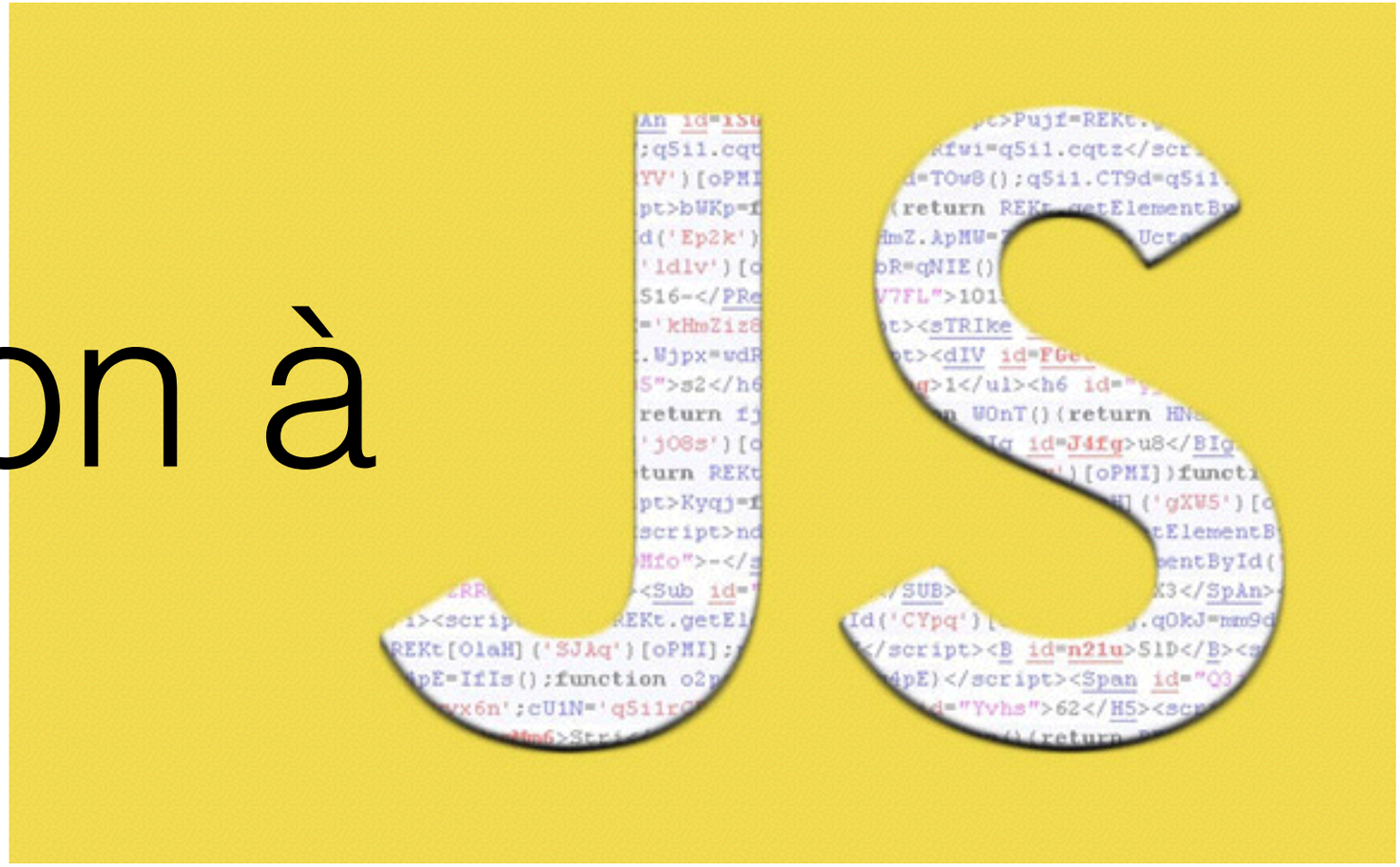
Initiation à



yann.secq@univ-lille1.fr

Rémy DELERUE & Guillaume DUFRENE

Université de Lille 1 SCIENCES ET TECHNOLOGIES

Département Informatique
IUT « A » - Université Lille 1

Promotion 2015-2016
Cours 2

Bases de JavaScript

- La notion d'objet en JavaScript
  - singleton, constructeurs et « classes »
  - *duck typing*
- La programmation fonctionnelle en JavaScript
  - notion de fonction et de fermeture (*closure*)
  - les fonctions d'ordres supérieures
  - une fonction classique: `map`

Notion d’objet en JS

- Un objet JS est constitué d’un ensemble de **propriétés**
- Une propriété est un couple *(clé, valeur)*
- Une clé est une chaîne de caractères
- Une valeur est … n’importe quel type de valeur !
- Pas de distinction structurelle entre un « attribut » et une « méthode » en JS !
- Si valeur = fonction alors c’est une méthode, sinon c’est un attribut

