SupGalilée info-1

2023-2024

Programmation Web
Partie 1 : JavaScript

Étienne André

Université Sorbonne Paris Nord
Etienne.Andre@univ-paris13.fr

Version diapositives à trous : 29 avril 2024

Outline

JavaScript

- Langage de programmation de scripts
- Créé en

JavaScript

- Langage de programmation de scripts
- Créé en
- Surtout utilisé dans les pages Web
  - interprété par le navigateur Web
  - donc une exécution côté client

JavaScript

- Langage de programmation de scripts
- Créé en
- Surtout utilisé dans les pages Web
  - interprété par le navigateur Web
  - donc une exécution côté client

⚠ Pas de rapport avec Java

JavaScript n'a (aujourd'hui) aucun rapport avec le langage Java

Versions de JavaScript

- Deux versions principales cohabitent : ES5 et 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
- Presque tout ES6 est implanté dans les navigateurs récents, mais pas les anciens

Principales caractéristiques de JavaScript

- Langage interprété

Principales caractéristiques de JavaScript

- Langage interprété
- Langage objet à prototype
  - Les objets ne sont pas (forcément) des instances de classes
  - Les fonctions sont des objets de première classe (utilisables sans restrictions, y compris comme paramètres)

Principales caractéristiques de JavaScript

- Langage interprété
- Langage objet à prototype
  - Les objets ne sont pas (forcément) des instances de classes
  - Les fonctions sont des objets de première classe (utilisables sans restrictions, y compris comme paramètres)
- Langage impératif (et fonctionnel !)
  - autres langages impératifs :

Principales caractéristiques de JavaScript

- Langage interprété
- Langage objet à prototype
  - Les objets ne sont pas (forcément) des instances de classes
  - Les fonctions sont des objets de première classe (utilisables sans restrictions, y compris comme paramètres)
- Langage impératif (et fonctionnel !)
  - autres langages impératifs :
  - autres langages fonctionnels :

Principales caractéristiques de JavaScript

- Langage interprété
- Langage objet à prototype
  - Les objets ne sont pas (forcément) des instances de classes
  - Les fonctions sont des objets de première classe (utilisables sans restrictions, y compris comme paramètres)
- Langage impératif (et fonctionnel !)
  - autres langages impératifs :
  - autres langages fonctionnels :

Principales caractéristiques de JavaScript : typage

- Langage faiblement typé
  - autorisé :

```
1 console.log("20" + 46); // affiche "2046"
```

  - autres exemples :

Principales caractéristiques de JavaScript : typage

- Langage faiblement typé
  - autorisé :

```
1 console.log("20" + 46); // affiche "2046"
```

  - autres exemples :
  - contre-exemples (fortement typés) :

Principales caractéristiques de JavaScript : typage

- Langage faiblement typé
  - autorisé :

```
1 console.log("20" + 46); // affiche "2046"
```

  - autres exemples :
  - contre-exemples (fortement typés) :

Principales caractéristiques de JavaScript : typage

- Langage faiblement typé
  - autorisé :

```
1 console.log("20" + 46); // affiche "2046"
```

  - autres exemples :
  - contre-exemples (fortement typés) :

- Langage à typage dynamique
  - autorisé :

```
1 let ma_var = 2046;
2 ma_var = 'ITMFL'; // pas d'erreur de typage
```

  - autres exemples :

Principales caractéristiques de JavaScript : typage

- Langage faiblement typé
  - autorisé :

```
1 console.log("20" + 46); // affiche "2046"
```

  - autres exemples :
  - contre-exemples (fortement typés) :

- Langage à typage dynamique
  - autorisé :

```
1 let ma_var = 2046;
2 ma_var = 'ITMFL'; // pas d'erreur de typage
```

  - autres exemples :
  - contre-exemples (typage statique) :

Principales caractéristiques de JavaScript : typage

- Langage faiblement typé
  - autorisé :

```
1 console.log("20" + 46); // affiche "2046"
```

  - autres exemples :
  - contre-exemples (fortement typés) :

- Langage à typage dynamique
  - autorisé :

```
1 let ma_var = 2046;
2 ma_var = 'ITMFL'; // pas d'erreur de typage
```

  - autres exemples :
  - contre-exemples (typage statique) :

Contexte de ce cours

Contours du cours

Ce cours s'intéresse à l'utilisation de JavaScript dans un contexte de programmation Web, aux côtés de HTML et CSS.

