Rappels Javascript

Animé par Mazen Gharbi

L'histoire de Javascript

- Création du langage en **1995** ;
  - Version initiale de Javascript créée en 10 jours seulement !

- **1997** : Javascript gagne la guerre et s'impose comme un standard « cross-browser » sous le nom officiel « EcmaScript » ;

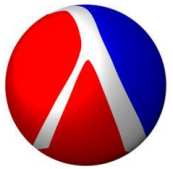
- **2009** : Sortie de NodeJS ;

- **2015** : Finalisation du standard EcmaScript 6 ;

Propriétés du langage

- Dynamiquement typé
- Faiblement typé
- Multi-paradigme (Orienté Objet / Fonctionnel)
  - Brendan Eich s'est inspiré de Self, Scheme, Java et C
- « Cross-browser » et « cross-platform »






Asynchrone et Single-Thread

- "Single-Threaded" = "Thread-Safe" / performant
  - La fonction ne peut être interrompue de l'extérieur
  - Pas limité par un nombre maximum de threads et les allocations mémoire associées.

```
function (user) {
        user.firstName = 'Foo’;

        /* Will never never never be interrupted here...
        * ...except if you shutdown your computer. */

        user.lastName = 'BAR’;
}
```

Asynchrone et Single-Thread

- "Single-Threaded" = "Thread-Safe" / performant
  - La fonction ne peut être interrompue de l'extérieur
  - Pas limité par un nombre maximum de threads et les allocations mémoire associées.

```
function (user) {
        user.firstName = 'Foo’;

        /* Will never never never be interrupted here...
        * ...except if you shutdown your computer. */

        user.lastName = 'BAR’;
}
```

Les variables

```
var userName = 'Foo’;

userName += ' BAR’;

console.log(userName); // Foo BA
```

- Déclarez vos variables en début de bloc ;
- Toujours initialiser les variables !

Valeur par défaut

```
var userName;

console.log(userName) // ???
```

Valeur par défaut

```
var userName;

console.log(userName) // undefined
```

- Quelle différence entre **null** et **undefined** ?

Les objets

```
/* Create user object. */
var user = { firstName: 'Foo' };

/* Add `lastName` attribute. */
user.lastName = 'BAR’;

/* Remove `firstName` attribute. */
delete user.firstName;

/* Change value */
user.lastName = ‘Foo’;
```

Les fonctions

```
var userName = function userName(user) {
     return user.firstName + ' ' + user.lastName;
};

var user = { firstName: 'Foo', lastName: 'BAR’ };

userName(user) // Foo BAR
userName(user, 1, 2, 3, 4) // =>
userName() // =>
```

Foo BAR

TypeError: Cannot read property…

Classes et Objets

- L’ajout dynamique de méthode fonctionne..

```
var user = { firstName: 'Foo', lastName: 'BAR’ };

user.name = function () {
      return this.firstName + ' ' + this.lastName;
};
```

Classes et Objets

- ✓ Mais c'est mieux comme ça !

```
var user = null;
var User = function User(firstName, lastName) {
      this.firstName = firstName;
      this.lastName = lastName;
};
User.prototype.name = function name() {
      return this.firstName + ' ' + this.lastName;
};
user = new User('Foo');
console.log(user.name()) // =>  Foo undefined
```

Closures

Fermeture (informatique)

Pour les articles homonymes, voir Closure et Fermeture.

Dans un langage de programmation, une **fermeture** ou **clôture** (en anglais : ***closure***) est une fonction accompagnée de son environnement lexical. L'environnement lexical d'une fonction est l'ensemble des variables *non locales* qu'elle a capturé, soit par *valeur* (c'est-à-dire par copie des valeurs des variables), soit par *référence* (c'est-à-dire par copie des adresses mémoires des variables)[1]. Une fermeture est donc créée, entre autres, lorsqu'une fonction est définie dans le corps d'une autre fonction et utilise des paramètres ou des variables locales de cette dernière.

Une fermeture peut être passée en argument d'une fonction dans l'environnement où elle a été créée (passée *vers le bas*) ou renvoyée comme valeur de retour (passée *vers le haut*). Dans ce cas, le problème posé alors par la fermeture est qu'elle fait référence à des données qui auraient typiquement été allouées sur la pile d'exécution et libérées à la sortie de l'environnement. Hors optimisations par le compilateur, le problème est généralement résolu par une allocation sur le tas de l'environnement.

- Pour faire simple :
  - Peut être appeler dans n'importe quel contexte ;
  - Se souvient du contexte dans lequel l'appel a été fait.

```
var value = null;

setTimeout(function () {
        value = 'value has been set’;
}, 100 /* 100 ms. */);

console.log(value); // => null

setTimeout(function () {
        console.log(value); // => ‘value has been set’
}, 200);

(function (value) {
        console.log(value); // => undefined
})();
```

Attention aux abus !

