Initiation à

yann.secq@univ-lille1.fr

JeanRémy DELERUE & Guillaume DUFRENE

Université de Lille 1 SCIENCES ET TECHNOLOGIES

Département Informatique
IUT « A » - Université Lille 1

Promotion 2015-2016
Cours 1

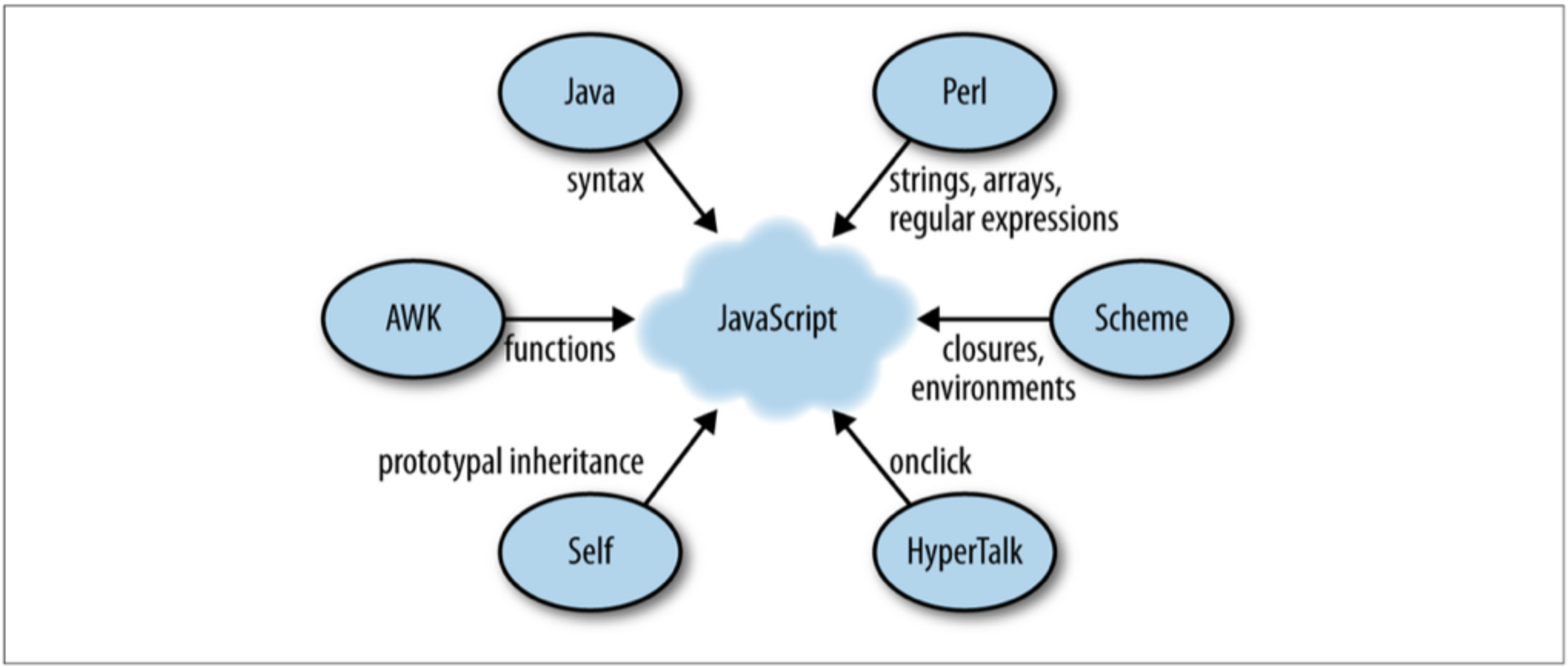
Initiation à **JavaScript**

- **Objectifs**
  - compréhension des principes du langage
  - mise en pratique pour une utilisation côté client
    - usage d'une librairie JS répandue (*jQuery*)
- **Organisation**
  - 5 semaines : 1h C + 3h TP
  - 1 semaine : projet agile avec prog. distribué + androïd !

