

301 – Introduction

node.js



WIK-NJS301

Durée estimée : 6h

Intervenant : Jeremy Trufier <jeremy@wikodit.fr>



WIK-NJS

Programme nodeJS

- 301 – Introduction**
- 302 – Scripting et CLI**
- 303 – Express.js**
- 304 – MVC Frameworks**
- 305 – Tests unitaires**

1XX – 1er année (pas de notion d'algorithmie)
2XX – 2e année (notions d'algorithmie succinctes)
3XX – 3e année (rappels et pratique, niveau moyen d'algorithmie)
4XX – 4e année (concepts avancés, niveau avancé d'algorithmie)
5XX – 5e année (approfondissement experts)

Au préalable

Qui l'utilise ?

ebay

NETFLIX

yammer

UBER

The New York Times



Linked in



Préparation

Tout d'abord, créez un dossier pour les cours nodejs

- Ouvrir un terminal ou PowerShell
- Naviguez vers ce dossier (avec la commande ``cd dossier1/sousdossier/...``)
- Créez un dossier appelé njs-301 (avec la commande ``mkdir njs-301``)
- Naviguez vers ce dossier (``cd njs-301``)

Tout ce qui se passera dans ce cours
devra se passer dans ce dossier "njs-301" dans le terminal

Avec Docker

Téléchargez et installez Docker :

<https://www.docker.com/community-edition#/download>

- Ouvrir un terminal ou PowerShell

```
$ docker run --rm -ti -v $(pwd):/srv -w /srv node:alpine node  
> console.log('hello')  
hello  
undefined
```

sous windows remplacez : \$(pwd) par %cd%

--rm Supprime le container Docker après utilisation

-ti Attach to TTY and keep STDIN open

-v folderPath:/srv Monte le repertoire folderPath de l'hôte à l'emplacement /srv du container

-w /srv Défini /srv en tant que repertoire de travail dans le container

node:alpine Image Docker à utiliser (alpine est un Linux très léger)

node La commande à lancer dans le container (sh fonctionne aussi pour un accès au container)

Téléchargement

Ignorer ce slide si utilisation de Docker

- <https://nodejs.org>
- Version "Current"



Dans Powershell ou le Terminal

```
$ node  
> console.log('hello')  
hello  
undefined
```

