DAR - Cours 6
Javascript

Romain Demangeon

APR, LIP6, UPMC

23/10/2017

Organisation

- Lundi 30/10: Vacances.
- Lundi 06/11: Cours 07, Amphi Durand
  Vincent Balat, P7/*BeSport*: Présentation d'Ocsigen.
- Lundi 13/11: Révisions.
- Lundi 20/11: Examen.
- Mercredi 22/11: Soutenances alternants.
- Lundi 27/11: Soutenances non-alternants.

Projet DAR 2017

Ce qui doit être fait:

- Prototype d'application.
- Prototype de rapport.

A faire cette semaine:

- Enrichir le projet (fonctionnalités).
- Enrichir le rapport.
- Décider de l'axe de la soutenance.

JavaScript (Rappel)

Langage de programmation de scripts, orienté-objet à prototypes avec typage dynamique et fonctions comme citoyens de première classe.

- prototypes: objets sans classes, modelés depuis un autre objet.
- typage dynamique faible: l'interpréteur gère le typage, JS est très permissif.
- fonctions de première classe: les fonctions peuvent être manipulées et créées à l'exécution.

- Langage ubiquitaire.
  - top 5 des langages programmés.
  - haut dans les langages exécutés (métrique ?).
- Utilisation principale: navigateurs.
  - code source téléchargé depuis le serveur.
- 3 couches:
  1. le langage JavaScript (standard ECMAScript)
  2. les APIs Javascript standardisées (DOM, ...)
  3. bibliothèques JavaScript (jQuery,...)

Javascript: Philosophie

- Langage initialement destiné au web designers.
- Langage initialement dessiné pour la manipulation de pages web.
- Langage dirigé par l'implémentation.

le langage que le web mérite . . .

- Facile d'acces.
- Impossible à maitriser.

Objectif du cours

"C'est pas facile."

Syntaxe: Point-virgule

Principe

Syntaxe impérative classique (C, Pascal).

Cas du ; (point-virgule): fin d'instruction (et non séparateur) optionnelle.

- ambiguïtés

```
1 function Personne(nom, age){this.nom = nom, this.age = age,
2                              this.affiche = function (){
3                                alert(nom + " a " + age + " ans.")}}
4 var annie = new Personne("Annie", 89)
5 var copie = annie
6 [annie].forEach((x) => {x.affiche()})
```

- l'utiliser autant que possible.
  - c'est un débat.

Syntaxe: RegExp

- Les expressions rationnelles sont des citoyens de première classe.
- Elles peuvent être argument et résultat des fonctions.
- Elles peuvent être stockées dans des variables.

```
function prefixe(pre){var reg = new RegExp(pre+"*",""); return reg}
prefixe("sau").test("saucisse") == true // vrai
prefixe("sau").test("chocolat") == false // vrai
```

- On peut les manipuler de manière anonyme avec /:

```
/sau*/.test("saucisse") == true // vrai
/sau*/.test("chocolat") == false // vrai
```

- une regexp comporte (entre autres) les méthodes:
  - `test`, `match` (occurences), `search` (indice), `split` (séparation), `replace` (search and replace)

```
var chaine = "stephanie + alex = APTSD";
function change(s){
    var separe = s.split(/[+=]/);
    var res = "";
    separe.forEach((x) => {if(/[A..Z]/.test(x)){res += ""}
                           else{res += x + "<3"}});
    return res
}
change(chaine) // "stephanie <3 alex <3"
```

UPMC SORBONNE UNIVERSITÉS

Syntaxe: Fonctions

- les fonctions sont des citoyens de première classe.
  - comme dans les langages fonctionnels (ML, Haskell, Swift, ...)
  - JS est un langage fonctionnel.
- Elles peuvent être argument et résultat des fonctions.
- Elles peuvent être stockées dans des variables.
  - Elles sont anonymes (lambdas).

```
1 function f1(x){
2     return x + 1}

4 var f2 = function(x) {return x + 1}

6 var f3 = (x) => {return x + 1}

8 var f4 = x => x + 1;
```