Outline

Bref aperçu de la syntaxe

```
1  // Commentaire
2  let x;              // déclaration de variable
3  let y = 2046;       // déclaration + initialisation de variable
4  var z = 'ITMFL'; // ancienne notation (ES5)
5  x = y + 1;          // affectation
6
7  foo(x, y);          // appel de fonction avec args x,y
8  objet.bar(z);       // appel de la méthode 'bar' de 'objet'
9
10 // conditionnelle
11 if(x === 0){        // x est-elle égale à 0 ?
12   x += 42;
13 }
14
15 // itérations
16 while (x > 0){
17   x = x-1;
18 }
19 for (let i = 0; i<10; i++){
20   console.log(i);
21 }
```

Mode strict

- JavaScript possède 2 modes de fonctionnement
  1. le mode normal ou `sloppy`
  2. le mode strict
- Le mode strict peut être activé :
  - pour un fichier complet avec en début le tag suivant :

```
'use strict'
```

  - par fonction :

```
function fonctionEnModeStrict() {
  'use strict';
  ...
}
```

- Le mode strict génère plus de *warnings* et d'erreurs mais est souvent plus rapide

Outline

Variables

- Doivent être déclarées (en mode strict)

```
let v;
```

- Peuvent être initialisées à la déclaration

```
let v = 2046;
```

- Sont typées dynamiquement (pas de typage à la déclaration)

```
let v = 2046;
v = "WKW";
v = false;
```

- Syntaxe des noms de variable : $(\textit{lettre} \mid \$ \mid \_)(\textit{lettre} \mid \textit{chiffre} \mid \$ \mid \_)^*$

```
// Exemples de déclarations de variables
let i, $v, _x, _12m, $2046, THX1138, _2001;
```

Déclaration et portée des variables

Deux syntaxes de déclarations de variables

- Mot-clé `var` (ES5) : la portée (*scope*) est l'ensemble de la fonction où elle est déclarée
- Mot-clé `let` (ES6) : la portée (*scope*) est limitée au bloc où elle est déclarée
- Une variable déclarée en dehors de tout bloc est globale

```
function f(){
  var v = 2046;
  if (v < 3000){
    let w = 3000-2046;
    var z = w+v;
    console.log(v, w); // OK
  }
  console.log(v,z,w);  // pas OK
}
```

Déclaration et portée des variables

Deux syntaxes de déclarations de variables

- Mot-clé `var` (ES5) : la portée (*scope*) est l'ensemble de la fonction où elle est déclarée
- Mot-clé `let` (ES6) : la portée (*scope*) est limitée au bloc où elle est déclarée
- Une variable déclarée en dehors de tout bloc est globale

```
function f(){
  var v = 2046;
  if (v < 3000){
    let w = 3000-2046;
    var z = w+v;
    console.log(v, w); // OK
  }
  console.log(v,z,w);  // pas OK
}
```

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

Constantes

Constante : valeur non modifiable

- Mot-clé `const`

Exemples :

```
1 // Constante globale
2 const DUREE_INTERVALLE = 1000;
3
4 window.setInterval(auditeur, DUREE_INTERVALLE);
```

```
1 const noeud = document.getElementById("photo");
2
3 noeud_image.addEventListener(auditeur);
```

Outline

Les valeurs

- Valeurs primitives
  - booléens : `true`, `false`
  - nombres : `2046`, `3.141592653`
  - chaînes : `'Quentin'`, `"Tarantino"`
  - `undefined`, `null`
- Objets
  - Objets classiques : `{ prenom : 'Quentin', nom : 'Tarantino' }`
  - Tableaux : `[ 2, 0, 4, 6 ]`
  - Fonctions et expressions régulières
- Les valeurs ont un type
  - Voir l'opérateur `typeof`

```
> typeof 2046
"number"
> typoeof {}
"object"
> typeof undefined
"undefined"
```