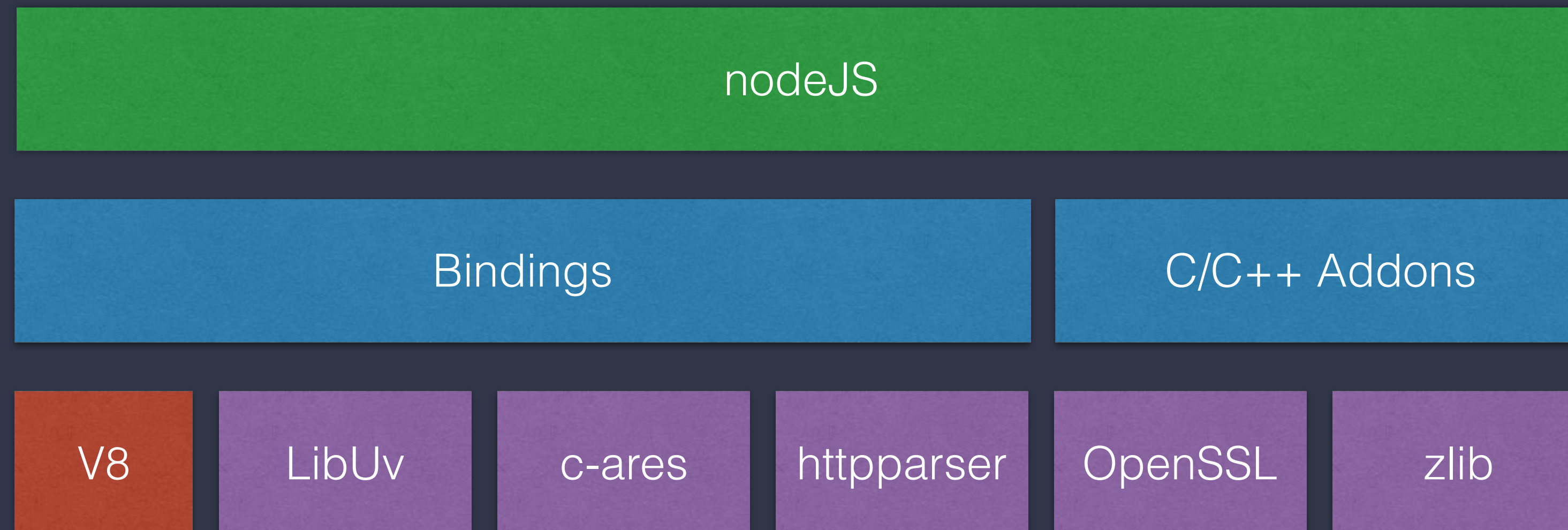
Installation Windows

Utilisateurs Docker, OSX & Linux : Ignorez ce slide :)

Utilisateurs Windows :

- Ouvrir PowerShell avec clic droit + ouvrir en tant qu'administrateur
 - exécuter la commande suivante : `npm install -g windows-build-tools`
- Fermer PowerShell, et réouvrir en utilisateur standard
 - exécuter : `npm install sqlite3`
- Si ça ne fonctionne pas et qu'une erreur rouge concernant "CL.exe" apparaît
 - Si vous avez Visual Studio, l'ouvrir, créer un projet Visual C++ pour télécharger les outils Visual C++ 2015
 - Réessayez
 - Sinon, ouvrir "Invite de commande développeur" et reessayez
- Sinon... essayer ce qui est en note de ce slide
- Sinon... utilisez une VM

La mécanique



nodeJS s'appuie un maximum sur des fonctions systèmes écrites en C/C++

Les avantages

- Asynchrone
- Événementiel
- Javascript
- Communauté
- Gros volumes de connections



CLI

Command Line Interface

Commande `node`

Terminal ou PowerShell

Evaluation de script

Mode interactif

Code javascript

Output

Return

Pour quitter

Execution d'un fichier JS

```
$ node -e "console.log('hello')"
```

```
$ node
```

```
> console.log('hello')
```

```
hello
```

```
undefined
```

```
> 1+3
```

```
4
```

```
> .exit
```

```
$ node ./hello.js
```

```
Hello World
```

```
$
```

Un serveur Web

```
const http = require('http')  
  
http.createServer((req, res) => {  
  res.end('Bonjour à tous !')  
}).listen(8080)
```

JavaScript ES6, ES7, ES8

Notions & Rappels

Les bases

constante `const hello = 'Bonjour'`

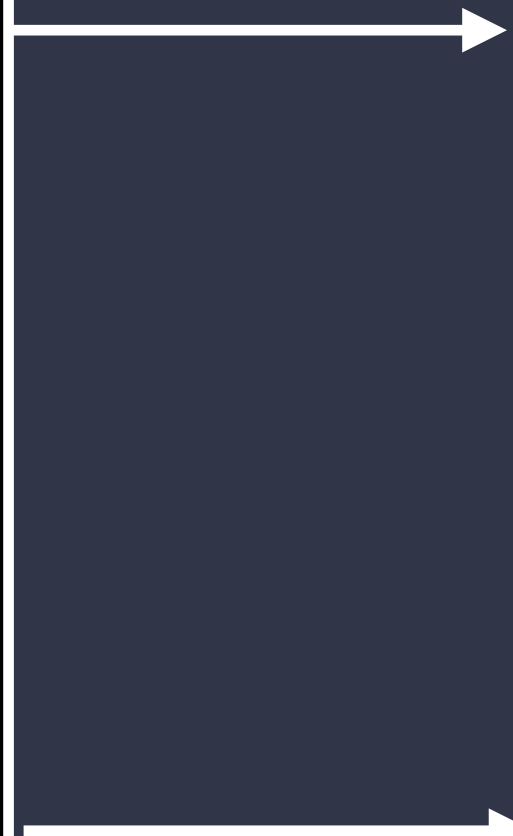
variable et objet `let currentUser = {
 firstname: 'Jean',
 lastname: 'Dubois'
}`

fonction nommée `function welcome (person) {
 return `${hello} ${person.firstname} ${person.lastname}`
}`

