programmation web en javascript

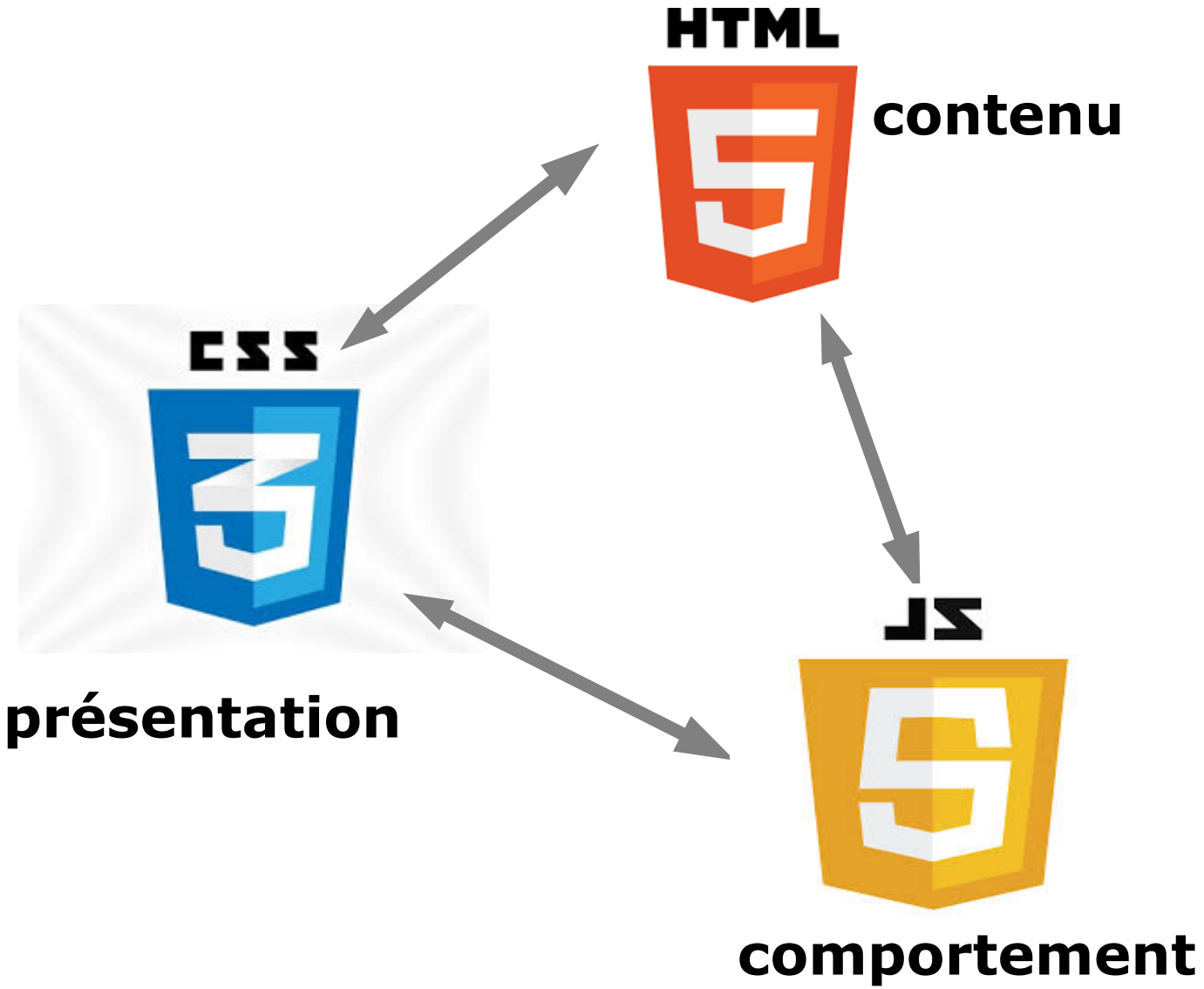
1/6

contenus et organisation

- programmation en javascript
- le DOM : javascript dans le navigateur
- jQuery : une librairie JS

