IIM
DIGITAL SCHOOL
DE VINCI PARIS

Algorithmie JS

C&DI – 2023-2024

Les outils d'un dev

Les navigateurs Web

Les navigateurs ne sont pas uniquement des moteurs de recherche.
Ils sont aussi dotés d'outils indispensables aux développeurs web, pour tester, debugger, et visualiser leur code.

Chaque navigateur à son comportement qui lui est propre.
Il est donc préférable d'en installer plusieurs pour pouvoir tester et adapter son site.

Les éditeurs de code

Un éditeur de code est indispensable pour coder efficacement.

Des outils intégrés aux éditeurs, et des plugins ajoutés, permettent de faciliter la vie du développeur en accélérant son workflow et en améliorant son environnement de travail.

Les forums / blogs

Il faut comprendre qu'un développeur c'est avant tout quelqu'un qui cherche la solution à son problème.

Heureusement, cette communauté est solidaire et échange sur tous les sujets liés au code, peu importe le langage, peu importe le problème.

Lorsque vous avez un problème, quelqu'un l'a eu avant vous, alors renseignez-vous

Les IA génératives

Aujourd'hui il est indispensable pour un développeur d'avoir sous la main une IA qui va l'***ACCOMPAGNER*** au quotidien.

Les deux outils les plus utilisés en termes d'intelligence artificielles sont

- GitHub Copilot
- ChatGPT

On insiste sur le terme ACCOMPAGNER. C'est l'IA qui vous aide sur votre réflexion et pas l'inverse !

GitHub

Tout au long de l'année, vous allez développer divers petits scripts, petites pages web.

Pour déposer vos rendus, vous allez utiliser la plateforme GitHub.

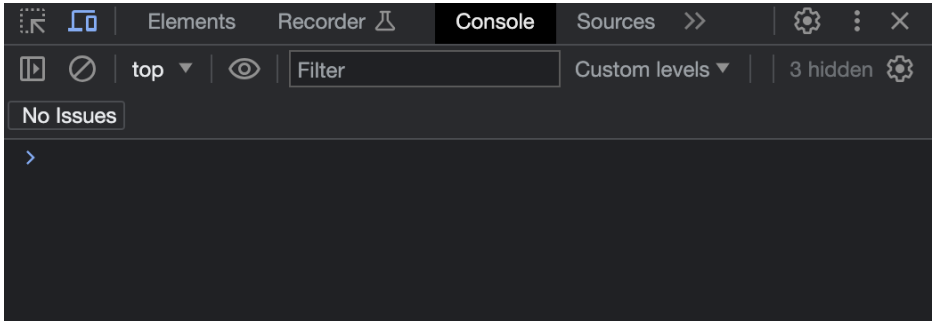
GitHub est vulgairement un mélange d'un grand drive et d'un réseau social pour développeurs.

Vous allez donc tous pouvoir vous créer un compte et le renseigner à votre professeur !

Retour sur le navigateur : La Console

Pour javascript, vous allez avoir besoin d'utiliser la console de votre navigateur.

Vous l'avez déjà toutes et tous probablement déjà ouverte sans faire exprès, en appuyant sur **F12** ou **clic droit > inspecter l'élément**.



C'est ici que nous allons tester et débugger notre code Javascript !

Le javascript : késako ?

Langage de programmation qui permet de :

- Créer du contenu mis à jour de façon dynamique,
- De contrôler des datas,
- D'animer tout ce à quoi on peut penser.
- Et bien plus encore

```
console.log('Hello world !')

if(person.skill == 'javascript') {
  person.cool = true
}
```

Comment ça marche ?

- On va créer un dossier dans lequel on va mettre un **index.html** et un **main.js**.

- Il faut lier le **JS** avec la page **HTML**. Pour ça on utilise la balise **‹script›‹/script›**

- Il faut ajouter comme attribut le **src** pour indiquer quel fichier **JS** on souhaite lier.

(Vous aurez plus de détail sur le fonctionnement de l'HTML lors de votre cours dédié.)

```
<script src="main.js"></script>
```

- Pour vérifier si notre fichier est bien lié, on va faire un petit message d'entrée avec une fonction **alert()**.

```
alert("je suis bien lié à l'index.html")
```

Les types de variables

En **JS** le type de variable est implicite, mais il existe.

- **int** => entier
- **bigint** => grand entier
- **float** => nombre à virgule
- **string** => chaîne de caractère
- **boolean** => vrai ou faux
- **array** => tableau
- **object** => objet
- **null** => rien

var, **let** et **const** ne sont pas des types de variable !

Mais que sont-ils alors ?

