



# DIGINAMIC

FORMATION  
NodeJS - 5 JOURS



# Chapitre 1

# Rappels ES6

# ES ou ECMASCIPT

# Introduction à ES6

- Publié en 2015, avec de nombreuses évolutions par rapport à sa version précédente ES5
- Un standard pour un langage de programmation

{ ES6 JS }

- Lien utile : <https://htmlcheatsheet.com/js/>
- Pour vous exercer : <https://www.programiz.com/javascript/online-compiler/>

# Concepts fondamentaux

# Opérateur ternaire

- Forme abrégée de l'instruction if...else

```
condition ? expressionSiVrai : expressionSiFaux;

// Exemple de ternaire
let nombre = 10;
let resultat = nombre > 0 ? "positif" : "negatif";
console.log(resultat); // affiche "positif"
```

# Rappel : let, var et const

- Introduction des mots clés let et const
- let et const ajoutent une notion de portée

```
function varTest() {  
    var x = 1;  
    if (true) {  
        var x = 2; // c'est la même variable !  
        console.log(x); // 2  
    }  
    console.log(x); // 2  
}  
  
function letTest() {  
    let x = 1;  
    if (true) {  
        let x = 2; // c'est une variable différente  
        console.log(x); // 2  
    }  
    console.log(x); // 1  
}  
  
function constTest() {  
    const x = 1;  
    if (true) {  
        const x = 2; // c'est une variable différente  
        console.log(x); // 2  
    }  
    console.log(x); // 1  
}  
  
// x = 3; // Erreur : Assignment to constant variable.  
  
varTest();  
letTest();  
constTest();
```

# Manipulation des array

- Une syntaxe plus concise qui simplifie le code

Uploaded using RayThis Extension

```
//array structuration
let arrayStructuration = [1, 2]
console.log(arrayStructuration); // [1, 2]

//array copy
let arrayCopy = [...arrayStructuration, 3, 4]
console.log(arrayCopy); // [1, 2, 3, 4]

//array destructurization
let [a, b] = arrayCopy;
console.log(a); // 1
console.log(b); // 2
```

Uploaded using RayThis Extension

```
//object structuration
let obj = {
  message: "hello world"
}
console.log(obj); // {message: "hello world"}
```

```
//object add key
obj.type = "success";
console.log(obj); // {message: "hello world", type: "success"}
```

```
//object copy
let copy = {...obj}
```

```
//object destructurization
const {message} = obj;
console.log(message); // "hello world"
```

# Les classes

- Avant sous forme de prototype
- Permet désormais d'utiliser des classes au lieu de prototypes

```
...  
Uploaded using RayThis Extension  
  
// ES5  
function Person(name, age) {  
    this.name = name;  
    this.age = age;  
}  
  
Person.prototype.greet = function() {  
    return "Hello, my name is " + this.name;  
};  
  
// ES6  
class Person {  
    constructor(name, age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    greet() {  
        return `Hello, my name is ${this.name}`;  
    }  
}
```

# Fonctions fléchées

- Une syntaxe plus concise qui simplifie le code

● ● ●      Uploaded using RayThis Extension

```
// ES5
function foo() {
    console.log("Hello from foo");
}

// ES6
const foo = () => {
    console.log("Hello from foo");
}

// plus concise arrow function
const foo = () => console.log("Hello from foo");
```

● ● ●      Uploaded using RayThis Extension

```
// ES5
const array = [1, 2, 3];

for (let i = 0; i < array.length; i++) {
    console.log(array[i]);
}

// ES6
const arrayES6 = [1, 2, 3];
array.map(item => console.log(item));
```

# Chaînes de caractères

- Une syntaxe plus concise qui simplifie le code : template literals

```
● ● ●          Uploaded using RayThis Extension

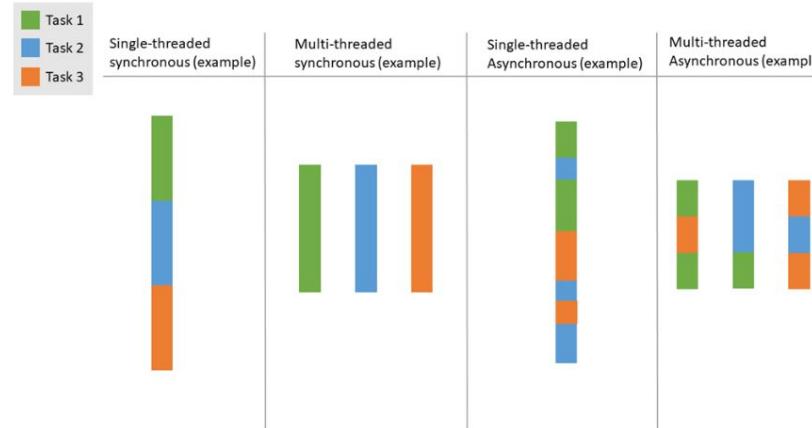
// ES5
var name = 'John';
var greeting = 'Hello, ' + name + '!';

// ES6
var name = 'John';
var greeting = `Hello, ${name}!`; // Template literals
```