`console.log(welcome(currentUser))`

Astuces

```
const user = {  
  firstname: 'Jean',  
  lastname: 'Dubois',  
  age: 30,  
}  
  
const { firstname, age } = user  
console.log(firstname, age)  
  
// Mais aussi:  
const city = 'Bordeaux'  
const address = {  
  city,  
  zipcode: '33000'  
}  
console.log(address.city)
```



```
const firstname = user.firstname  
const age = user.age
```

```
const address = {  
  city: city,  
  zipcode: '33000'  
}
```

(Fonctionne aussi avec les Array)

Les types

- boolean
- number
- string
- undefined
- null
- object
- symbol (ES6)

Que remarquez-vous ?

```
typeof 'une chaine'  
typeof true  
typeof undefined  
typeof { name: 'Jean' }  
typeof [0, 1, 2, 3]  
typeof null  
typeof function() {}
```

Les conditions

Faible égalité

```
let a = '2'

if (a == 2) {
  console.log('a est égal à 2')
} else {
  console.log('a n\'est pas égal à 2')
}
```

Strict égalité : le type est pris en compte

```
let a = '2'

if (a === 2) {
  console.log('a est le chiffre 2')
} else {
  console.log('a n\'est pas le chiffre 2')
}
```

Condition ternaire

```
let a = '2'
console.log(a == '2' ? 'a égal 2' : 'a n\'est pas égal à 2')
```

Les conditions

Opérateurs de comparaison

<	Inférieur à
>	Supérieur à
<=	Inférieur ou égal à
>=	Supérieur ou égal à
==	Égalité faible
===	Égalité stricte
!=	Inégalité faible
!==	Inégalité stricte

Opérateurs logiques

&&	ET logique
	OU logique
!	NON logique

```
let a = '2'

if (typeof a !== undefined && a !== null) {
  if (a == 1) {
    console.log('a est 1')
  } else if (a >= 5 || a === 10) {
    console.log('a est soit 8, soit supérieur à 2')
  } else {
    console.log('a n\'est pas entre 1 et 5')
  }
} else {
  console.log('a est null ou non défini')
}
```

Les fonctions

```
// Fonction nommée (hoisted)
console.log(sum(7, 4)) // => 11
function sum(x, y) {
  return x + y
}
```

```
// Fonction anonyme
console.log(diff(7, 4)) // => ERREUR
const diff = function (x, y) {
  return x - y
}
console.log(diff(7, 4)) // => 3
```

```
// Fonction anonyme avec flèche (conserve le scope)
const times = (x, y) => { return x * y }
```

```
// Fonction anonyme avec return implicite (conserve le scope)
const times = (x, y) => x * y
```

Si un seul paramètre, les parenthèses des fonctions fléchées sont facultatifs

Les objets et tableaux

```
// Les objets
const user = {
  firstname: 'Jean'
  lastname: 'Dubois'
}

const user2 = user
user2.firstname = 'Marc'
console.log(user.firstname) // => Marc
console.log(user['firstname']) // => Marc

// Les tableaux
const languages = [ 'fr', 'en', 'es' ]
user.language = languages[0]

// Le résultat ??
console.log(user)
```

Les itérations (for)

```
const fruits = ['apple', 'orange', 'strawberry', 'blueberry']
const l = fruits.length
for (let i = 0; i < l ; i++) {
  console.log(1, fruits[i])
}
```

```
for (let i = fruits.length; i--; ) {
  console.log(2, fruits[i])
}
```

```
for (let i = 0; i < l ; i++) {
  if (fruits[i] === 'apple') { continue }
  console.log(3, fruits[i])
  if (fruits[i] === 'strawberry') { break }
}
```