Syntaxe: Fonctions

- les fonctions sont des citoyens de première classe.
  - comme dans les langages fonctionnels (ML, Haskell, Swift, ...)
  - JS est un langage fonctionnel.
- Elles peuvent être argument et résultat des fonctions.
- Elles peuvent être stockées dans des variables.
  - Elles sont anonymes (lambdas).

```
1 function f1(x){
2     return x + 1}

4 var f2 = function(x) {return x + 1}

6 var f3 = (x) => {return x + 1}

8 var f4 = x => x + 1;
```

- Différence entre les deux (*hoisting*):

```
1 alert(f1(3)) // affiche 4

3 function f1(x){
4     return x + 1}
```

```
1 alert(f2(3)) //  TypeError

3 var f2 = function(x) {return x + 1}
```

UPMC SORBONNE UNIVERSITÉS

Syntaxe: Fonctions (II)

- Arguments récupérés depuis `arguments`.

```
1 function aplatir(){
2     var res = "";
3     for(let arg of arguments){
4         res += arg};
5     return res}

7 aplatir("1","2","3","saucisse") // "123saucisse"
```

  - `for in` et `for of`

```
1 for(i in ["a","b","c"])(console.log(i))
2 for(i of ["a","b","c"])(console.log(i))
```

- Arguments récupérés dans un tableau:

```
1 function aplatir(...mots){
2     return mots.reduce((x,y) => {return x + y},"")}

4 aplatir("1","2","3","saucisse") // "123saucisse"
```

- Note: les fonctions sont aussi des objets.

```
1 aplatir.length // 0 (nombre de variables)
```

Sémantique: Portée

Principe

La portée d'une variable est la fonction qui contient sa déclaration.

- Différent de C (déclaration en blocs).
- *Hoisting*: la déclaration peut se faire n'importe où dans la fonction

```
1 var a = "saucisse";

3 function f(){res = a + "-chocolat"; return res};
4 f() // "saucisse-chocolat"

6 function g(){res = a + "-chocolat"; return res; var a};
7 g() // "undefined-chocolat"
```

- JS "remonte" les déclarations en début de code.
- Fonctions: JS "remonte" aussi les définitions

```
1 console.log(x) // ReferenceError

3 console.log(x) // undefined
4 var x = 1

6 console.log(x()) // 1
7 function x(){return 1}
```

Portée de bloc: iffy

iffy *Immediately-Invoked Function Expression*

- Cadre: Hoisting de fonction.
- Objectif: fixer la valeur d'une variable dans une fonction.

```
1 var a = "saucisse";
2 var valeura = function(){return a};
3 a = "chocolat";
4 valeura() // "chocolat"
```

- Principe: créer une fonctionnelle, qui renvoie la fonction désirée et l'appliquer à la variable fixée.

```
1 var a = "saucisse";
2 var valeura = (function(x){return () => x})(a);
3 a = "chocolat";
4 valeura() // "saucisse"
```

- Notion de fermeture (comme en APS).

Fermeture

- Une fermeture capture la liaison d'une variable, elle ne capture pas sa valeur (finalement, pas comme en APS).

```
1  function Compteur(depart){
2      var compte = depart;
3      return {
4          val: function (){return compte},
5          incr: function (){compte += 1},
6          decr: function (){compte -= 1},
7      }}
8  var x = 1
9  var c1 = new Compteur(x);
10 var c2 = new Compteur(1000);
11 x += 99;
12 var c3 = new Compteur(x);
13 c1.incr();
14 c1.val(); // 2
15 c2.val(); // 1000
16 c3.val(); // 100
```

Fermeture (II)

- Attention aux erreurs.

```
1 function repete(n,mot){
2     var repetitions = [];
3     var acc = ""
4     for(var i = 0; i < n; i++){
5         acc += mot;
6         repetitions[i] = function (){return acc}}
7     return repetitions}
8 var blabla = repete(10,"bla") //
9 blabla[2]() // "blablablablablablablablabla"
```

Continuations

- Langage fonctionnel: on peut prendre des fonctions en paramètres.

```
function applique(f,x){return f(x)}
var incr = x => x + 1;
applique(incr,3) // 4
function mapreduce(f,g,x,t){
    var res = [];
    for(e of t){res.push(f(e))};
    var acc = x;
    for(e of res){acc = g(acc,e)}
    return acc}
mapreduce(incr, (x,y)=> x + y, 0, [2,2,3,1,0]) // 13
```

- Utilisation courante: continuation

```
function plusk(x,y,k){return k(x+y)}
function id(x){return x}
function alerte(x){return alert(x)}
plusk(1,3,alerte) // alerte 4
plusk(1,3,id) // 4
```

- *Continuation Passing Style* du le monde fonctionnel.
  - La continuation encode le future (la pile).
  - $[A \to B] = A \to (B \to C) \to C$
  - Curry-Howard avec la logique classique.

