

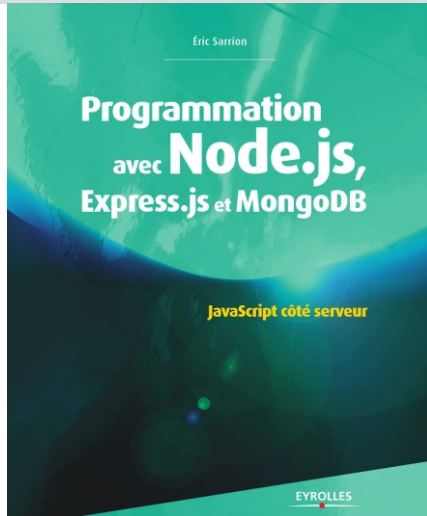
# Programmation avancée

## Node.js - MongoDB

Sylvie Trouilhet - Jean-Marie Pecatte

S4 – UE2

# Bibliographie



*Programmation avec Node.js, Express.js  
et MongoDB*

Éric Sarrion

Eyrolles, Paris, septembre 2014

*Node.js*

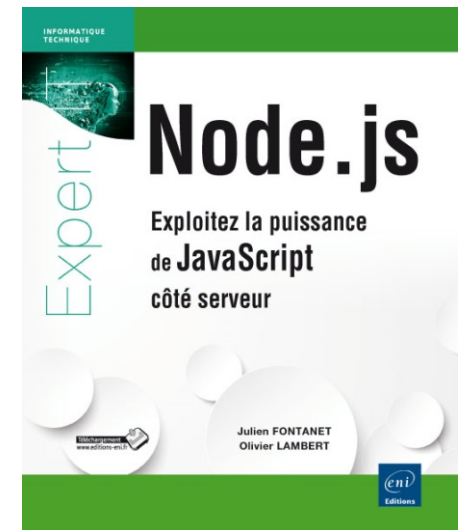
*Exploitez la puissance de JavaScript côté serveur*

Julien Fontanet et Olivier Lambert

Éditions ENI, St Herblain, avril 2015

<https://nodejs.org/en/>

Site officiel de Node.js (consulté le 07/01/2018)



partie Node.js = 8 h TD – 8 h TP

1. Notions préalables
2. Node.js : généralités
3. Premier exemple
4. Le module standard **events**
5. Exercice
6. Compléments sur les modules
7. L'objet **module**

# 1. Notions préalables : fonctionnement d'un programme

- Appel **synchrone** : le traitement se fait au moment de la demande  
→ asynchrone : le traitement ne se fait pas au moment de la demande
- Appel **bloquant** : le programme attend tant que le traitement n'est pas complètement terminé  
→ non bloquant : le programme n'attend pas

# 1. Notions préalables : fonctionnement d'un programme

```
<html>
<head>
  <title>Premier essai</title>
</head>
<body>
  <p id="p1"></p>
  <script>
    const f=function(i) { return ++i;}

    let r=f(0);
  </script>
</body>
</html>
```

➔ **f(0) ?**

# 1. Notions préalables : fonctionnement d'un programme

```
<html>
<head>
  <title>Premier essai</title>
</head>
<body>
  <p id="p1"></p>
  <script>
    const f=function(i) { return ++i;}

    let r=f(0);
  </script>
</body>
</html>
```

programme  $\neq$  processus

Gestionnaire de tâches - Google Chrome

Tâche	Mémoire	UC	Réseau	ID du processus
• Navigateur	32 600 Ko	0.8	0	10316
• GPU	15 776 Ko	0.4	0	376
• Onglet : Developer Tools - ...	32 608 Ko	0.0	0	1304
• Onglet : Premier essai	9 632 Ko	0.0	0	5640

Arrêter le processus

# 1. Notions préalables : fonctionnement d'un programme

**processus** : programme en cours d'exécution auquel est associé un environnement processeur et un environnement mémoire.

Chaque processus possède un **espace d'adressage**, c'est-à-dire un ensemble d'adresses mémoires dans lesquelles il peut lire et écrire.