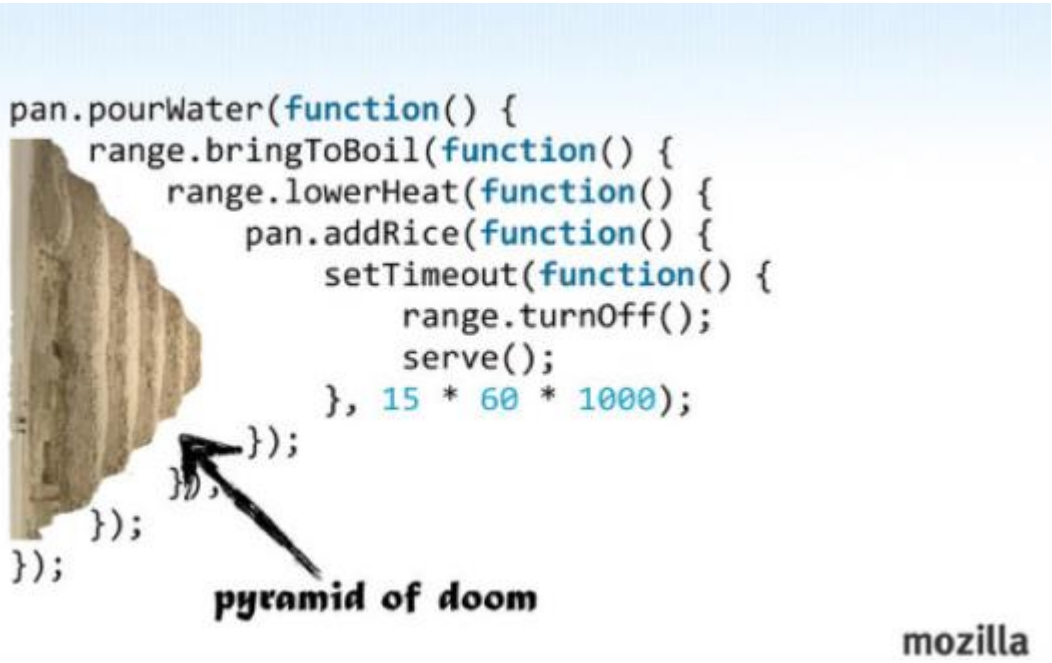
```
var lastInfo = null;

server.loadUser(function (user) {
      user.loadInfos(function (infos) {
            infos[0].save(function (info) {
                  lastInfo = infos[0] = wish;
            });
      });
});
```

```php
function register()
{
    if (!empty($_POST)) {
        $msg = '';
        if ($_POST['user_name']) {
            if ($_POST['user_password_new']) {
                if ($_POST['user_password_new'] === $_POST['user_password_repeat']) {
                    if (strlen($_POST['user_password_new']) > 5) {
                        if (strlen($_POST['user_name']) < 65 && strlen($_POST['user_name']) > 1) {
                            if (preg_match('/^[a-z\d]{2,64}$/i', $_POST['user_name'])) {
                                $user = read_user($_POST['user_name']);
                                if (!isset($user['user_name'])) {
                                    if ($_POST['user_email']) {
                                        if (strlen($_POST['user_email']) < 65) {
                                            if (filter_var($_POST['user_email'], FILTER_VALIDATE_EMAIL)) {
                                                create_user();
                                                $_SESSION['msg'] = 'You are now registered so please login';
                                                header('Location: ' . $_SERVER['PHP_SELF']);
                                                exit();
                                            } else $msg = 'You must provide a valid email address';
                                        } else $msg = 'Email must be less than 64 characters';
                                    } else $msg = 'Email cannot be empty';
                                } else $msg = 'Username already exists';
                            } else $msg = 'Username must be only a-z, A-Z, 0-9';
                        } else $msg = 'Username must be between 2 and 64 characters';
                    } else $msg = 'Password must be at least 6 characters';
                } else $msg = 'Passwords do not match';
            } else $msg = 'Empty Password';
        } else $msg = 'Empty Username';
        $_SESSION['msg'] = $msg;
    }
    return register_form();
}
```