# Les threads

# Notion de thread

- Unité d'exécution dans un processus qui permet d'effectuer plusieurs tâches en parallèle
- Permet le multitâche, la réactivité, mais souvent plus complexe et dépendant du reste



# Les promesses

# Les promesses

- Pour gérer les opérations asynchrones de manière plus propre et plus lisible qu'avec les callbacks traditionnels
- Les callbacks sont des fonctions passées en argument à d'autres fonctions et exécutées après la fin de l'opération principale
- Une fonction avec `async` retourne une promesse.
- `await` est utilisé pour attendre la résolution de cette promesse

```
Uploaded using RayThis Extension

// ES5, utilisation de callback
function sleep(ms, callback) {
  setTimeout(callback, ms);
}

function asyncFunctionCallback() {
  console.log("startES5");
  sleep(3000, function() {
    console.log("endES5"); // exécuté après 3 secondes
  });
}

asyncFunctionCallback();

// ES6 : promesses
const sleep = (ms) => new Promise(resolve => setTimeout(resolve, ms));

const asyncFunction1 = () => {
  console.log("startES6");
  sleep(3000).then(() => {
    console.log("endES6"); // exécuté après 3 secondes
  });
}

asyncFunction1();

// ES8 : await / async
const sleep = (ms) => new Promise(resolve => setTimeout(resolve, ms));

const asyncFunction2 = async () => {
  console.log("startES8");
  await sleep(3000);
  console.log("endES8"); // exécuté après 3 secondes
};

asyncFunction2();
```

# Les promesses

- Après 3 secondes, ou immédiatement ?

```
// Ex 1
const sleep = (ms) => new Promise(resolve => setTimeout(resolve, ms));

const asyncFunction = async () => {
  console.log("start");
  await sleep(3000);
  console.log("end");
};

asyncFunction();

// Ex 2
const sleep = (ms) => new Promise(resolve => setTimeout(resolve, ms));

const asyncFunction2 = () => {
  console.log("startES6");
  sleep(3000).then(() => {
    console.log("endES6");
  });
};

asyncFunction2();
```

# Les promesses, gestion d'erreur

- resolve et reject utilisés pour marquer une promesse comme résolue ou la rejeter
- try/catch utilisé pour la gestion d'erreurs

```
...  
Uploaded using RayThis Extension  
  
// ES6  
asyncFunction()  
  .then(result => {  
    console.log('Résultat : ', result);  
  })  
  .catch(error => {  
    console.error('Erreur : ', error.message);  
});
```

```
...  
Uploaded using RayThis Extension  
  
// ES6  
function asyncFunction() {  
  return new Promise((resolve, reject) => {  
    // Simulation d'une opération asynchrone  
    setTimeout(() => {  
      const randomNumber = Math.random();  
      if (randomNumber > 0.5) { // Résolution de la promesse avec le nombre aléatoire  
        resolve(randomNumber);  
      } else { // Rejet de la promesse avec une erreur  
        reject(new Error('Une erreur s\'est produite !'));  
      }  
    }, 1000);  
  });  
}  
  
// ES8  
try {  
  const result = await asyncFunction();  
  console.log('Résultat : ', result);  
} catch (error) {  
  console.error('Erreur : ', error.message);  
}
```

# Les promesses, exemple “waterfall”

- Exécution séquentielle d'opérations asynchrones.
- Chaque opération dépend du résultat de la précédente

```
Uploaded using RayThis Extension

async function asyncOperation1() {
}

async function asyncOperation2() {
}

async function asyncOperation3() {
}

async function executeAsyncOperations() {
    try {
        const result1 = await asyncOperation1();
        const result2 = await asyncOperation2(result1);
        const result3 = await asyncOperation3(result2);
        console.log('Résultat final : ', result3);
    } catch (error) {
        console.error('Erreur : ', error);
    }
}

executeAsyncOperations();
```

# Les promesses, exemple “parallèle”

- Exécution parallèle d'opérations asynchrones.
- Toutes les opérations sont lancées en même temps, et les résultats sont traités ensemble.