Les variables

Une **variable** est un conteneur (associé à un nom) qui garde des valeurs.

Pour déclarer une variable, il existe 3 mots-clés :

- **var** : définit une variable en global
- **let** : définit une variable dans son bloc courant
- **const** : définit une variable qui ne peut pas être modifiée par la suite

```
var animal = 'cat'
let age = 21
const lastname = 'de Garrigues'
```

Mais de quel type c’est ?

- Le nombre d’Avengers ?
- Le nom de famille de l’empereur romain Jules ?
- Le nombre d’étoiles dans l’univers ?
- Message de bienvenue sur un paillasson ?
- Présence d’un Face ID sur un téléphone ?
- Prix total d’un ordinateur monté à la main ?
- Noms des 7 nains de Blanche Neige ?
- Profil d’un beau gosse sur Tinder ?

Mais de quel type c’est ?

- La quantité de clones que peut faire Naruto ?
- 3 défauts à donner à l’embauche (#perfectionniste) ?
- Le prénom Anne dans la Flamme ?
- Un panier sur Amazon ?
- Durée en années de la guerre de 100 ans ?
- Caractéristiques d’une bière ?
- Le nom du prochain président de la République ?

Les conditions

If / Else :

Pour exécuter une instruction :

- Si une condition donnée est vraie, on utilise le **if**.
- Si la condition n'est pas vérifiée, on utilise le **else**

```
if(condition == true) {
  // do something
} else {
  // do something else
}
```

If / Else :

Il existe différents opérateurs pour vérifier une instruction, parmi elles, on retrouve :

==	qui vérifie si les arguments sont égaux
===	qui vérifie si les arguments sont égaux **et** de même type
!=	qui vérifie si les arguments sont différents
> /< / >= / <=	qui vérifie si un argument est plus petit ou plus grand que l'autre.
&& / ||	respectivement « **ET** » et « **OU** »

Exercice **if**/**else**:

- Déclarer trois variables **a = 4**, **b = 4**, **c = 3**

    - si A est égal à B, faire un **console.log("c'est égal")**

    - si C est plus petit que B et que A + C est différent de 3 faire un **console.log("ok")**, sinon faire un **console.log("raté")**

```
let a = 4
let b = 4
let c = 3

if(a == b) {
  console.log('égal')
}

if(c < b && a+c ≠ 3) {
  console.log('ok')
} else {
  console.log('raté')
}
```

Switch case:

Le switch est un ensemble de **if/else** avec une syntaxe plus propre.
C'est un comparateur.

Chaque cas possible est écrit **case**.
À chaque fin de cas on écrit **break**.

si **a** égal à 1

si **a** égal à ‘oklm’

si **a** n'est égal à aucune case, il passe dans le default

```
switch(a) {
  case 1:
    console.log('a = 1')
    break
  case 'oklm':
    console.log('a = oklm')
    break
  default:
    console.log('default')
}
```

Exercice **switch**:

- Déclarer trois variables **a = 4**, **b = 4**, **c = 3**
    - **a** sera l'expression de référence
    - si **a** est égal à **b**, alors faire un **console.log('égal à b')**
    - si **a** est égal à **c**, alors faire un **console.log('égal à c')**
    - par défaut, il y a un **console.log('égal à rien')**

```
let a = 4
let b = 4
let c = 3

switch(a) {
  case b:
    console.log('égal à B')
    break
  case c:
    console.log('égal à C')
    break
  default:
    console.log('égal à rien')
}
```

Les boucles

for :

Le **for** est une boucle qui se répète **tant que la condition est valide**.
Elle initialise un **index** qui va s'incrémenter ou se décrémenter avec le temps.

```
for(let i = 0; i < 10; i++) {
  console.log(i)
}
```

Initialement i est égale à zéro. Tant que i est plus petit que 10, il passe dans la boucle, et à la fin, il augmente de 1 (i++)

Exercice **for** :

- Déclarer une variable **a** qui est égale à un entier positif

- Faire un console log de "**oklm**" autant de fois que la valeur de **a**

```
let a = 4

for(let i = 0; i < a; i++) {
  console.log('oklm')
}
```

while :

le **while**, comme le for est autre une boucle qui se répète tant que la condition est valide.

```
let i = 0
while(i < 10) {
  console.log(i)
  i++
}
```

Deux mots clés rentrent en jeux:

- **break** qui stop la boucle et passe à la suite
- **continue** qui retourne en haut de la boucle

Exercice **While**:

- Déclarer une variable **a** qui est égal à 3
- Tant que **a** est plus petit que 9

    - incrémenter **a** de 1

    - si **a** est égal à 8 on arrête la boucle

    - si **a** est égal à 7 on revient au début de la boucle

    - on **console.log(a)**

```
let a = 3
while(a < 9) {
  a++
  if(a == 7) {
    continue
  }
  if(a == 8) {
    break
  }
  console.log(a)
}
console.log('j\'ai cassé la boucle au bout de ' + a + ' tours')
```

Les fonctions

Fonction :

C'est un ensemble d'instructions effectuant une tâche ou calculant une valeur.

```
let a = 3
let b = 4

function addition(number1, number2) {
  console.log(number1 + number2)
}

addition(a, b)
```

Ici on déclare deux variables et une fonction.