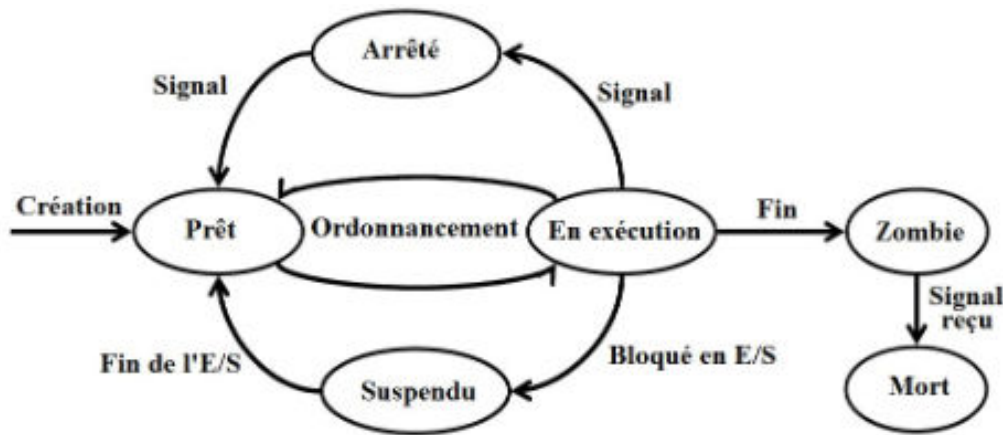
Cet espace est divisé en trois parties :

- le segment de texte (le code du programme)

- le segment de données (les variables)

- la pile

# 1. Notions préalables : fonctionnement d'un programme



Cycle de vie d'un processus (source : <http://www.uqac.ca/>)



# 1. Notions préalables : fonctionnement d'un programme

```
<html>
<head>
  <title>Premier essai</title>
</head>
<body>
  <p id="p1"></p>
  <script>
    document.getElementById("p1").addEventListener("click", f);
    function f(evt) { console.log(evt);};
  </script>
</body>
</html>
```



`addEventListener("click", f)?`

# 1. Notions préalables : fonctionnement d'un programme

Comment effectuer plusieurs traitements  
**simultanément** en mode bloquant ?

# 1. Notions préalables : fonctionnement d'un programme

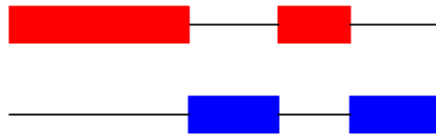
- traitement séquentiel = un traitement après l'autre



Exécution séquentielle sans thread

→ comment effectuer plusieurs traitements en même temps ?

- traitement concurrent

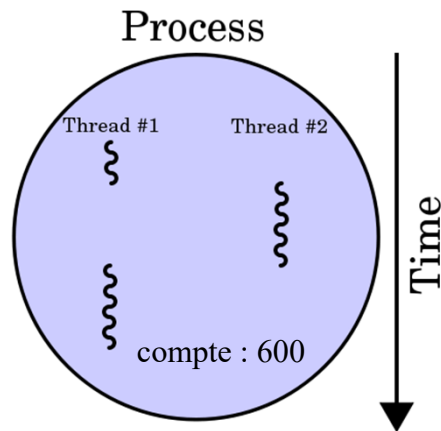


Exécution concurrente (non-parallèle) avec threads

→ **pseudo-parallélisme** : un seul processeur

- \* plusieurs processus
- \* un processus et des **threads**  
(même espace d'adressage)

# 1. Notions préalables : fonctionnement d'un programme



Thread #1  
si (compte > 300)  
{prélever(300)}

Thread #2  
si (compte > 500)  
{prélever(500)}

Source : Par I, Cburnett, CC BY-SA 3.0,  
<https://commons.wikimedia.org/w/index.php?curid=2233446>