Caractéristiques de JS

- **dynamique**: langage à prototype, pas de définition préalable de classe nécessaire !
- **dynamiquement typé**: les variables et propriétés (attributs) peuvent contenir n'importe quelles valeurs
- **fonctionnel et orienté objets**: fonction d'ordre supérieur, *closures* + objets, héritage, variables mutables
- **déploiement du code source**: JS déployé via un code source compilé par la VM l'exécutant
- **élément essentiel du web**: l'*esperanto* des clients webs …
- **échoue silencieusement** :(

JavaScript: influences



```
// Déclaration d'une variable
var x;
// Affectation d'une valeur à la variable `x`
x = 3 + y;
// Appel de la fonction `foo` avec les paramètre `x` et `y`
foo(x, y);
// Appel de la méthode `bar` de l'objet `obj`
obj.bar(3);
```

```
// Expression conditionnelle
if (x === 0) { // `x` vaut zéro ?
  x = 123;
}
// Définition de la fonction `baz` avec paramètres `a` et `b`
function baz(a, b) {
  return a + b;
}
```

Bases de JavaScript

- Déclaration des variables avant utilisation: `var x;`
- Nom de variable sensibles à la casse (!)
- Opérateur d'affectation: = (ainsi que += -= …)
- Commentaires presque similaire à Java: // et /* */
- Terminer les instructions et expressions par un ;

Mots réservés du langage

`arguments`	`break`	`case`	`catch`		
`class`	`const`	`continue`	`debugger`		
`default`	`delete`	`do`	`else`		
`enum`	`export`	`extends`	`false`		
`finally`	`for`	`function`	`if`		`Infinity`
`implements`	`import`	`in`	`instanceof`	&	`NaN`
`interface`	`let`	`new`	`null`		`undefined`
`package`	`private`	`protected`	`public`		
`return`	`static`	`super`	`switch`		
`this`	`throw`	`true`	`try`		
`typeof`	`var`	`void`	`while`		

Valeurs en JS

- Distinction entre valeurs primitives et objets (cf. Java):
  - **types primitifs**: booléens (`true` et `false`), nombres (réels uniquement !), chaînes (avec `''` ou `""`), ainsi que les deux non valeurs *`null`* (pas d'objet) et *`undefined`* (pas de valeur)
  - **types objets**: les objets, les tableaux, les expressions régulières
- Les types primitifs sont comparés par valeur et ne sont pas mutables
- Les types objets sont comparés par référence et sont mutables
- Les valeurs ont des propriétés (attributs) se présentant sous la forme de (clé,valeur)

Primitifs vs. Objets

Comparaison logique
(par valeur)

```
> 3 === 3
true
> 'abc' === 'abc'
true
```

Immutabilité

```
> var str = 'abc';
> str.length = 1;
> str.length
3
> str.foo = 3;
> str.foo
undefined
```

Comparaison physique
(par référence)

```
> {} === {}
false
> var obj1 = {};
> var obj2 = obj1;
> obj1 === obj2
true
```

Mutabilité

```
> var obj = {};
> obj.foo = 123;
123
```

Booléens

- Deux valeurs: `true` et `false`
- Opérateurs logiques (presque) classiques: `&&`, `||`, `!`
- Opérateurs d'égalité (!): **`===`**, **`!==`**, `==`, `!=`
- Opérateurs de comparaison: `<`, `<=`, `>`, `>=`
- Valeurs fausses: `undefined`, `null`, `false`, `0`, `NaN`, `''`
- Toutes les autres valeurs sont considérées `true` !

Booléens (suite)

- Court-circuit pour `&&` et `||`
  - `op1 && op2`: `op1` est faux, retourne `op1` sinon `op2`
  - `op1 || op2`: `op1` est vrai, retourne `op1` sinon `op2`

```
> NaN && 'abc'
NaN
> 123 && 'abc'
'abc'
```

```
> 'abc' || 123
'abc'
> '' || 123
123
```

Nombres

- Tous des réels !! : `1 === 1.0`
- Deux valeurs spéciales: `NaN` et `Infinity`
- Ex: `Number('xyz')` => `NaN` / `3.1415926/0` => `Infinity`
- Opérateurs: `+`, `-`, `*`, `/`, `%`, `++`, `—`, `-value`

Chaînes de caractères

- Création avec les délimiteurs `'` ou `"`
- Utilisation du backslash `\` pour échapper une séquence
- Accès direct aux caractères comme dans un tableau:
  `"Hello"[1]` => `"e"`
- Propriétés `length` pour la taille: `"Hello".length` => `5`
- Opérateur de concaténation: `+`
- Quelques méthodes: `slice`, `indexOf`, `trim` ...