Les valeurs primitives

- Valeurs primitives : objets non modifiables
  - Possèdent des propriétés et méthodes
  - Ne peuvent pas être changées
- Comparaison par valeur

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

```
# attention :
    transtypage
> 2046 === "2046"
false
> 2046 == "2046"
true
# éviter d'utiliser
    ==
```

Les nombres

- Les nombres sont tous des flottants
- Opérateurs : `+`, `-`, `/`, `*`, `++`, `--`, `%`
- Deux valeurs spéciales : `Infinity`, `NaN` (*not a number*)

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

Les nombres flottants : danger !

⚠ Danger avec les nombres flottants

Attention aux débordements et erreurs d'approximation !

```
1 let a = 1;
2 let petit = 0.001;
3 while(a !== 0){
4   a = a - petit;
5 }
```

Pourquoi ce programme ne termine-t-il pas ?

Les nombres flottants : danger !

⚠ Danger avec les nombres flottants

Attention aux débordements et erreurs d'approximation !

```
1 let a = 1;
2 let petit = 0.001;
3 while(a !== 0){
4   a = a - petit;
5 }
```

Pourquoi ce programme ne termine-t-il pas ?

Les nombres flottants : danger !

⚠ Danger avec les nombres flottants

Attention aux débordements et erreurs d'approximation !

```
1 let a = 1;
2 let petit = 0.001;
3 while(a !== 0){
4   a = a - petit;
5 }
```

Pourquoi ce programme ne termine-t-il pas ?

Les nombres : incréments

- Incrémenter la valeur d'une variable numérique en JavaScript
  - Même mécanisme que nombreux autres langages (C, C++, Java, Python…)

Syntaxe `i++` :

Les nombres : incréments

- Incrémenter la valeur d'une variable numérique en JavaScript
  - Même mécanisme que nombreux autres langages (C, C++, Java, Python…)

Syntaxe `i++` :

Syntaxe `++i` :

Les nombres : incréments

- Incrémenter la valeur d'une variable numérique en JavaScript
  - Même mécanisme que nombreux autres langages (C, C++, Java, Python…)

Syntaxe `i++` :

Syntaxe `++i` :

```
1 let i = 0;
2 console.log(i++); // affiche 0 (puis incrémente i à 1)
3 console.log(i);   // affiche 1
4
5 let j = 0;
6 console.log(++j); // affiche 1
7 console.log(j);   // affiche 1
```

(et idem avec « `i--` » et « `--i` »)

Les booléens

- Deux valeurs : `true`, `false`
- Valeurs fausses : `undefined`, `null`, `false`, `0`, `NaN`, `''`
- Opérateurs à résultat booléen :
  - Comparaisons : `<`, `<=`, `>`, `>=`, `===`, `!==` (ainsi que `==`, `!=` : à éviter)
  - Connecteurs booléens : `!`, `&&`, `||` (opérateurs court-circuit)

```
# illustration du court-circuit : plop() n'est jamais
    appelée ci-dessous :
> false && plop()
false
> true || plop()
true
```

Les chaînes de caractères

- Deux syntaxes de guillemets : `'simples'` ou `"doubles"`

```
let prenom = "Quentin";
let nom    = 'Tarantino';
```

- Concaténation

```
console.log(prenom + " " + nom);
// affiche "Quentin Tarantino"
```

- Propriétés

```
console.log(prenom.length); // affiche 7
```

- Méthodes, par ex :

```
console.log(nom.toUpperCase()); // affiche "TARANTINO"
```

Les littéraux de gabarits

- Expression de chaîne de caractères permettant d'inclure des variables et des expressions évaluées au moment de l'affectation
  - En anglais : *string literals* ou *string templates*
  - Depuis ES6
  - Encadrés par des caractères accent grave (*backtick* ou *backquote*)

```
let nom        = 'Sciamma';
let prenom     = 'Céline';
let naissance  = 1978;
let annee_courante = new Date().getFullYear();
let phrase = `La réalisatrice ${prenom + ' '+ nom} a
    cette année ${annee_courante - naissance} ans`;
console.log(phrase);
// affiche "La réalisatrice Céline Sciamma a cette
    année 46 ans"
```