# 1. Notions préalables : fonctionnement d'un programme

Comment effectuer plusieurs traitements

**simultanément** en mode bloquant ?

☞ gérer la concurrence

**simultanément** en mode non bloquant (CPS) ?

☞ pas de concurrence mais gérer les **événements**

# 1. Notions préalables : fonctionnement d'un programme

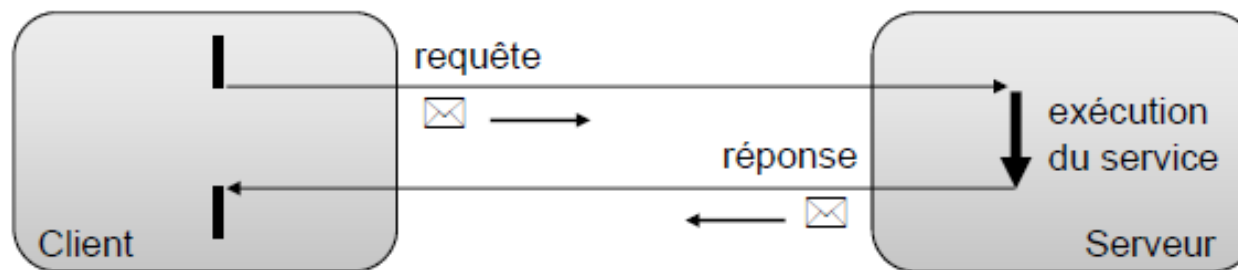
- **Boucle d'événements** : il s'agit d'une file de type FIFO.  
Une callback s'inscrit pour le ou les événements qui l'intéressent.  
La boucle d'événements attend les événements et lance les abonnés.  
Les callbacks ne sont pas interrompues et sont généralement très courtes.

# Event-driven programming versus threads ?

- Métaphore du fast-food

# En fonction des entrées/sorties et du temps d'exécution

- (source : [www.gipsa-lab.grenoble-inp.fr/~christian.bulfone](http://www.gipsa-lab.grenoble-inp.fr/~christian.bulfone) )





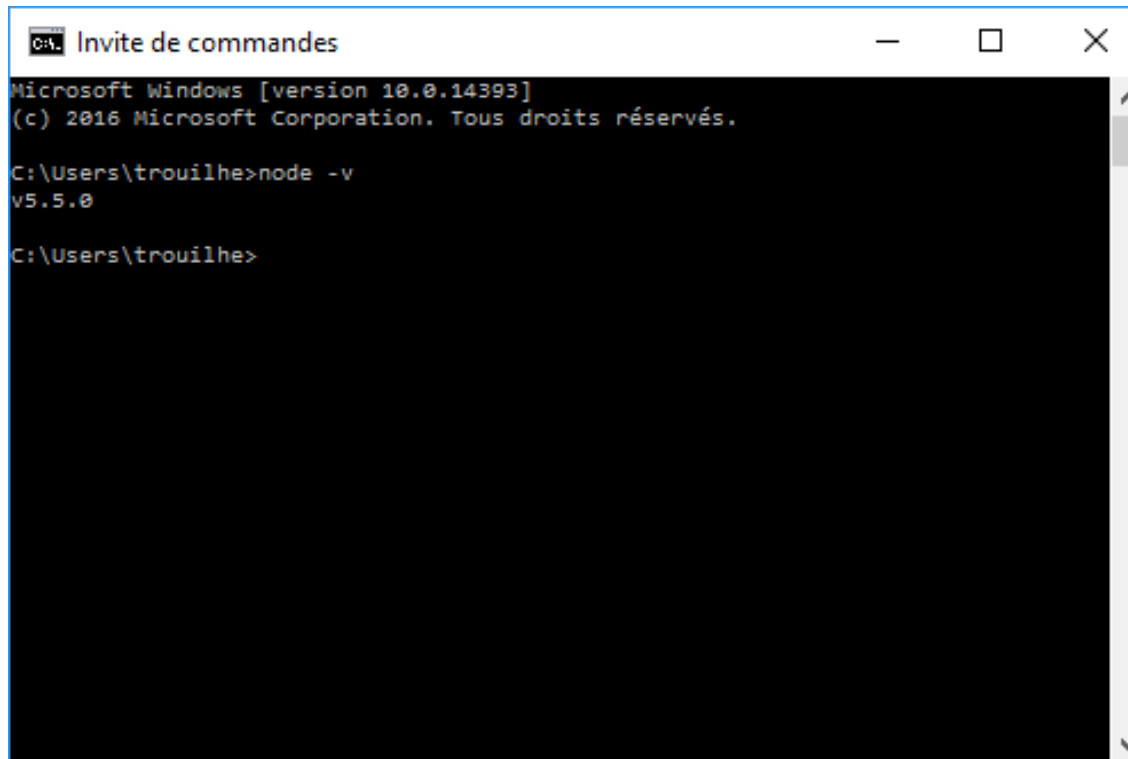
# Programmation avec Node.js

## 2. Node.js

- Environnement d'exécution autonome, pour des fichiers Javascript.
- Permet de développer des applications *standalone*
- Installation : <https://nodejs.org/en/>
- Exécution (dans une fenêtre de commande) :  
**`node <nomFichier.js>`**