Les itérations (while)

```
let found = false
let i = 0
while (!found) {
  if (fruits[i] === 'strawberry') {
    found = true
  }
  console.log(4, fruits[i])
  i++
}
```

La portée des variables (SCOPE)

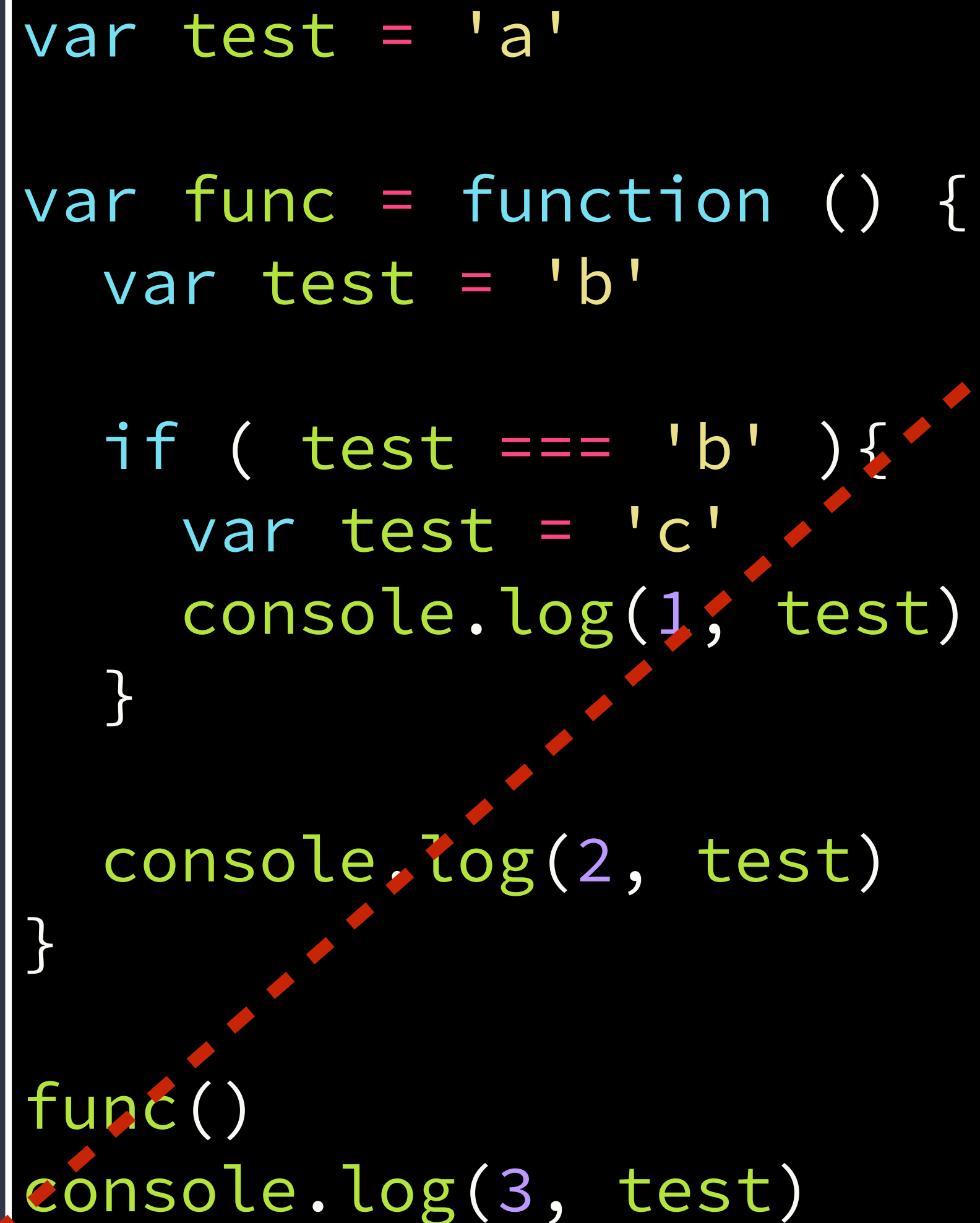
```
var test = 'a'

var func = function () {
  var test = 'b'

  if ( test === 'b' ) {
    var test = 'c'
    console.log(1, test)
  }

  console.log(2, test)
}

func()
console.log(3, test)
```



```
let test2 = 'a'

const func2 = function() {
  let test2 = 'b'

  if ( test2 === 'b' ) {
    let test2 = 'c'
    console.log(1, test2)
  }

  console.log(2, test2)
}

func2()
console.log(3, test2)
```


