T1a - Utilisation de Node.js

Exécution de scripts avec Node.js

Contenu de l'atelier T1

T1a Exécution de code **JavaScript** avec **Node.js**
Variables et objets en **JavaScript**

T1b Écriture de code **JavaScript**. Tableau. Boucle **for...of**
Fonction constructeur. Fonction auxiliaire.

T1c Standards **CommonJS** et **ESM**. Configuration de **Node** et de **VS Code**
Création et utilisation d'un module **ESM** : **export**, **import**

T1d Utilisation de dépendances :
`pnpm add`, **`package.json`**, **`node_modules`**, **`pnpm run`**

T1e Génération du squelette d'un projet avec un framework web
`pnpm add -g`, **`CLI`**, **`pnpm install`**, stratégie de stockage de **PNPM**

T1f Compléments / Entraînement
Conflit de noms entre objets importés

Création du dossier de travail global

Vous allez choisir un dossier pour stocker tous les projets qui seront créés au cours de cette série d'ateliers. Ce sera votre **dossier de travail global** pour les ateliers **Outils Dev**.

- Déterminez l'emplacement que vous souhaitez pour ce dossier.
  Attention ! Il vaut mieux que le chemin absolu de ce dossier ne contienne pas d'espaces et pas de caractères accentués.
  Choisissez plutôt un chemin court comme `C:\Ateliers\Outils-Dev`
- Créez ce dossier sur votre disque.
- Dans ce dossier, créez un premier sous-dossier nommé `t1-bonjour`
  Il va contenir votre premier projet **JavaScript**.

Node.js

Node.js est un environnement d'exécution **JavaScript** gratuit, open-source et multiplate-forme qui permet aux développeurs de créer des serveurs, des applications web, des outils en ligne de commande et des scripts.

Licence Creative Commons BY-SA 4.0 - AMBLARD Emmanuel 2026

Node.js permet aussi de développer des applications de bureau comme, par exemple, **VS Code** ou **Postman**. Elles sont généralement basées sur le framework **Electron**, qui combine le moteur de rendu **Chromium** avec le runtime **Node.js**.

Node.js repose sur le moteur **JavaScript V8** développé par **Google** pour son navigateur **Chrome**. Il intègre également plusieurs modules dont ceux qui lui permettent de gérer une boucle d'évènements et de fonctionner en tant que serveur web.

Exécution en ligne de commande

- Dans **VS Code**, ouvrez le dossier **t1-bonjour** que vous avez créé à l'étape précédente.
  ➔ Menu : **File** > **Open Folder...** ou Ctrl+K Ctrl+O
- Créez un nouveau fichier nommé **bonjour.js**
- Commencez à taper : `log`
  Puis, utilisez l'autocomplétion de code pour écrire la ligne :
  ```
  console.log('Bonjour')
  ```
- Enregistrez le fichier.
- Dans **VS Code**, ouvrez la vue **Terminal** (Ctrl+ù ou menu : **View** > **Terminal**)
- Exécutez la commande :
  ```
  node bonjour.js
  ```

Comme vous le voyez, la commande est tapée dans la vue **Terminal** et le résultat est également affiché dans cette vue.

Utilisation de Code Runner

Lorsque l'extension `Code Runner` est installée dans **VS Code**, il n'est pas nécessaire de taper une commande dans le **Terminal** pour exécuter un **script**.

- Faites un clic-droit n'importe où dans le fichier **bonjour.js** (c'est-à-dire dans la vue centrale). Puis : **Run Code**
  On peut aussi appuyer sur les touches Ctrl+Alt+N

Notez que le résultat est affiché, non pas dans la vue **Terminal**, mais dans la vue **Output**.

Variables et objets en JavaScript

Puisqu'il est possible d'exécuter du code, c'est le moment de rafraîchir vos connaissances en **JavaScript**.

Licence Creative Commons BY-SA 4.0 - AMBLARD Emmanuel 2026

Nature dynamique d'un objet

En **Java**, un objet appartient à une classe ce qui lui donne une structure figée.
Par exemple, chaque objet de la classe **Personne** aura un nom, une adresse, une date de naissance, etc.

En **JavaScript**, il n'y a pas de classes.

Un objet est donc créé à partir de rien et on peut lui ajouter des données de façon dynamique tout au long de son existence. On peut même lui en retirer si on le souhaite.

- Créez un nouveau fichier nommé **`objets.js`**
- Écrivez le code ci-dessous :

```
const p1 = new Object()
console.log(p1)

p1.nom = 'Aline'
console.log(p1)

p1.age = 20
console.log(p1)

p1.enEmploi = false
console.log(p1)
```

- Exécutez le script. (Pensez à enregistrer le fichier au préalable)

  Vous devez obtenir le résultat suivant :

```
{}
{ nom: 'Aline' }
{ nom: 'Aline', age: 20 }
{ nom: 'Aline', age: 20, enEmploi: false }
```

Au départ, on a créé un objet vide : **`const p1 = new Object()`**

Puis, on lui a rajouté petit à petit des propriétés, ce qui fait que sa structure a évolué dynamiquement au cours du traitement. Ceci n'est pas possible en **Java**.