## 2. Node.js

**node -v** : donne le numéro de version



```
Microsoft Windows [version 10.0.14393]
(c) 2016 Microsoft Corporation. Tous droits réservés.

C:\Users\trouilhe>node -v
v5.5.0

C:\Users\trouilhe>
```

## 2. Node.js

5 . 5 . 0

0 : **révision** =

correction de bugs (pas de perte de compatibilité)

5 : **version mineure** = changements légers

5 : **version majeure** =

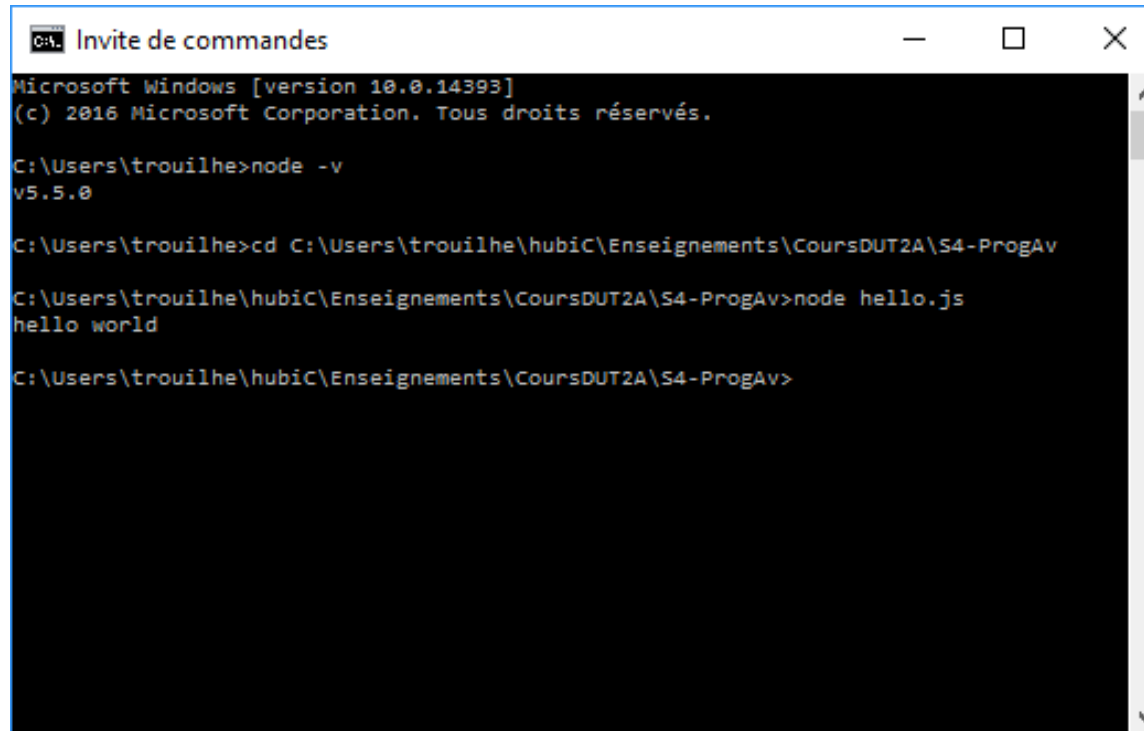
gros changements (perte de compatibilité)

## 2. Node.js

- Fichier `hello.js`

```
console.log("hello world");
```

- `node hello.js`



```
C:\Users\trouilhe>node -v
v5.5.0

C:\Users\trouilhe>cd C:\Users\trouilhe\hubic\Enseignements\CoursDUT2A\S4-ProgAv

C:\Users\trouilhe\hubic\Enseignements\CoursDUT2A\S4-ProgAv>node hello.js
hello world

C:\Users\trouilhe\hubic\Enseignements\CoursDUT2A\S4-ProgAv>
```

## 2. Node.js

- Variables globales :  
définies par ECMAScript 5 : **Boolean**, **Date**...  
  
définies par Node :  
**global** = contexte racine du fichier courant  
**process** = processus courant  
méthodes : **exit(<code>)**  
**hrtime(<depart>)**  
**console** = affichage sur la sortie standard  
**module** = module courant