- Evaluation :
  - un projet
  - des tps/tds à rendre

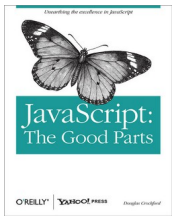
les standards du web : html5



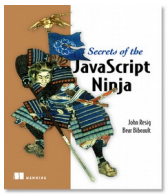
quelques ressources

- mozilla developper network

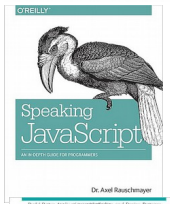
- JavaScript : the good parts
  - O'Reilly, *D. Crockford*



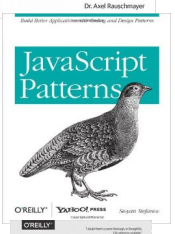
- Secrets of the Javascript Ninja
  - Manning, *J. Resig, B. Bibeault*



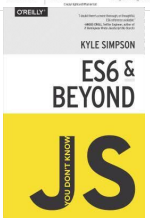
- Speaking JavaScript
  - O'Reilly, *A. Rauschmayer*



- JavaScript Patterns
  - O'Reilly, *S. Stefanov*



- ES6 & Beyond
  - O'Reilly, *Kyle Simpson*



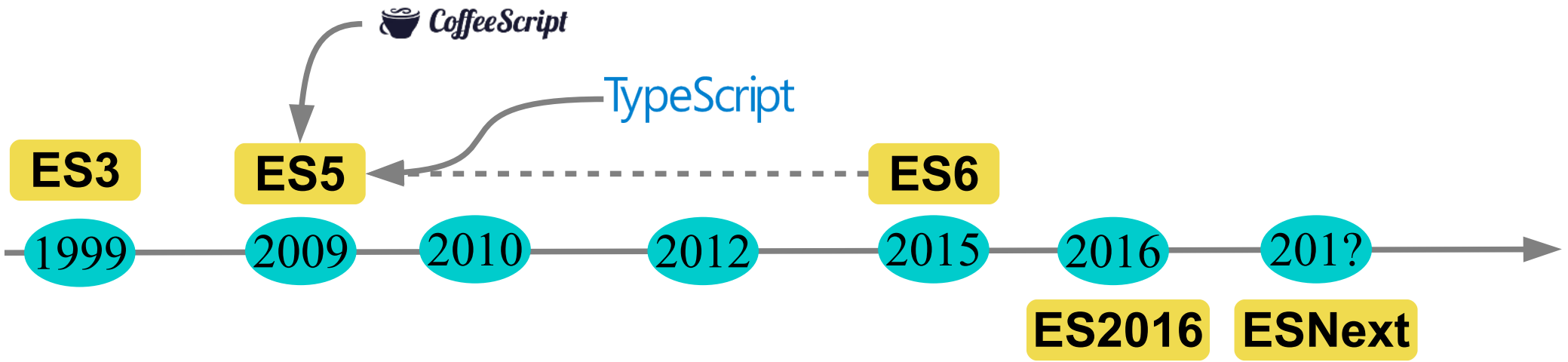
javascript : quelques caractéristiques

- rien à voir avec java

- variables non typées, valeurs typées,
- des fonctions, des fonctions anonymes, des callbacks, des closures
- des objets, des prototypes, (mais pas de classes)
- utilisé de manière événementielle dans un navigateur web : les actions sont exécutées en réponse à des événements

les versions de javascript

- 2 versions principales cohabitent : ES5, ES6
- ES6 **ajoute** des constructions à ES5 mais ne les remplace pas
- le passage de ES5 à ES6 se fait **progressivement** au fil des versions des navigateurs
- tout n'est pas encore implanté dans les navigateurs récents : nécessité de *transpiler*