Syntaxe littérale

L'initialisation d'un objet peut être faite propriété par propriété comme dans notre exemple :

```
const p1 = new Object()
p1.nom = 'Aline'
p1.age = 20
p1.enEmploi = false
```

Licence Creative Commons BY-SA 4.0 - AMBLARD Emmanuel 2026

Mais, on aurait pu aussi utiliser la **syntaxe littérale** qui est plus concise et plus lisible :

```
const p1 = {
  nom: 'Aline',
  age: 20,
  enEmploi: false,
}
```

- Dans votre script, laissez la définition de l'objet **p1** dans sa forme initiale (initialisation des propriétés l'une après l'autre)
- Écrivez le code qui crée un objet **p2**, en utilisant la syntaxe littérale.
  Cet objet correspond à une personne nommée Brice, âgé de 30 ans et qui a un emploi.
- À la fin, faites afficher l'objet **p2**. Vous devez obtenir :

```
{ nom: 'Brice', age: 30, enEmploi: true }
```

Remarque 1 La syntaxe littérale ressemble au **JSON**. Mais il y a une différence importante.

En **JSON**, les noms des propriétés doivent obligatoirement être entre **doubles quotes** alors que dans la **syntaxe littérale** c'est facultatif.

Remarque 2 Pour créer un **objet vide**, deux solutions sont possibles :

```
const p1 = new Object()
```

ou bien :

```
const p1 = {}
```

Absence de typage des données

En **Java**, chaque variable a un type bien défini (entier, chaîne, etc.) et ne peut recevoir que des données compatibles avec son type.

En **JavaScript**, les variables n'ont pas de type fixe. Au cours d'un même traitement, une variable peut recevoir des données de types complètement différents.

Par exemple, il est possible d'écrire le script suivant :

```
const p2 = {
  nom: 'Brice',
  age: 30,
  enEmploi: true,
}
console.log(p2)

p2.age = 'trente'
p2.enEmploi = 'oui'
console.log(p2)
```

Licence Creative Commons BY-SA 4.0 - AMBLARD Emmanuel 2026

Il produit le résultat suivant :

```
{ nom: 'Brice', age: 30, enEmploi: true }
{ nom: 'Brice', age: 'trente', enEmploi: 'oui' }
```

Déclaration des variables

Utilisation de variables non déclarées

JavaScript est un langage très laxiste. Dans sa version d'origine, il permet d'utiliser une variable sans l'avoir déclarée au préalable.

Donc, même si on n'a jamais déclaré la variable **x**, on peut écrire :

```
x = 12
console.log(x)
```

Cette pratique est **à éviter absolument**. Elle produit un code qui est beaucoup moins prévisible, donc **source d'erreurs**. Mais également, elle **dégrade les performances** lors de l' exécution.

Il faut donc **toujours déclarer** une variable avant de l'utiliser.

Pour être sûr de le faire, on peut mettre la directive suivante au début du script **JavaScript**, :

```
'use strict'
```

Lors de l'exécution, **Node.js** signalera une **erreur** s'il rencontre une variable non déclarée.

var

Historiquement, on pouvait déclarer une variable avec le mot-clé **var**.

```
var x
var y = 30
```

Mais, il s'agit d'un vestige **du passé** et il ne faut **plus** l'utiliser, car les variables ainsi déclarées n'ont **pas** de portée. Elles sont donc accessibles même en dehors du bloc où elles ont été déclarées. Ce qui est une **mauvaise pratique**.

Pour déclarer une variable, il faut **toujours** utiliser les mots-clés **const** ou **let**.

const

const est le mot-clé à utiliser **systématiquement** par défaut.

Il permet de déclarer une variable qui ne pourra pas être réassignée. Autrement dit, son contenu ne pourra pas être modifié.

Sa valeur doit donc être fixée au moment de la déclaration.

```
const x = 18                // nombre
const p1 = new Object()     // Objet
const notes = []            // Tableau
```

Licence Creative Commons BY-SA 4.0 - AMBLARD Emmanuel 2026

Cela signifie que, dans la suite du code, il sera **interdit** d'écrire des instructions du type :

```
x = 9                          // Erreur
p1 = null                      // Erreur
notes = [11, 14, 8]            // Erreur
```

En revanche, si la variable représente un objet ou un tableau, il est tout à fait possible de modifier les propriétés de l'objet ou le contenu du tableau.

```
p1.nom = 'Aline'
notes.push(18)
```

let

let permet de déclarer une variable dont le contenu pourra être réassigné.

On peut donc l'utiliser pour déclarer une variable sans lui donner de contenu initial.
La valeur de cette variable sera **undefined**

```
let a
let b = 'bonjour'
console.log(a,b);
```

Bonne pratique

Par défaut, lorsque vous déclarez une variable, utilisez toujours **const**.

Ensuite, si vous vous rendez compte que son contenu a besoin d'être modifié, vous pourrez toujours remplacer **const** par **let**.

Et vous verrez que ça n'arrive pas si souvent.

Licence Creative Commons BY-SA 4.0 - AMBLARD Emmanuel 2026