## 2. Node.js : inclusion de modules

- module = ensemble de fichiers javascript
- 3 types de modules :
  - modules standards (*node core*)
  - modules créés par l'utilisateur
  - modules externes
- Pour utiliser un module : `[module].require(<nomModule>)`

## 2. Node.js : module standard

- Quelques modules :
- `http` : pour les fonctionnalités http
- `url` : pour *parser* les url
- `events` : pour la gestion des événements
- `async` : pour gérer les fonctions de callback
- `node-inspector` et `debug`: pour le *deboggage*
- Instruction `require("http")`  
retourne un **objet** encapsulant les fonctionnalités du module c'est-à-dire des **méthodes** à appliquer à l'objet



### 3. Exemple : créer un serveur HTTP à l'écoute sur le port 4000

- Documentation : <https://nodejs.org/api/http.html>
- méthode `createServer([f])`  
paramètre : `f`, fonction de callback appelée lorsqu'un accès est effectué sur le serveur  
retour : objet de type `server`
- méthode `listen(port)` : boucle d'écoute des requêtes sur `port`

NB : pour arrêter l'exécution `ctrl+c`

### 3. Exemple : créer un serveur HTTP à l'écoute sur le port 4000

- Fichier useHttp.js :

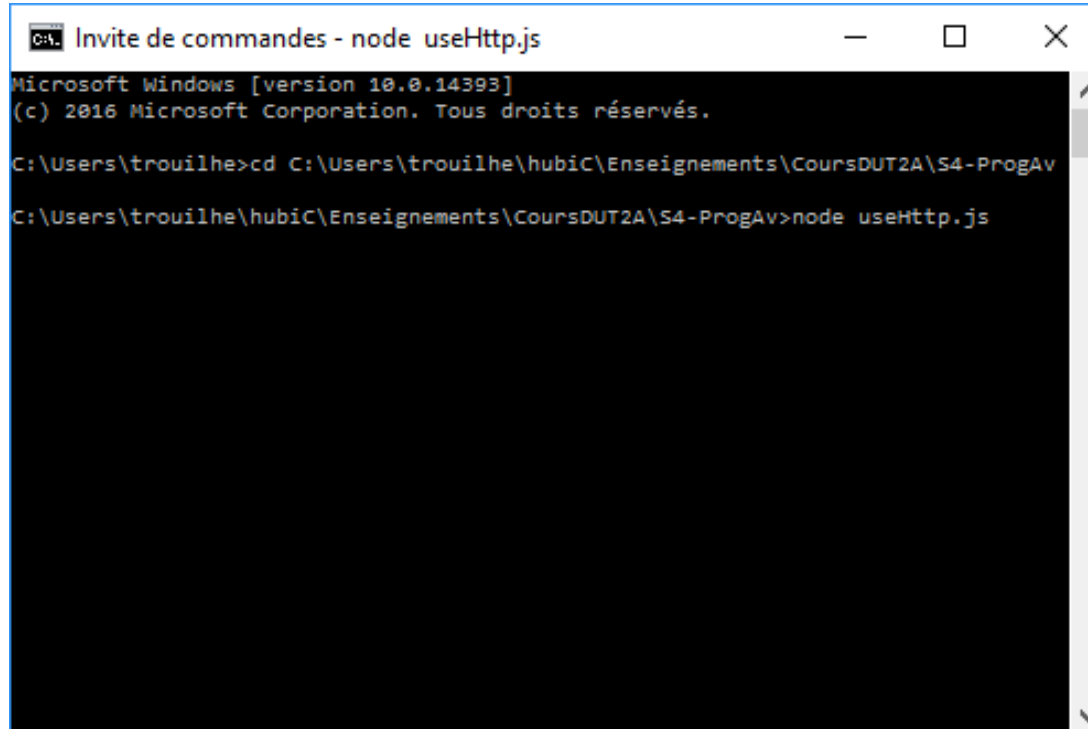
```
const http = require('http');

const server = http.createServer(f);
server.listen(4000);

// fonction de callback
// paramètres : request, requete envoyee au serveur
//                response, réponse retournée
function f(request, response)
{
    response.writeHead(200);
    response.write("Hello world !");
    response.end();
}
```