Outline

Les structures de contrôle

Les structures de contrôle en JavaScript sont similaires à celles rencontrées dans de nombreux langages impératifs

- Conditionnelle : `if (...) {...} else {...}`

```
const langage = 'JavaScript';
if(langage === 'HTML') {
  console.log('Langage statique');
}else{
  console.log('Langage dynamique');
}
```

Les structures de contrôle : `while`

- boucle `while(){...}`

```
1 let i = 2;
2 while(i < 100) {
3   console.log(i);
4   i = i * i;
5 } // affiche ?
```

Les structures de contrôle : `while`

- boucle `while(){...}`

```
1 let i = 2;
2 while(i < 100) {
3   console.log(i);
4   i = i * i;
5 } // affiche ?
```

Les structures de contrôle : `while`

- boucle `while(){...}`

```
1 let i = 2;
2 while(i < 100) {
3   console.log(i);
4   i = i * i;
5 } // affiche ?
```

Les structures de contrôle : `while`

- boucle `while(){...}`

```
1 let i = 2;
2 while(i < 100) {
3   console.log(i);
4   i = i * i;
5 } // affiche ?
```

Les structures de contrôle : `do { ... } while`

- boucle `do { ... } while(...)`
  - La condition est évaluée après l'exécution du corps de boucle

- Exemple

```
1  let i = 20;
2  while(i < 10) {
3    console.log(i);
4    i = i * i;
5  }
```

Affiche :

```
1  let i = 20;
2  do {
3      console.log(i);
4      i = i * i;
5  } while(i < 10);
```

Affiche :

Les structures de contrôle : `do { ... } while`

- boucle `do { ... } while(...)`
  - La condition est évaluée après l'exécution du corps de boucle

- Exemple

```
1  let i = 20;
2  while(i < 10) {
3    console.log(i);
4    i = i * i;
5  }
```

Affiche :

```
1  let i = 20;
2  do {
3      console.log(i);
4      i = i * i;
5  } while(i < 10);
```

Affiche :

Les structures de contrôle : `do { ... } while`

- boucle `do { ... } while(...)`
  - La condition est évaluée après l'exécution du corps de boucle

- Exemple

```
1  let i = 20;
2  while(i < 10) {
3    console.log(i);
4    i = i * i;
5  }
```

Affiche :

```
1  let i = 20;
2  do {
3      console.log(i);
4      i = i * i;
5  } while(i < 10);
```

Affiche :

Les structures de contrôle : `for`

- boucle `for(...){...}`

```
1 for(i = 3; i < 6; i++) {
2   console.log(i);
3 } // affiche ?
```

Les structures de contrôle : `for`

- boucle `for(...){...}`

```
1 for(i = 3; i < 6; i++) {
2   console.log(i);
3 } // affiche ?
```

Les structures de contrôle : `for`

- boucle `for(...){...}`

```
1 for(i = 3; i < 6; i++) {
2   console.log(i);
3 } // affiche ?
```

Les structures de contrôle : `for`

- boucle `for(...){...}`

```
1 for(i = 3; i < 6; i++) {
2   console.log(i);
3 } // affiche ?
```

Les structures de contrôle : switch

- Objectif : effectuer un traitement particulier sur une valeur en énumérant des cas possibles

```
let nourriture = window.prompt("Que voulez-vous manger ?", "asperges");
switch (nourriture) {
  case 'asperges':
  case 'aubergines':
    console.log("Vous aimez les légumes");
    break;
  case 'oranges':
  case 'pommes':
    console.log("Vous aimez les fruits");
    break;
  case 'kouign-amann':
    console.log("Vous aimez le beurre");
    break;
  default:
    console.log("Vous aimez des choses bizarres");
}
```