UPMC SORBONNE UNIVERSITÉS

Callbacks (*suites*)

- Réalité des continuations en JS: fonctions de *callback*.

```
1 var acc = 0
2 function accu(x){acc += x}
3 function incrk(x,k){return k(x +1)}
4 [2,2,3,1,0].forEach(x => incrk(x,accu));
5 acc // 13
```

- ubiquité dans les API (couche 2):

```
1 object.onclick = function(){...};

3 $.ajax({
4   method: "GET",
5   url: "chanson",
6   data: { chanteur: "Annie Cordy", start : 1984, end : 1998 }
7 })
8   .done(function(resultat) {
9     ... );
10  });
```

- Enchaînement de callbacks.
  - la fonction de callback peut elle aussi déclencher un callback.

Callbacks (*suites*) (suite)

- Javascript est non-préemptif (Couche 1).
  - aucune gestion de la concurrence.
- Javascript est utilisé de manière intrinsèquement concurrente (Couche 2)
  - requêtes/réponses, AJAX, évènements, ...
- Toute la concurrence est gérée par les callback.
  - donner la suite d'actions à effectuer quand quelque chose (réponse, évènements) se produira.

Suites

```
1  function recupererk(cible, k){return k("Bonjour " + cible)}
2  // recupere une donnee (abstraction)

4  //Utilisation
5  recupererk("cible1", (x) => { if(x){return x} else {return "error"}})

7  //Utilisation double, sequentielle
8  recupererk("cible1", (x) => { if(x){
9                    return recupererk("cible2",
10                   (y) => {if(y){return [x,y]} else {return "error"}})
11                   } else {return "error"}})

13 //Utilisation double, parallele
14 [recupererk("cible1",x => x), recupererk("cible2",x => x)]
```

Suites

```
1  function recupererk(cible, k){return k("Bonjour " + cible)}
2  // recupere une donnee (abstraction)

4  //Utilisation
5  recupererk("cible1", (x) => { if(x){return x} else {return "error"}})

7  //Utilisation double, sequentielle
8  recupererk("cible1", (x) => { if(x){
9                    return recupererk("cible2",
10                   (y) => {if(y){return [x,y]} else {return "error"}})
11                   } else {return "error"}})

13 //Utilisation double, parallele
14 [recupererk("cible1",x => x), recupererk("cible2",x => x)]
```

```
1  //Utilisation double, sequentielle
2  function checkk(x, k){if(x){return k(x)} else {return "error"}}
3  recupererk("cible1", x => checkk(x, y => recupererk("cible2",
4                                          z => checkk(z, t => [y,t]))))
```

mot-clef this

`this` utilisé dans le corps d'une fonction pour faire référence à l'objet auquel la fonction appartient.

```
1  MyClass = function() { ... };

3  MyClass.prototype.myMethod = function() { ... this ... };

5  var myObject = new MyClass();

7  myObject.myMethod();
8  // this est myObject

10 var fn = myObject.myMethod;
11 fn(); // this n est pas myObject

13 var fn2 = myObject.myMethod.bind(myObject);
14 fn2(); // this est myObject,  liaison explicite
```

Sémantique: Valeurs de vérité

- ==: égalité (conversion de type),
- ===: identité (pas de conversion),
- `Object.is`: identité (comportement spécial).

```
1  "3" == 3 // vrai
2  "3" === 3 // faux
3  -0 === 0 // vrai
4  Object.is(-0,0) // faux
5  NaN === NaN // faux
6  Object.is(NaN, NaN) // vrai
7  0 == false // vrai
8  0 === false // faux
9  1 == true // vrai
10 1 === true // faux
11 [] == false // vrai
```

Types

- un seul type `number`
- type générique complexe `object`
- types “vides”:

```
1 typeof(null)  // "object"
2 typeof(Null) // "undefined"
```

- Polymorphismes faible: instances de typage

```
1 var x;
2 typeof(x) // "undefined"
3 var x = 5;
4 typeof(x) // "number"
5 var x = "Annie";
6 typeof(x) //"string"
```

- Erreur de typage: ordre.