Cette fonction prend en paramètres deux nombres.

On finit par appeler la fonction avec nos variables en paramètre.

Fonction :

Il est possible de faire un **return** plutôt qu'un **console.log()** pour stocker dans une autre variable le résultat de la fonction.

```
let a = 3
let b = 4
let somme = addition(a, b)

function addition(number1, number2) {
  return number1 + number2
}
```

Ici on déclare trois variables et une fonction.

Une des variables appelle la fonction et attend donc le résultat du return

Exercice **fonction**:

- Déclarer trois variables **a = "Jean"** et **b="Paul"** et **result**

- Déclarer une fonction **checkName** (qui possède 2 paramètres) qui vérifie si les deux noms sont identiques

- console.log le **result**

Les tableaux

Tableau :

En **JS** le tableau est un objet qui regroupe plusieurs éléments.

let array = ["NicoDG", "PierrC"]

on accède aux différents éléments via un index et des crochets

array[0] => "NicoDG"

on peut:

-ajouter des éléments => **array.push("AlexB")**
-supprimer des éléments => **array.splice(index, nombreDElementASupp)**
-connaître la taille du tableau => **array.length**
- et bien d'autres choses....

Foreach:

Le foreach permet d'effectuer des instructions sur chacun des éléments du tableau

```
let fruits = ['pomme', 'poire', 'banane']

fruits.forEach(function (fruit) {
  console.log(fruit)
})
```

Fonctions Fléchées:

Une expression de fonction fléchée permet d'avoir une syntaxe plus courte.
Une fonction fléchée est souvent anonyme et n'est pas destinée à être utilisée pour déclarer une méthode.

```
// fonction classique
fruits.forEach(function (fruit) {
  console.log(fruit)
})

// fonction fléchée
fruits.forEach((fruit) ⇒ {
  console.log(fruit)
})
```

Exercice **Tableau**:

- Déclarer un tableau vide **names**

    - Ajouter "Vincent", "Paul" et "Arthur" dans le tableau via la fonction **push**

- Pour chaque élément du tableau

    - Ajouter "va à la pêche" à la fin

    - Afficher chaque élément

Les objets

Object :

En **JS** l'objet est une liste qui regroupe une ou plusieurs propriétés.

On accède aux différentes caractéristiques via le nom de l'objet et le nom de la propriété.

professor.favoriteFood => 'Salad'

```
let student = {
  name: 'Nicolas',
  favoriteFood: 'Salad',
  city: 'Paris',
}
```

Exercice **Objet**:

- Déclarer un objet **student** avec **name, favoriteFood, city**.

- Récupérer le nombre de caractères dans chaque propriété, puis les additionner pour obtenir un nombre.

- Si ce nombre est pair, afficher dans la console "pair"
- Si ce nombre est impair, afficher dans la console "impair"

Comme pour le reste, il existe plusieurs solutions possibles :

- Object.keys() => récupérer les propriétés
- Object.values() => récupérer les valeurs

Les classes

Class:

La classe est une structure de donnée.

Un Schéma d'objet qui peut avoir plusieurs instances.

Elle peut avoir des variables et des fonctions propres

```
class Guerrier {
  constructor(attack, defense) {
    this.attack = attack;
    this.defense = defense;
  }

  getAttack() {
    console.log(this.attack);
  }
}

let Alexis = new Guerrier(10, 5);
Alexis.getAttack();
```

Exercice Class:

-Créer une classe **Pokemon** avec comme paramètres **name, attack**, **defense**, **hp** et **luck** une fonction **isLucky**, et une fonction **attackPokemon**
-Créer **deux instances** de **Pokémon** avec des stats différentes.