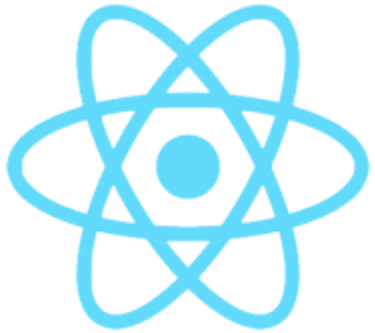
WATERFALL SUICIDE

PYRAMID OF DOOM



Quelques limites du langage

- Très permissif.. Trop..
- Pas fait pour les grosses applications



React

- Pas d’introspection

```
var User = function User() {
        var user = {
                isDirty: false, firstName: 'Foo', lastName: 'BAR’
        };
        return {
                firstName: function firstName() {
                        return user.firstName;
                },
                setFirstName: function setFirstName(firstName) {
                        user.firstName = firstName;
                        user.isDirty = true;
                }
        }
};
var user = User();
user.setFirstName('John’);
console.log(user.firstName()); // 'John' // How to test the value of 'isDirty' ?
```

Quelques bonnes pratiques

- Préférez toujours "if (value === true)" à "if (value)".
- camelCase
- Indentation à 4 espaces plutôt que 2 pour décourager les cascades de callback.
- Nommez toutes vos fonctions.
- Déclarez et initialisez vos variables.
- Mettez toujours les « ; » en fin de ligne, même si optionnel.
- Déclarez vos variables en début de bloc ;
- Initialisez vos variables ;
- Utilisez un maximum les nouvelles fonctionnalites ES6 !

EcmaScript 6 / 2015

- Les classes !! Avec héritage
- Modules
- Arrow Functions
- Template Strings
- Spread & Rest
- Déstructuration (objet et array)

Arrow functions

```
let maFonction = (a) => {
     let result = a + 1;
     return result;
}

console.log(maFonction(2));
```

Destruction d’array

```
let userList = [
      {firstName: 'Foo’},
      {firstName: 'Mads’}
];

let [user1, user2] = userList;

console.log(user1); // { firstName: 'Foo’ }
console.log(user2); // { firstName: 'Mads' }
```

Destructuration d’objet

```
let user = {
     firstName: 'Foo’,
     lastName: 'BAR’,
     email: 'foo.bar@wishtack.com’
};

let {lastName, firstName} = user;
console.log(firstName); // Foo
console.log(lastName); // BAR
```

Les classes

```
class User {
        constructor(firstName, lastName) {
            this._firstName = firstName;
            this._lastName = lastName;
        }

        /* Getter. */
        firstName() {
            return this._firstName;
        }
        /* Property. */
        get lastName() {
            return this._lastName;
        }
        set lastName(lastName) {
            this._lastName = lastName;
        }
}
```

```
let user = new User('Foo’);

console.log(user.firstName());
// => ‘Foo’
console.log(user.lastName);
// => undefined

user.lastName = 'BAR';
console.log(user.lastName);
// => ‘BAR’

console.log(user._lastName);
// => ‘BAR’
```

Scope du mot-clé ‘let’

```
let x = 1;

if (x === 1) {
     let x = 2;
     console.log(x);  // => 2
}

console.log(x); // =>  1
```

Quizz

```
var foo = 1;

function bar() {
  if(!foo) {
    var foo = 10;
  }

  console.log(foo);
}

bar();
```

> Que s’affiche-t-il dans la console?

```
var a = 1;

function b(a) {
      console.log(a);
      a = 10;
}

b();
console.log(a);
```

> Que s’affiche-t-il dans la console?

```
var a;
var r2 = a || {name: 'toto’};

console.log(r2);
```

> Que s’affiche-t-il dans la console?

```
var c = 10;
var r2 = c && function() { var b = 3 + 8; console.log(b); return b;
} && 'tt';

console.log(r2);
```

> Que s’affiche-t-il dans la console?

```
var myObject = {};
console.log(myObject.a);

var foo;
console.log(foo);

console.log(bar);
```

> Que s’affiche-t-il dans la console?

```
var hi = function(name) {
  return 'Hi ' + name;
}

var greeting1 = function(name) {
  return hi(name);
}

var greeting2 = hi;

console.log(greeting1('Abdel'));
console.log(greeting2('Mazen'));
```

> Que s’affiche-t-il dans la console?

```
var obj = {
  data: 'ma chaine'
};

function myFunc() {
  console.log(this);
}

obj.myFunction = myFunc;

obj.myFunction();
```

> A qui s'appliquera le '*this*' affiché?

```
var fact = function factorial(n) {
  console.log(n);
  return n === 0 ? 1 : (n * this.factorial(n - 1));
}

var r = fact(5);
console.log(r);
```

> Ce code fonctionne-t-il ?

```
for(var i = 1; i < 4; i++) {
  setTimeout(function() {
    console.log(i);
  }, 1000 * i)
}
```

> Que s’affiche-t-il dans la console?

```
(function() {
  var createWorker = function() {
    var count = 0;
    var task1 = function() {

    };

    var task2 = function() {

    };

    return {
      job1: task1,
      job2: task2
    };
  }

  var worker = createWorker();
  worker.job2();
})();
```

> A quoi sert la syntaxe (function () { ... }()); syntax?

Questions