Créer un objet JS

- Plusieurs approches sont possibles:
  - création directe (*object literal*)
  - création via un constructeur (*constructor*)
  - création via `Object.create`

Via un *object literal*

```
var p = {
  name: 'John',
  describe:
    function () {
      return 'I m '+this.name;
    }
};
```

```
> p.name
'John'
> p.name = 'Totoro';
> p.age  = 42;
> p.describe();
'I m Totoro'
> p.name = 'Akira';
> p.describe()
'I m Akira'
> 'age' in p
true
> 'foo' in p
false
> delete p.age
true
> 'age' in p
false
```

Via un constructeur

```
// Attributs
function Point(x, y) {
  this.x = x;
  this.y = y;
}
// Méthodes
Point.prototype.dist =
  function () {
    return Math.sqrt(
      this.x*this.x +
      this.y*this.y);
  };
```

```
> var p = new Point(3, 5);
> p.x
3
> p.dist()
5.830951894845301
> p instanceof Point
true
```

Via `Object.create`

```
// Shape - superclass
function Shape() {
  this.x = 0;
  this.y = 0;
}

// superclass method
Shape.prototype.move = function(x, y) {
  this.x += x;
  this.y += y;
  console.info('Shape moved.');
};

// Rectangle - subclass
function Rectangle() {
  Shape.call(this); // call super constructor.
}

// subclass extends superclass
Rectangle.prototype = Object.create(Shape.prototype);
Rectangle.prototype.constructor = Rectangle;

var rect = new Rectangle();

console.log("Is instance of Rectangle? " + (rect instanceof Rectangle)); // true
console.log("Is instance of Shape? " + (rect instanceof Shape)); // true
rect.move(1, 1); // Outputs, 'Shape moved.'
```

Quand les utiliser ?

- Selon les besoins:
  - *object literal*: ne pollue pas l'espace de nommage, utile pour les singletons
  - constructeur: pratique pour initialiser les objets à la création et lorsque plusieurs instances sont requises
  - `Object.create`: lorsque l'on n'a pas d'initialisation à la création de l'objet, pour la définition de liens d'héritage

duck typing

- En Java le typage est statique et fige la structure des objets à l'exécution

- En JS, le typage est vérifié à l'exécution, la structure peut donc évoluer dynamiquement

- Possibilité de tester la structure d'un objet pour savoir si il possède une propriété donnée

duck typing

- « *Si je vois un animal qui vole comme un canard, cancane comme un canard, et nage comme un canard, alors j'appelle cet oiseau un canard* » (James Whitcomb Riley)

```
function calcule(a, b) { return a+b; }
var a = calcule (1, 2);
var b = calcule ('foo', 'bar');
console.info(a); // => 3
console.info(b); // => foobar
```

Fonctions et *closure*

- Les fonctions sont un type comme les autres
- Lors de la création d'une fonction, les variables utilisées sont capturées (notion de fermeture)
- Une fonction peut prendre en paramètre d'autres fonctions et même retourner une fonction !

Exemple de fermeture

```
function init() {
    // variable locale
    var name = "Mozilla";
    // fonction interne, fermeture
    function displayName() {
        // accès à la variable locale
        alert(name);
    }
    displayName();
}
init();
```

Fonction d'ordre supérieur

- Les fonctions étant manipulables directement, il est possible de les passer en paramètre ou de les retourner comme résultat

- Fonction d'ordre supérieur = fonction prenant en paramètre et/ou retournant une fonction

- Exemple: fonction de tri selon divers comparateurs

- Opération classique: `map`

Fonction d’ordre supérieur

```
function creerFonction() {
  var name = "Mozilla";
  function displayName() {
    alert(name);
  }
  return displayName;
}
// myFunc référence la fonction displayName
var myFunc = creerFonction();
myFunc();
```

Définition d’une fonction qui retourne une fonction

Recréons `map`

```
var map = function (liste, f) {
  var res = [];
  // appliquer f sur élément de la liste
  for (var i=0; i<liste.length; i++) {
    res.push(f(liste[i]));
  }
  return res;
};
// la fonction à appliquer
var carre = function(nb) { return nb*nb; }
// sur chacun des éléments de la liste
var a = map([1, 2, 3], carre);
console.info(a); // => [1,4,9]
```

Array.map(array,function)

```
var numbers = [1, 4, 9];
var roots = numbers.map(Math.sqrt); // [1, 2, 3]
// ou plus simplement: [1, 4, 0].map(Math.sqrt)
```

```
var numbers = [1, 4, 9];
var doubles = numbers.map(function(nb) {
  return nb * 2;
}); // 2, 8, 18]

var map = Array.prototype.map;
var a = map.call('Hello World', function(x)
{ return x.charCodeAt(0); });
// [72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100]
```