- Tant que l'un des deux n'est pas mort
    - le premier attaque le second (isLucky + attackPokemon)
    - afficher la vie et les dégâts endommagés du second
    - si le second est mort, arrêter la boucle
    - le second attaque le premier (isLucky + attackPokemon)
    - afficher la vie et les dégâts endommagés du premier
- Afficher un message de mort pour le pokemon perdant

Attention

- la formule des dégâts est la suivante: **dégâts = att de l'attaquant – def du defenseur**
- la luck correspond à la probabilité de toucher l'ennemi (précision en pourcentage)
- générer un nombre aléatoire avec **Math.random()** (qui retourne un décimal entre 0 et 1)
- si ce nombre est inférieur à luck du Pokemon alors le Pokemon peut attaquer

Le debug

Le debug :

Il arrive que votre code ne fonctionne pas malgré une bonne logique.

Les causes :
- une erreur de syntaxe
- un oubli de ponctuation
- mauvaise utilisation d'une propriété

La solution - Délimiter le problème :
Utiliser des logs à chaque étape afin de regarder à quel moment le résultat attendu ne s'affiche pas correctement.

Exercice :

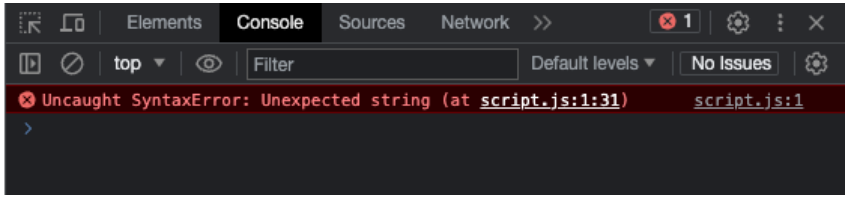
Le code suivant ne fonctionne pas (cf le résultat ci-dessous dans la console).

Corrigez-le afin de le faire fonctionner et que le résultat de la fonction countCharacter soit logique.

```
let users = ["Stéphanie" Gaïa", "Etienne", "Michel"; "Roberto", "Julie"]

function countCharacter(value) {
    return value.length
}

users.forEach(user => {
    if (countcharacter(users) < 5) {
        console.log('c'est un prénom long de plus de 5 lettres.')
    } else {
        console.log("ce n'est pas du tout un prénom long.")
    }
});
```



Rendu

Exo 1 : Le taxi

John essaie de rentrer chez lui en taxi.
Problème : Il a horreur d'écouter Anissa de Wejdene qui passe tout le temps à la radio.
Dès qu'il entend cette musique, il perd **1** de santé mentale et change de taxi.

Partons du principe qu'une musique se change à chaque feu rouge qu'il rencontre et qu'il est à 30 feux rouges de chez lui.

À chaque feu rouge, afficher la musique jouée + le nombre de feux restants.

Deux possibilités de fin :

- Si il a passé les 30 feux rouges, il est arrivé à destination et donc affiche qu'il est bien arrivé et qu'il lui a fallu x changements de taxi pour y arriver
- Sa santé mentale tombe à 0, il explose et donc affiche « explosion »

Nous avons besoin d'un **Personnage** avec un prénom et une santé mentale à 10.
Nous avons besoin d'une liste de **5 musiques** dont **Anissa - Wejdene**
Nous avons besoin d'un **Trajet** avec une radio, 30 feux rouges et un nombre de changements

Exo 2 : Le tueur en série

Un tueur en série nommé Jason est en cavale. Il est caché quelque part en forêt.
Une équipe de choc est présente pour le neutraliser.

Nous avons besoin d'un **tueur** nommé Jason et qui possède **100 points de vie**.
Nous avons besoin de **Caractéristiques** de personnages avec des noms bien clichés (nerd, sportif, blonde…), une probabilité de mourir, une de mettre des dégâts et une de mourir en mettant des dégâts (ex: 0.3, 0.5, 0.2)
Nous avons besoin de **5 Survivants** avec un nom généré aléatoirement d'un tableau de prénoms et d'une caractéristique prise de celles disponibles (toujours aléatoire).

Tant que le tueur n'est pas mort ou que les survivants n'ont pas tué Jason :

Le tueur attaque un des survivants :

- soit le survivant **meurt**
- soit le survivant esquive et inflige **10 points de dégâts**
- soit le survivant inflige **15 points de dégâts** mais **meurt**

Les morts seront affichés à la fin

Un message est attendu pour chaque action (Jason a tué X, X a esquivé et a infligé X dmg, Jason est mort, Les survivants ont gagné mais RIP à X, X, X…)

Utilisation des conditions	Utilisation des boucles	Syntaxe JS	Algorithme Taxi	Algorithme Jason
/3	/3	/4	/5	/5

Le rendu se fera sur GitHub.

Le repository comprendra la totalité des fichiers vus en cours, nommés selon les parties (ex: *boucles.js*, *conditions.js*...).

Il comprendra en plus 2 fichiers *taxi.js* et *jason.js* représentant les 2 exos de rendu.