Careful of SCOPE

```
const l = fruits.length
for (var i = 0; i < l; i++) {
  process.nextTick(() => {
    console.log(i, fruits[i])
  })
}
```

```
const l = fruits.length
for (let i = 0; i < l; i++) {
  process.nextTick(() => {
    console.log(i, fruits[i])
  })
}
```

process.nextTick : Décale l'exécution de la fonction en paramètre à la prochaine boucle de l'event loop

NE JAMAIS UTILISER LE CODE DE GAUCHE

TD : Le jeu du plus ou moins

Memo & Tips

```
// Initialisation
const rl = require('readline').createInterface({
  input: process.stdin, output: process.stdout
})

// Pose une question à l'utilisateur
rl.question('Question ?', (answer) => {
  process.stdout.write("Tu as répondu : " + answer + "\n")
})

// Retourne un entier aléatoire entre 0 et 10
Math.floor(Math.random() * 10)

// RegExp qui retourne true si input contient 1 à 3 chiffres
/^\\d{1,3}$/.test(input)

// Quitte le programme
process.exit()
```

Au lancement, le script choisit un nombre de 0 à 999.

Le but du jeu est de trouver ce nombre.

À chaque mauvaise réponse, le script indique simplement si le nombre est supérieur ou inférieur.

Langage Prototypé

Langage prototypé

```
// foo hérite du prototype de Object
foo = new Object()
foo.bar = 'toto'

// le code ci dessus est l'équivalent de
foo = { bar: 'toto' }
```

- Tout type non primitif est un objet
- Objets ont des prototypes
- Objets héritent les propriétés de leur prototype
- L'opérateur `new` crée une instance d'un objet en appelant la méthode `constructor` du prototype

```
// Création d'un prototype avec un constructeur
const User = function (firstname, lastname) {
  this.firstname = firstname
  this.lastname = lastname
}

// Ajout d'une méthode au prototype de Person
User.prototype.name = function() {
  return this.firstname + ' ' + this.lastname
}

// Utilisation de notre classe
user = new User('Jean', 'Bon')
user.name() // => "Jean Bon"
```

Propriétés avancées

Définition de propriétés

```
const foo = {}

Object.defineProperty(foo, 'prop1', {
  value: 'Valeur 1',
  configurable: true,
  enumerable: false,
  writable: false
})

Object.defineProperty(foo, 'prop2', {
  value: 'Valeur 2',
  configurable: false,
  enumerable: true,
  writable: true
})

// prop1 n'est pas énumérable
console.log(foo)
console.log(foo.prop1)

// prop1 ne peut être modifié
foo.prop1 = 'New Valeur 1'
foo.prop2 = 'New Valeur 2'
console.log(foo.prop1, foo.prop2)

// prop2 n'est pas configurable
delete foo.prop1 // Working
delete foo.prop2 // Not possible
```

Getter / Setter

```
const bar = {}
let barValue = 3

Object.defineProperty(bar, 'value', {
  enumerable: true,
  get: function () { return barValue },
  set: function (val) { barValue = val }
})

console.log(bar)

bar.value = 10
console.log(barValue)
```