résumé de syntaxe

```
// commentaire

var x ;              // déclaration d'une variable
var y=5 ;            // déclaration / initialisation de variable
let z=8 ;            // déclaration / initialisation de variable

x = y+8 ;            // affectation

foo(x,y) ;     //appel de la fonction 'foo' avec les args x,y
obj.bar(y) ;   //appel de la méthode 'bar' de l'objet 'obj'

// conditionnelle
if (x === 0) {       // x egal à 0 ?
  x = 42 ;
}
// itération                    // autre itération
while (x > 0) {                 for (x=0 ; x<100 ; x++) {
  x = x-1 ;                           console.log( x ) ;
}                               }

// définition d'une fonction
function baz( a, b) {    // 2 paramètres
   return a * b ;
}
```

mode strict

- javascript possède 2 modes de fonctionnement
  - le mode normal ou *sloppy*
  - le mode *strict* qui génère plus de *warnings* et d'erreurs
- le mode strict peut être activé :
  - pour un fichier complet avec le tag `'use strict'`
  - par fonction

```
function fonctionEnModeStrict() {
    'use strict'
    …
}
```

- le mode strict est souvent plus rapide

Variables et affectation

- en mode strict, les variables **doivent être déclarées**
- elles peuvent être **initialisées** à la déclaration
- elle sont typées **dynamiquement :** pas de typage à la déclaration, typage à l'affectation
- identificateurs : (lettre | $ | _)(lettre | digit | $ | _)*
  - i, $v, _x, v42, $1337, _73x, _tmp, $elem

déclaration et portée des variables

- 2 déclarations de variables
  - `var` : la portée de la variable (scope) est **toute la fonction** où elle est déclarée **ES5**
  - `let` : la portée de la variable (scope) est limitée au **block** où elle est déclarée **ES6**

```
function foo() {
    var v = 42;

    if ( v < 50 ) {
        let w = 50-v;
        var z = w+v;
        console.log( v, w);
    }
    console.log(v, w, z);
}
```

- une variable déclarée en dehors de tout block ou fonction est **globale**
- **éviter ces déclarations globales**

```
Uncaught ReferenceError: w is not defined
```

les valeurs

- les valeurs primitives :
  - **booléens** : `true`, `false`, **nombres** : `42, 3.141592`
  - **strings** : `'abc', "abcd"`
  - **`undefined, null`**
- les objets
  - objets classiques : `{ prenom : 'joe', nom :'bar' }`
  - tableaux : `[ 1, 2, 3 ]`,
  - **fonctions** et **regexp**
- les valeurs ont un type :

```
> typeof 1
"number"
> typeof {}
"object"
> typeof undefined
"undefined"
```

les valeurs primitives

- les valeurs primitives sont des **objets non modifiables**
  - possèdent des propriétés et des méthodes, mais ne peuvent pas être changés
- comparées par valeur :

```
> 3 === 3
true
> let s = 'abc'
> s === 'abc'
true
> s.length
3
> s.length = 5
5
> s.length
3
```

```
//attention : transtypage

> 42 === "42"
false

> 42 == "42"
true

// éviter d'utiliser ==
```

les strings

- "string" ou 'string'

```
> let prenom = "joe" ;
> let nom = 'bar' ;
```

- propriétés

```
> prenom.length  ;
3
```

- méthodes

```
> nom.toUpperCase()  ;
"BAR"
```

- concaténation

```
> prenom+ " " + nom ;
"joe bar"
```

les "string templates"

- expression de chaine de caractères permettant d'inclure des **variables** et des **expressions** évaluées au moment de ***l'affectation*** **ES6**

```
let nom  = 'neymar';
let prenom = 'jean';
let age = 72;

let hello = `Hi, ${prenom} !,
     how are you today, ${prenom + ' '+nom},
     happy birthday ${age+1}` ;

console.log(hello);
```

```
Hi, jean !,
     how are you today, jean neymar,
     happy birthday 73
```

numbers, booleans

- les nombres sont tous des flottants
- opérateurs : `+, -, /, *, ++, —, %`
- 2 valeurs spéciales : `Infinity , NaN`