### 3. Exemple : créer un serveur HTTP à l'écoute sur le port 4000

- Lancer l'exécution du fichier  
`node useHttp.js`

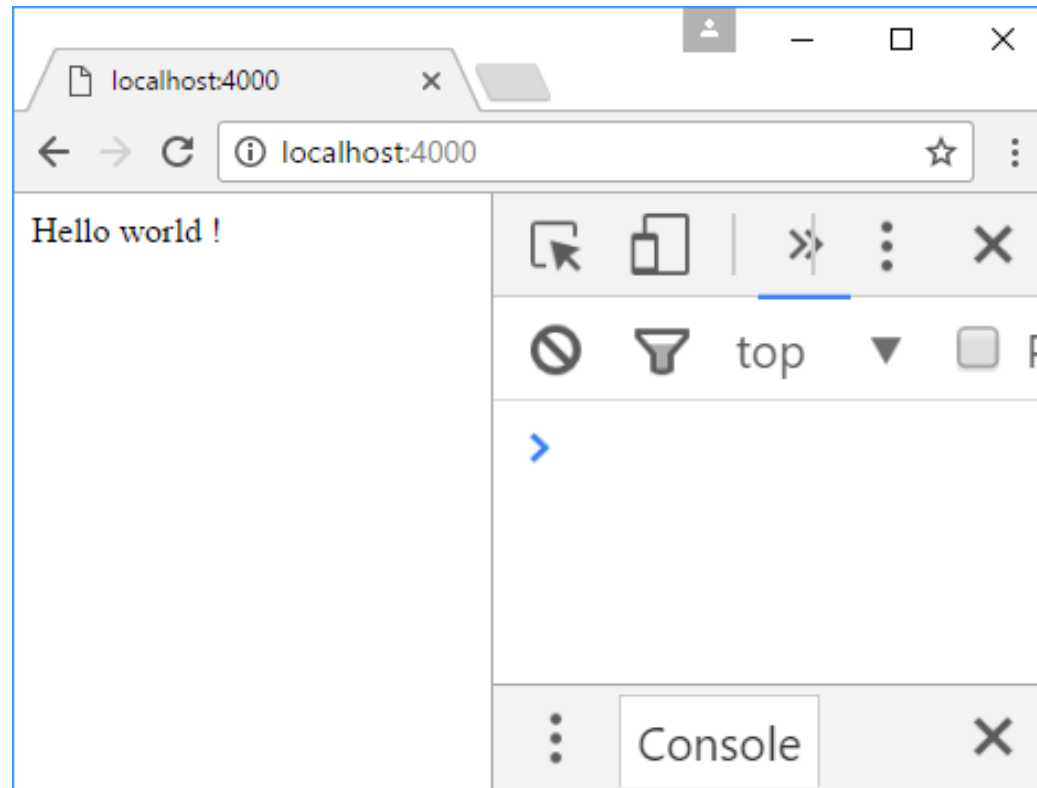


```
Microsoft Windows [version 10.0.14393]
(c) 2016 Microsoft Corporation. Tous droits réservés.

C:\Users\trouilhe>cd C:\Users\trouilhe\hubiC\Enseignements\CoursDUT2A\S4-ProgAv
C:\Users\trouilhe\hubiC\Enseignements\CoursDUT2A\S4-ProgAv>node useHttp.js
```

### 3. Exemple : créer un serveur HTTP à l'écoute sur le port 4000

- Lancer l'exécution du fichier  
`node useHttp.js`
- Dans un navigateur effectuer une requête sur le port 4000



### 3. Exemple : créer un serveur HTTP à l'écoute sur le port 4000

- La fonction de callback  
a **toujours 2 paramètres**

un objet de type `http.IncomingMessage`  
regarder l'attribut `url` :

un objet de type `http.ServerResponse`  
regarder l'attribut `statusCode` :  
regarder l'attribut `statusMessage` :

### 3. Exemple : créer un serveur HTTP à l'écoute sur le port 4000

- Amélioration: le programme ne répond que si la page demandée est `index.html`
- En cas de nom différent, le programme retourne le code 404

### 3. Exemple : créer un serveur HTTP à l'écoute sur le port 4000

- utiliser le module `url`
- Documentation <https://nodejs.org/api/url.html>
- méthode `parse()`  
paramètre : l'url (passée dans la requête)  
retour : un objet de type `URL`
- attribut `pathname`