Plus de fun

```
// Lister les propriétés
console.log(Object.getOwnPropertyNames(foo))

// Récupérer des infos sur une propriété
console.log(Object.getOwnPropertyDescriptor(foo, 'prop1'))

// Sceler un objet
const qux = { test: 10 }
Object.seal(qux)

qux.newProp = 'kek' // not working
qux.test = 20 // still working
console.log(qux)
delete qux.test // not working

// Verrouiller un objet
const baz = { test: 10 }
Object.freeze(baz)

baz.newProp = 'kek' // not working
baz.test = 20 // not working
console.log(baz)
```

Plus facile avec les classes ES6

```
class Employee {
  constructor (firstname, lastname) {
    this.firstname = firstname
    this.lastname = lastname
  }

  static createFromName (name) {
    const p = new Employee()
    p.name = name
    return p
  }

  get name () {
    return `${this.firstname} ${this.lastname}`
  }

  set name (name) {
    [ this.firstname, this.lastname ] = name.split(' ')
  }

  sayHello () {
    return `Bonjour ${this.firstname} !`
  }
}

let jeanEmployee = new Employee('Jean', 'Bon')
let emilieEmployee = Employee.createFromName('Emilie Bond')
```

Les classes ES6 : Héritage

```
class Boss extends Employee {  
  get name () {  
    return `Mr. ${super.name}`  
  }  
  
  stressOut (person) {  
    return `Plus vite que ça ${person.firstname} !!`  
  }  
}  
  
let michelBoss = new Boss('Michel', 'Dubois')  
michelBoss.stressOut(jeanEmployee) // "Plus vite que ça Jean !!"  
michelBoss.name // "Mr. Michel Dubois"
```


Careful of SCOPE

```
class Hello {  
  constructor (message) {  
    this.message = message  
  }  
  
  waitAndLogMessage () {  
    setTimeout(() => {  
      console.log(this.message)  
    }, 1000)  
  }  
  
  waitAndTryToLogMessage () {  
    setTimeout(function () {  
      console.log(this.message || 'No message to  
log...')  
    }, 1000)  
  }  
}  
  
const hello = new Hello('Hey !')  
hello.waitAndLogMessage()  
hello.waitAndTryToLogMessage()
```

Lorsqu'on utilise une fonction fléchée anonyme, le scope est conservé donc **this** correspond bien à celui de l'instance de **Hello**.

Par contre avec une fonction anonyme, le scope n'est pas conservé, et le **this** correspond donc au contexte de la méthode qui appelle la fonction (en l'occurrence c'est le **setTimeout** qui décide ce que sera le **this**).

Langage prototypé

```
// foo hérite du prototype de Object
foo = new Object()
foo.bar = 'toto'

// le code ci dessus est l'équivalent de
foo = { bar: 'toto' }
```

- Tout type non primitif est un objet
- Objets ont des prototypes
- Objets héritent les propriétés de leur prototype
- L'opérateur `new` crée une instance d'un objet en appelant la méthode `constructor` du prototype

```
// Création d'un prototype avec un constructeur
const Person = function (firstname, lastname) {
  this.firstname = firstname
  this.lastname = lastname
}

// Ajout d'une méthode au prototype de Person
Person.prototype.name = function() {
  return this.firstname + ' ' + this.lastname
}

// Utilisation de notre classe
p = new Person('Jean', 'Bon')
p.name() // => "Jean Bon"
```

Propriétés avancées

Définition de propriétés

```
const foo = {}

Object.defineProperty(foo, 'prop1', {
  value: 'Valeur 1',
  configurable: true,
  enumerable: false,
  writable: false
})

Object.defineProperty(foo, 'prop2', {
  value: 'Valeur 2',
  configurable: false,
  enumerable: true,
  writable: true
})

// prop1 n'est pas énumérable
console.log(foo)
console.log(foo.prop1)

// prop2 n'est pas configurable
delete foo.prop1 // Working
delete foo.prop2 // Not possible

// prop1 ne peut être modifié
foo.prop1 = 'New Valeur 1'
foo.prop2 = 'New Valeur 2'
console.log(foo.prop1, foo.prop2)
```