- `true, false`
- valeurs fausses :
  - `undefined, null, false, 0, NaN,''`
- opérateurs produisant des booleens :
  - <, <=, >, >=, ===,!==
  - !, &&, || : *(court-circuit)*

```
> 1 === 1.0
true
> 3 / 0
Infinity
> Number( 'abc' ) ;
NaN
> let i = 6 ;
> 5 + --i
10
```

```
// foo() n'est jamais
// appelée :

> false && foo()
false
> true || foo()
true
```

les valeurs objets

- objets classiques : `{ }, { x:12, y:24 }`
- tableaux : `[1,2,3,4,5]`
- regexp et fonctions

- modifiables, comparés et affectés par référence :

```
> let o = { } ;
> o === {}
false
> o === o
true
> let o2 = o ;
> o === o2
true
```

```
> let o = { } ;
> o
Object {}

> o.x = 12

> o
Object { x:12 }
```

les fonctions en javascript

- **une fonction js est un objet**, donc elle peut être :
  - **créée** via un littéral
  - **affectée** à des variables, tableaux, propriétés
  - passée en **paramètre** à des fonctions
  - retournée comme **résultat** d'une fonction
  - possède des **propriétés** et des **méthodes**

- **et en plus, elle peut être *invoquée* en plaçant des parenthèses: ()**

déclaration de fonctions

- le mot clé *function*
- un nom
- une liste de paramètres entre ( )
- une série d'instruction entre { }

```
function [name] ( [p1] [,p2 …] ) { … ; return … }
```

```
//déclaration de fonction
function add ( a,b ) {
      return a+b ;
}


>add.name
"add"
>add( 30, 12 )
42
```

```
>add
ƒ add( a, b ) {
     return a+b;

}

>typeof add
"function"

>typeof add(3,4)
"Number"
```

expressions de fonctions, fonctions anonymes

- il est possible d'affecter ou passer en paramètre des **expressions de fonction** sans avoir à déclarer la fonction
- en **ES5** : fonctions **anonymes :** elles n'ont pas de nom
- en **ES6** : le nom est **inféré**

```
let somme= function( a,b ) {
    return a+b ;
}

console.log ( somme.name );
console.log( somme(1300, 37 )) ;
```

```
somme
1337
```

expressions de fonctions =>

- **ES6** introduit une nouvelle notation pour les expressions de fonctions : les fonctions =>
  - `( p1, p2 ) => { …. ; return v }`

```
let somme   = ( a,b ) => { return a+b ; }
let adams   = () => {return 42; }
let incrBy1 = (x) => x+1;
console.log( adams.name )
```

```
adams
```

invocation immédiate

- une fonction ou une expression de fonction peut être invoquée **immédiatement** à sa création

```
let x = (function return42() {return 42;}) ();

let y = (function () {return 73;}) ();

let z = ((x) =>  1300+x) (37);

x
// 42
y
// 73
z
// 1337
```

arguments des fonctions

ES5

- lors d'un appel, le nombre d'arguments peut être différent du nombre de paramètres
  - arguments manquants : `undefined`
  - arguments supplémentaires : dans le pseudo-tableau `arguments`

```
> function f( x , y ) {
      console.log( x, y) ;
      return arguments ;
}

> f(1,2)
// 1 2
[1,2]
```

```
> f(1,2,3)
// 1 2
[1,2,3]

> f(1)
// 1 undefined
[1]
```

arguments des fonctions

- ES6 permet de définir des valeurs par défaut
- notation explicite pour les paramètres optionnels

ES6

```
let buzz = (x, y=42, ...args) =>
{
    console.log('x : ', x);
    console.log('y : ', y);
    console.log('args : ', args);
}
```

```
>buzz()
// undefined 42 []
>buzz(12)
// 12 42 []
>buzz(12,13)
// 12, 13
>buzz(12,13,14,15)
// 12, 13, [14,15]
```

- Si l'argument 2 est absent, y reçoit la valeur 42
- tous les arguments supplémentaires sont dans le tableau args