Structures de contrôle

- Comme en Java pour les alternatives: `if () {} else {}`
- Idem pour: `switch () { case … : … break; ….}`
- Syntaxe similaire aussi pour les boucles à compteur (`for`) et évènements (`while` / `do while`)
- Quitter une boucle `break`, passer à l'itération suivante `continue`

Fonctions

- Déclaration de fonction vs. expression fonctionnelle
- Déclaration: `function add(a, b) { return a+b; }`
- Expression: `var add = function(a,b) {return a+b;};`
- Arité des fonctions: JS ne nécessite pas le nombre exact de paramètres déclarés par une fonction !
- Trop d'arguments sont rendus accessibles par la variable spéciale `arguments` et des arguments manquants sont remplacés par `undefined` ...

Variables

- Les variables doivent être déclarées avant d'être utilisées
- Les variables n'existent pas seulement au sein du bloc où elles sont déclarées, mais au sein de la fonction dans laquelle elles sont déclarées

- Les variables sont *hoisted* ie. automatiquement déclarées en début de fonction …

```
function foo() {
  var x = -512;
  if (x < 0) { // (1)
    var tmp = -x;
  }
  console.log(tmp); // 512
}
```

```
function foo() {
  console.log(tmp);
  if (false) {
    var tmp = 3;
  }
}
```

```
function foo() {
  var tmp; // hoisted declaration
  console.log(tmp);
  if (false) {
    tmp = 3;
  }
}
```

Closures (fermetures)

- Lorsqu'une fonction est définie, elle capture les variables l'environnant au moment de sa création (*closure*)

- Permet de définir des fonctions comme valeurs

- Permet donc de passer des fonctions en paramètre, c'est ce que l'on appelle des **fonctions d'ordre supérieur** (ie. fonctions paramètres ou retournant des fonctions)

```
function createIncrementor(start) {
  return function () {
    start++;
    return start;
  }
}
```

```
> var inc = createIncrementor(5);
> inc()
6
> inc()
7
> inc()
8
```

Objets

- Pas de notion de classes en *JavaScript* !
- Singleton et constructeurs (ie. « classes »)
  - Singleton: un objet comme un ensemble de clés/valeurs !
  - Constructeurs: définition de l’état puis des méthodes d’un prototype (puis copie de ce prototype)

Constructeurs

- Fonction commençant par une majuscule et définissant les données (état) de l'objet

- Définition de la propriété prototype de cette fonction pour ajouter des méthodes au prototype

```
// Set up instance data

function Point(x, y) {

  this.x = x;

  this.y = y;

}

// Methods

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
```

Singletons

```
'use strict';
var person = {
  name: 'Alan',
  describe: function () {
    return 'Je suis '+this.name;
  }
};
```

```
> person.name // get
'Alan'
> person.name = 'Kurt'; // set
> person.newProperty = 'abc';
> person.describe() //appel de méthode
'Je suis Kurt'
> person.name = 'Yann';
> person.describe()
'Je suis Yann'
```

```
> 'newProperty' in person
true
> 'foo' in person
false
> person.newProperty !== undefined
true
> person.foo !== undefined
false
> delete person.newProperty
true
> 'newProperty' in person
false
```

Tableaux

- Notion classique d'ensemble de données accessibles via un indice (commençant à 0)
- La propriété `length` permet de déterminer le nombre d'éléments que contient le tableau
- L'opérateur `in` teste la présence d'un élément à un index donné

```
> var t = [ 'a', 'b', 'c' ];
> t[0]

'a'

> t[0] = 'x';

> t
[ 'x', 'b', 'c' ]

> t.length

3
```

```
> t.length = 1;

> t

['x']

> var t = [ 'a', 'b', 'c' ];
> 1 in t // élément à l'indice 1?

true
> 5 in t // élément à l'indice 5?

false
```

```
> var arr = [];

> arr.foo = 123;

> arr.foo
123

[ 'a', 'b', 'c' ].forEach(
function (elem, index) {

            console.log(index +

'. ' + elem);

        });
```