Getter / Setter

```
const bar = {}
let barValue = 3

Object.defineProperty(bar, 'value', {
  enumerable: true,
  get: function () { return barValue },
  set: function (val) { barValue = val }
})

console.log(bar)

bar.value = 10
console.log(barValue)
```

Plus de fun

```
// Lister les propriétés
console.log(Object.getOwnPropertyNames(foo))

// Récupérer des infos sur une propriété
console.log(Object.getOwnPropertyDescriptor(foo, 'prop1'))

// Sceler un objet
const qux = { test: 10 }
Object.seal(qux)

qux.newProp = 'kek' // not working
qux.test = 20 // still working
console.log(qux)
delete qux.test // not working

// Verrouiller un objet
const baz = { test: 10 }
Object.freeze(baz)

baz.newProp = 'kek' // not working
baz.test = 20 // not working
console.log(baz)
```

Plus facile avec les classes ES6

```
class Employee {
  constructor (firstname, lastname) {
    this.firstname = firstname
    this.lastname = lastname
  }

  static createFromName (name) {
    const p = new Employee()
    p.name = name
    return p
  }

  get name () {
    return `${this.firstname} ${this.lastname}`
  }

  set name (name) {
    [ this.firstname, this.lastname ] = name.split(' ')
  }

  sayHello () {
    return `Bonjour ${this.firstname} !`
  }
}

let jeanEmployee = new Employee('Jean', 'Bon')
let emilieEmployee = Employee.createFromName('Emilie Bond')
```

Les classes ES6 : Héritage

```
class Boss extends Employee {  
  get name () {  
    return `Mr. ${super.name}`  
  }  
  
  stressOut (person) {  
    return `Plus vite que ça ${person.firstname} !!`  
  }  
}  
  
let michelBoss = new Boss('Michel', 'Dubois')  
michelBoss.stressOut(jeanEmployee) // "Plus vite que ça Jean !!"  
michelBoss.name // "Mr. Michel Dubois"
```

Careful of SCOPE

```
class Hello {  
  constructor (message) {  
    this.message = message  
  }  
  
  waitAndLogMessage () {  
    setTimeout(() => {  
      console.log(this.message)  
    }, 1000)  
  }  
  
  waitAndTryToLogMessage () {  
    setTimeout(function () {  
      console.log(this.message || 'No message to  
log...')  
    }, 1000)  
  }  
}  
  
const hello = new Hello('Hey !')  
hello.waitAndLogMessage()  
hello.waitAndTryToLogMessage()
```

Lorsqu'on utilise une fonction fléchée anonyme, le scope est conservé donc **this** correspond bien à celui de l'instance de **Hello**.

Par contre avec une fonction anonyme, le scope n'est pas conservé, et le **this** correspond donc au contexte de la méthode qui appelle la fonction (en l'occurrence c'est le **setTimeout** qui décide ce que sera le **this**).

TD : Le jeu du plus ou moins

Memo & Tips

```
// Initialisation
const rl = require('readline').createInterface({
  input: process.stdin, output: process.stdout
})

// Pose une question à l'utilisateur
rl.question('Question ?', (answer) => {
  process.stdout.write("Tu as répondu : " + answer + "\n")
})

// Retourne un entier aléatoire entre 0 et 10
Math.floor(Math.random() * 10)

// RegExp qui retourne true si input contient 1 à 3 chiffres
/^\\d{1,3}$/.test(input)

// Quitte le programme
process.exit()
```

Au lancement, le script choisit un nombre de 0 à 999.

Le but du jeu est de trouver ce nombre.

À chaque mauvaise réponse, le script indique simplement si le nombre est supérieur ou inférieur.