Conversion et Empaquetage

- Javascript est agressif sur la conversion de type.
  - `"8" + "3"` vs. `"8" - "3"`
  - associativité de `+` ? `"saucisse" + 4 + 2`
- Possibilité d'empaqueter les valeurs de base dans des objets (*boxing*)
  - ajouter/retirer des propriétés/méthodes dynamiquement à des objets de base,
  - durée limitée:

```
1 var huit = 8;
2 huit.trois = 3;
3 (huit + huit.trois) // NaN
```

Objets

Les objets de Javascript sont référencés par des prototypes définis par des constructeurs là où d'autres langages utilisent des classes.

- un objet est une collection désordonnée de paires (clef,valeur).
- l'héritage est dynamique (un objet peut changer de parent à l'exécution).
- le prototype d'un objet est sa référence (il peut être modifié à l'exécution)
  - c'est la liste des méthodes ajoutées au constructeur à l'exécution.

```
1  function Personne(nom, prenom) { // Constructeur
2      this.nom = nom;
3      this.prenom = prenom;
4  }
5  function Chanteur(nom, prenom) { // Constructeur
6      this.nom = nom;
7      this.prenom = prenom;
8      this.chante = function (){return "Lalala";}
9  }
10 Personne.prototype.nationalite = "Belge";
11 var chanteuse = new Personne("Cordy", "Annie"); // Instance
12 "Elle est " + chanteuse.nationalite; // "Elle est Belge"
13 Personne.prototype.age = 89;
14 "Elle a " + chanteuse.age + " ans"; // "Elle a 89 ans"
```

Liaison dynamique du prototype

On peut changer dynamiquement le prototype d'un objet.

```
1  function Personne(nom, prenom) {
2      this.nom = nom;
3      this.prenom = prenom;
4  }
5  function Chanteur(nom, prenom) {
6      this.nom = nom;
7      this.prenom = prenom;
8      this.chante = function(){return "Lalala"};
9  }
10 Personne.prototype.nationalite = "Belge";
11 var chanteuse = new Personne("Cordy", "Annie");
12 chanteuse.nationalite // "Belge"
13 Object.setPrototypeOf(chanteuse, Chanteur.prototype);
14 chanteuse.nationalite // undefined
15 chanteuse.nom // "Cordy"
16 chanteuse.chante() // TypeError
17 Chanteur.prototype.chante = function(){return "Lololo"};
18 chanteuse.chante() // "Lololo"
```

Liaison dynamique du prototype

On peut changer dynamiquement le prototype d'un objet.

```
1  function Personne(nom, prenom) {
2      this.nom = nom;
3      this.prenom = prenom;
4  }
5  function Chanteur(nom, prenom) {
6      this.nom = nom;
7      this.prenom = prenom;
8      this.chante = function(){return "Lalala"};
9  }
10 Personne.prototype.nationalite = "Belge";
11 var chanteuse = new Personne("Cordy", "Annie");
12 chanteuse.nationalite // "Belge"
13 Object.setPrototypeOf(chanteuse, Chanteur.prototype);
14 chanteuse.nationalite // undefined
15 chanteuse.nom // "Cordy"
16 chanteuse.chante() // TypeError
17 Chanteur.prototype.chante = function(){return "Lololo"};
18 chanteuse.chante() // "Lololo"
```

On ne doit pas le faire (efficacité).

- Javascript utilise des classes cachées pour optimiser l'accès mémoire aux données.
- Ajouter/Retirer des propriétés dynamiquement à un objet met à jour sa classe cachée.

