleur.prototype = new Employe();
```

© 2018, ENI Services. Tous droits réservés.

```
// ************ Constructor avec des paramètres ************ //

function Employe (nom, branche) {
  this.nom = nom;
  this.branche = branche;
}

function Manager (nom , branche, rapports) {

  Employe.call(this, nom, branche);

  this.rapports = rapports;
}
Manager.prototype = new Employe();
Manager.prototype.constructor = Manager;
```

© 2018, ENI Services. Tous droits réservés.

4.5. JavaScript Scopes

```
// code here can not use carName

function myFunction() {
  var carName = "Volvo";

  // code here can use carName

}


// exemple même scope
var x=5;
{
  var x = 10;
  console.log(x);          // => 10
}
console.log(x);            // => 10

// Exemple avec un scope définit par une fonction
var x = 10;
function f(){
  var x = 5;
  console.log(x);          // => 5
}
f();
console.log(x);            // => 10
```

© 2018, ENI Services. Tous droits réservés.

4.6. JavaScript Hoisting

```
// ***************************** //
// ********* Hoisting ********** //
// ***************************** //

// Seule les déclarations sont hoisted, pas les initialisations.
// Les exemples 1 et 2 produiront le même résultat contrairement aux exemples 3 et 4

// ****** Exemple 1 ******** //

x = 5;          // Assign 5 to x

x += 10         // => 15

var x;          // Declare x

// ********* Exemple 2 ******* //

var x;          // Declare x

x = 5;          // Assign 5 to x

x += 10         // => 15

// ********* Exemple 3 ******* //

var x = 5; // Initialize x
var y = 7; // Initialize y

 x + " " + y;           // => '5 7'

// ********* Exemple 4 ******* //

var x = 5; // Initialize x

"x is " + x + " and y is " + y;      // => 'x is 5 and y is undefined'

var y = 7; // Initialize y
```

© 2018, ENI Services. Tous droits réservés.

4.7. JavaScript Promise

```
// ********************************* //
// ************ Callbacks ********** //
// ********************************* //

function faireQqc(successCallback, failureCallback) {
  // code ....
  setTimeout(function () {
    successCallback(true);
  }, 10000);

  console.log('Synchrone');
}

function faireAutreChose (result, successCallback, failureCallback) {
  // code ....
}

function faireUnTroisiemeTruc(result, successCallback, failureCallback){
  // code ...
}

function failureCallback(){}

faireQqc(function(result) {
  faireAutreChose(result, function(newResult) {
    faireUnTroisiemeTruc(newResult, function(finalResult) {
      console.log('Résultat final :' + finalResult);
    }, failureCallback);
  }, failureCallback);
}, failureCallback);


// ********************************* //
// ************* Promises ********** //
// ********************************* //

// ********* Créer une Promise ********* //

// Exemple 1

const maPromise = new Promise(  function(resolve, reject) { /* Code... */} );
maPromise
  .then(
    (value) => {/*Success*/},
    (value) => {/*Failure*/}
  )
```

© 2018, ENI Services. Tous droits réservés.

```
maPromise
  .then(
    (value) => {/*Success*/}
  )
  .catch(
    (value) => {/*Failure*/}
  )


// Exemple 2

function faireQqc (isValid) {
  if (!isValid) {
    return Promise.reject('Error')
  }
  return Promise.resolve('Ok')
}

const maPromise = faireQqc(true).then(()=>{/*Code*/}).catch(()=>{/*Code*/})


// ********* Chaine de promesses ********* //

function faireQqc () {
  return new Promise((resolve) => {
    setTimeout(()=> {
      resolve(true, [], {name: 'niko'});
    }, 10000);
  });
}

function faireAutreChose () {
  return new Promise((resolve) => {
    setTimeout(()=> {
      resolve(true);
    }, 10000);
  });
}

function faireUnTroisiemeTruc () {
  return new Promise((resolve) => {
    setTimeout(()=> {
      resolve(true);
    }, 10000);
  });
}

faireQqc()
```

© 2018, ENI Services. Tous droits réservés.

```
  .then(function(isTest, array, obj) {
    return faireAutreChose(isTest);
  }, () => {
    console.log('Rejected ');
  })
  .then(function(newResult) {
    return faireUnTroisiemeTruc(newResult);
  })
  .then(function(finalResult) {
    console.log('Résultat final : ' + finalResult);
  })
  .catch(failureCallback);

// ********* Async/Await ********** //

// Ce fonctionnement est construit sur les promesses et faireQqc() est la même
fonction que celle utilisée dans les exemples précédents.


// Exemple 1

async function toto() {
  try {
    let result = await faireQqc();
    let newResult = await faireQqcAutre(result);
    let finalResult = await faireUnTroisiemeTruc(newResult);
    console.log('Résultat final : ' + finalResult);
  } catch(error) {
    failureCallback(error);
  }
}

// Exemple 2

function resolveAfter2Seconds(x) {
  return new Promise(resolve => {
    setTimeout(() => {
      resolve(x);
    }, 2000);
  });
}

async function add1(x) {
  var a = resolveAfter2Seconds(20);
  var b = resolveAfter2Seconds(30);
  return x + await a + await b;
}

add1(10).then(v => {
```

© 2018, ENI Services. Tous droits réservés.

```
  console.log(v);  // affiche 60 après 2 secondes.
});

async function add2(x) {
  var a = await resolveAfter2Seconds(20);
  var b = await resolveAfter2Seconds(30);
  return x + a + b;
}

add2(10).then(v => {
  console.log(v);  // affiche 60 après 4 secondes.
});
```

© 2018, ENI Services. Tous droits réservés.