Uploaded using RayThis Extension

```
async function asyncOperation1() {
  // ...
}

async function asyncOperation2() {
  // ...
}

async function executeParallelOperations() {
  try {
    const [result1, result2, result3] = await Promise.all([
      asyncOperation1(),
      asyncOperation2(),
    ]);
    console.log('Résultat 1 : ', result1);
    console.log('Résultat 2 : ', result2);
    console.log('Résultat 3 : ', result3);
  } catch (error) {
    console.error('Erreur : ', error);
  }
}

executeParallelOperations();
```

# **Exercices**

# Chapitre 2

# Node et ses modules

# Introduction à Node JS

# Qu'est ce que Node JS ?

- Un environnement d'exécution Javascript créé par Ryan Dahl en 2009
- Permet le développement côté serveur
- Utilise le moteur JavaScript V8, le même que celui de Google Chrome
- Asynchrone et évènementiel
- Stateless : chaque requête est indépendante



# De nombreuses applications utilisent Node JS

- IBM
- Netflix
- Amazon Web Services
- LinkedIn
- Microsoft
- Yahoo
- Rakuten
- PayPal



# Les objets globaux

- **global** : Les variables définies globalement peuvent être accessibles via cet objet.
- **process** : Fournit des informations et un contrôle sur le processus Node.js actuel
- **console** : Utilisé pour imprimer des messages de debug à la console.
- **dirname** : Le chemin du répertoire du fichier en cours d'exécution.
- **filename** : Le chemin complet du fichier en cours d'exécution.

```
...  
Uploaded using RayThis Extension  
  
console.log(__dirname); // Imprime le répertoire actuel  
console.log(__filename); // Imprime le chemin complet du fichier actuel  
  
globalThis.myGlobalVariable = "Hello, World!"; // Définition d'une variable globale  
console.log(globalThis.myGlobalVariable); // Accès à la variable globale
```

# Gestionnaire de paquets

# Utiliser NPM comme gestionnaire de packages

- Gestionnaire de packages officiel de node (2010)
- Permet l'installation de modules complémentaires

```
npm install nom_module
```

```
npm uninstall nom_module
```

```
npm init
```



# Modules fondamentaux

# http

- Créer des serveurs web et de gérer les requêtes et réponses HTTP

```
● ● ●          Uploaded using RayThis Extension

const http = require('http');

const server = http.createServer((req, res) => {
    res.statusCode
    res.setHeader
    res.end('Hello World');
});

server.listen(3000, () => {
    console.log('Server running at http://localhost:3000/');
});
```

# fs

- Lire, écrire et modifier des fichiers

```
● ● ● Uploaded using RayThis Extension

const fs = require('fs');

fs.readFile('c:/Users/rddin/Desktop/test.js', (err, data) => {
    if (err) throw err;
    console.log(data.toString());
});
```

# path

- Manipuler les chemins

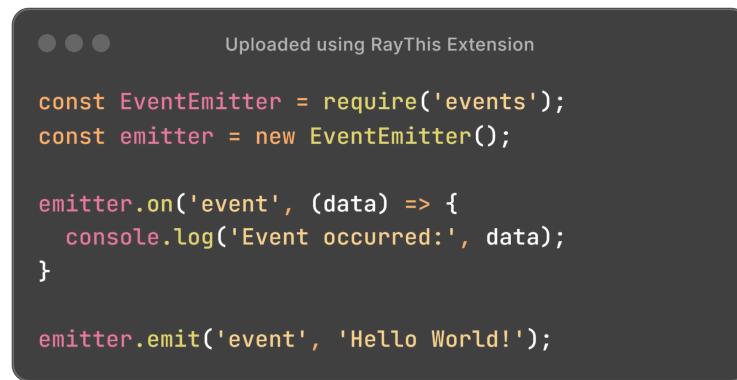


Uploaded using RayThis Extension

```
const path = require('path');
const filePath = path.join(__dirname, 'test.txt');
console.log(`File path: ${filePath}`);
```

# events

- Gérer des évènements



Uploaded using RayThis Extension

```
const EventEmitter = require('events');
const emitter = new EventEmitter();

emitter.on('event', (data) => {
  console.log('Event occurred:', data);
}

emitter.emit('event', 'Hello World!');
```

# OS

- Récupérer les informations du système d'exploitation

```
const os = require('os');

const getSystemInfo = () => {
  const platform = os.platform(); // e.g., 'linux', 'darwin', 'win32'
  const arch = os.arch(); // e.g., 'x64', 'arm64', 'ia32'
  const cpus = os.cpus().length; // Number of CPU cores
  const totalMemory = os.totalmem(); // Total system memory in bytes
  const freeMemory = os.freemem(); // Free system memory in bytes

  return {
    platform,
    arch,
    cpus,
    totalMemory,
    freeMemory
  };
}
```

# dns

- Fonctions pour effectuer des requêtes DNS

● ● ●                          Uploaded using RayThis Extension

```
const dns = require('dns');

dns.lookup('google.com', function(err, address) {
  if (err) {
    console.log('error');
  } else {
    console.log(address);
  }
});
```

url

- Effectuer différentes opérations sur les URLs

```
● ● ● Uploaded using RayThis Extension

const url = require('url');

const myUrl = new URL('https://example.com:8000/path?query=123#hash');

const parsedUrl = new URL(myUrl);

console.log(parsedUrl.protocol); // https:
console.log(parsedUrl.hostname); // example.com
console.log(parsedUrl.port); // 8000
console.log(parsedUrl.pathname); // /path
// ... etc
```



# **Exercices**



# TP 1 : Validation des acquis