### 3. Réponse : créer un serveur HTTP à l'écoute sur le port 4000

```
const http = require('http');  
const url = require('url');  
  
const server = http.createServer(f);  
server.listen(4000);  
  
function f(request, response) {  
  let page = url.parse(request.url).pathname;  
  if (page=="/index.html")  
    { response.writeHead(200);  
      response.write("hello word !"); }  
  else { response.writeHead(404);  
        response.write(response.statusMessage); }  
  response.end();  
}
```



## 4. Le module standard events

- Le code

```
const f=function(req, rep) {};  
const server = http.createServer(f);
```

- Est équivalent au code

```
const f=function(req, rep) {};  
const server = http.createServer();  
server.addListener('request', f);
```

→ on retrouve la gestion des événements

## 4. Le module standard events

- <https://nodejs.org/api/events.html>
- objet de type `EventEmitter` : peut émettre et recevoir un événement

### Méthodes

#### constructeur

`addListener(event, callback)`

**OU** `on(event, callback)`

`emit(event)` : **déclenche l'événement** `event` **sur l'objet**

`removeListener(event, callback)`

**NB** : un objet de type `http.Server` hérite donc de `EventEmitter`

# Exercice

- Créer un objet de type **EventEmitter**  
Lui faire écouter l'événement **'gameOver'**  
La fonction de *callback* affiche dans la console le message  
**"bye bye"**  
→ tester
- Ajouter l'instruction pour que l'objet créé émette l'événement  
**'gameOver'**  
→ tester

# Exercice

- L'objet créé attend 1 seconde pour émettre l'événement `'gameOver'`  
→ tester

NB : utiliser la fonction `setTimeout(callback, ms)`

Comment ajouter des modules  
pour de nouvelles fonctionnalités ?

## 6. Compléments sur les modules : charger des modules

- npm : outil disponible sur la machine et référentiel central en ligne de modules utilisables

- Commande `npm` pour installer un module externe

Exemple : `npm install colors`

- Instruction `require("colors")`

→ Node installe le module en local dans le répertoire prédéfini `node_modules`. Cela évite d'avoir à préciser le chemin.

NB : <https://www.npmjs.com/package/colors>

- <https://www.npmjs.com>
- écosystème le plus vivace toutes technos confondues, avec près de **250 000 modules** début mars 2016

## 6. Compléments sur les modules : module personnel

- Dans le répertoire principal, créer un sous-répertoire : **moduleEssai**
- Créer dans le sous-répertoire un fichier vide : **util.js**
- Créer dans le sous-répertoire un fichier de nom **package.json**

```
{"main" : "util.js"}
```

- Dans le répertoire créer un fichier contenant l'instruction :  
**const mod=require("./moduleEssai");**  
**console.log(mod);**



## 6. Compléments sur les modules : module personnel

- Ajouter dans `util.js` une fonction qui retourne le plus grand de deux valeurs passées en paramètres

- Utiliser ce module et tester :

```
const mod=require("../moduleEssai");  
console.log("l'objet est "+mod);  
console.log(mod.max(2,3));
```

- Que se passe-t-il ?

## 6. Compléments sur les modules : module personnel

- Exporter les méthodes : l'attribut exports

```
function max(a,b)
    { return a>b?a:b; }
```

```
module.exports.max=max;
```

- Ce qui n'est pas exporté est **local** au module

## 7. L'objet module

- [https://nodejs.org/api/modules.html#modules\\_the\\_module\\_object](https://nodejs.org/api/modules.html#modules_the_module_object)
- Pour chaque module créé par l'utilisateur, Node.js crée un objet de nom `module`
- L'objet `module` est accessible dans le module

# 7. L'objet module

attribut	rôle
id	Correspond au chemin d'accès vers le module
exports	Méthodes et attributs exportés vers les autres modules
parent	Objet module parent (null si main)
filename	Chemin d'accès complet
loaded	true si module complètement chargé
children	Tableau des objets module correspondant aux modules inclus
path	Répertoires node_modules dans lesquels les modules sont recherchés

## 7. L'objet module

- Faire afficher toutes les propriétés de l'attribut module de util.js

```
for (let p in module)
    console.log(p+ « --> "+module[p]);
```