Classes cachées

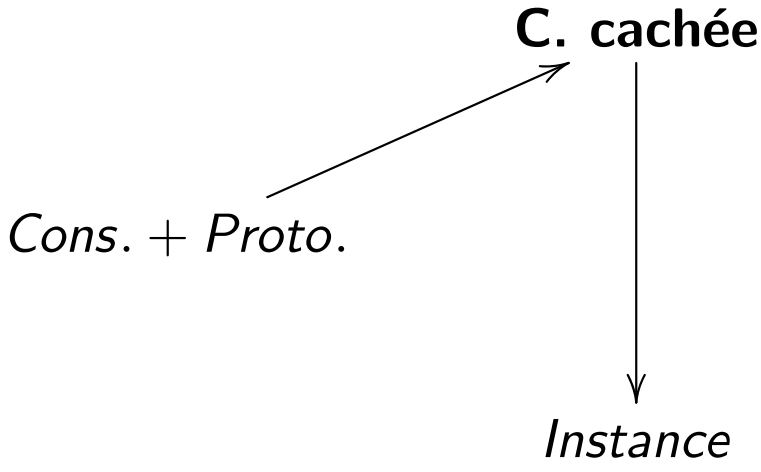
Attention à l'efficacité:

- m-à-j de classe cachée:

```
1 var a =
  [1, 2, 2.5, "annie"];

3 var a = new Array();
4 a[0] = 1;
5 a[1] = 2;
6 a[2] = 2.5;
7 a[3] = "annie"
```

- unification de classe cachée:

```
1 function Personne(nom, age){
2     this.nom = nom;
3     this.age = age;
4 }

6 var p1 =
  new Personne("Annie", 86);
7 var p2 =
  new Personne("Valerie", 88);

9 p2.est_academicien = true;
```



- Un prototype pour plusieurs classes cachées:

```
1 p1.prototype.nationalite =
  "Belge";
2 p2.nationalite
```

Héritage

Comment représenter l'héritage dans un langage à prototypes ?

- Copie du prototype:
- Copie des attributs:
- Chaînage de prototypes.
- Chaînage de prototypes (avec une doublure):

```
1 var extendClass = function(child, parent) {
2     var Surrogate = function() {};
3     Surrogate.prototype = parent.prototype;
4     child.prototype = new Surrogate();
5 };
```

  - utilisé dans la plupart des implémentations.

Manipulation du DOM (couche 2)

```
1 document.getElementById(id) // par nom (unique)
2 document.getElementsByTagName(nom) //  par etiquette (plusieurs)
3 document.createElement(nom) // creer un element
4 parentNode.appendChild(noeud) // binder un element
5 element.innerHTML // acceder au contenu de l element
6 element.setAttribute(nom,valeur) // ajouter un attribut
7 element.getAttribute(nom)
8 element.addEventListener(type, listener, usecapture)
```

Evènements

click, error, keydown, message, mousemove, keypress, offline,
load, focus, drag, drop, chargingchange, gamepadconnected, ...

Sécurité

- Langage de script:
  - support pour attaques, scripts malicieux.
- Manipule le DOM
  - opérations dangereuses cachées au sein d'une page.
- Code envoyé non compilé.
  - peu de code propriétaire.

Same Origin Policy: uniquement les scripts exécutés sur des pages d'un même site peuvent accéder à leurs DOMs.

Environnement global

Javascript est exécuté dans un environnement global qui correspond au navigateur (restriction de portée)

- Windows Script Host (ransomware, ...)

Minification de JavaScript

- retirer du code source tous les caractères superflus sans en changer la sémantique.

- Pourquoi:
  - accélérer le téléchargement, l'analyse syntaxique et l'évaluation.
- Comment:
  - enlever tous les espacements et les commentaires.
  - changer les noms de variables:

```
1 function sum(num1, num2) {
2     return num1 + num2;
3 }
```

devient

```
1 function sum(A,B){return A+B;}
```

  - retirer du code mort.
- Outils: Closure, YUI Compressor, minify.

Conclusion

Un langage de syntaxe simple et de sémantique complexe.

- beaucoup de sources
  - O'Reily, *Javascript*
  - Encyclopédie *Techniques de l'Ingénieur*, *Javascript*, C. Queinnec,
  - Tutoriels sur le net (attention),
  - Blogs de *web developers*.
- utilisation de librairies de haut-niveau (JQuery, outils d'AngularJS),
- formalisation: en cours, mais difficile.