Outline

Les fonctions en JavaScript

- Une fonction JavaScript est un objet ; par conséquent, elle
  - peut être créée via un littéral
  - peut être affectée à des variables, tableaux, propriétés
  - peut être passée en paramètre à d'autres fonctions
  - peut être retournée comme résultat d'une autre fonction
  - possède des propriétés et des méthodes
  - peut être invoquée en plaçant des parenthèses :

```
1 ma_fonction();
```

Types des fonctions

Plusieurs types de fonctions :

1. Les fonctions natives appartenant au langage JavaScript
   - Exemples : `parseInt()`, `isNaN()`, `eval()`...
2. Les fonctions (méthodes) relatives à des objets de JavaScript
   - Exemples : `Math.random()`, `Math.ceil()`, `window.alert()`, `window.prompt()`, `document.write()`...
3. Les fonctions définies dans le script

```
1 function age(naissance){
2   let annee_courante = new Date().getFullYear();
3   return annee_courante - naissance;
4 }
```

Fonctions natives

1. Conversions de types :
   - `parseInt()` : conversion d'une chaîne de caractères en entiers
   - `parseFloat()` : conversion d'une chaîne de caractères en réels (renvoie `NaN` si ce n'est pas un nombre)

2. Détection de valeurs exceptionnelles :
   - `isNaN()` : indique si l'argument est `NaN`
   - `isFinite()` : indique si l'argument est une valeur numérique finie

3. Code d'évaluation : fonction `eval()`
   - analyse une chaîne et l'exécute comme s'il s'agissait d'un script JavaScript

Fonctions natives

1. Conversions de types :
   - `parseInt()` : conversion d'une chaîne de caractères en entiers
   - `parseFloat()` : conversion d'une chaîne de caractères en réels (renvoie `NaN` si ce n'est pas un nombre)

2. Détection de valeurs exceptionnelles :
   - `isNaN()` : indique si l'argument est `NaN`
   - `isFinite()` : indique si l'argument est une valeur numérique finie

3. Code d'évaluation : fonction `eval()`
   - analyse une chaîne et l'exécute comme s'il s'agissait d'un script JavaScript

⚠ Fonction `eval`

La fonction `eval` de JavaScript peut être dangereuse : possibilité d'exécuter du code JavaScript, potentiellement néfaste.

Fonctions relatives à des objets JavaScript

- Exemple : l'objet `window`

```
1 window.alert("Message à afficher");
```

Affiche une boîte de dialogue avec la chaîne de caractères en paramètre

```
1 let resultat = window.prompt("Message affiché",
    "Valeur par défaut affichée");
```

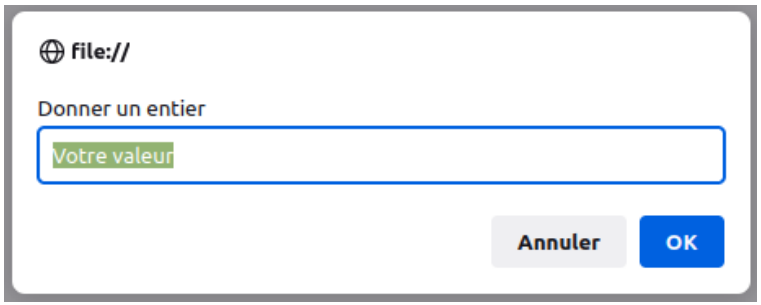
Affiche une boîte de dialogue attendant une réponse de l'internaute qui sera stockée dans la chaîne de caractères `resultat`

Fonctions natives : exemple

```
let nombreChaine = window.prompt("Donner un entier", "Votre valeur");
let nombre = parseInt(nombreChaine);
if(nombre % 2 === 0){
  document.write('<p>' + nombre + ' : entier saisi pair</p>');
}else{
  document.write('<p>' + nombre + ' : entier saisi impair</p>');
}
```

Fonctions natives : exemple

```
1 let nombreChaine = window.prompt("Donner un entier", "Votre
    valeur");
2 let nombre = parseInt(nombreChaine);
3 if(nombre % 2 === 0){
4   document.write('<p>' + nombre + ' : entier saisi
    pair</p>');
5 }else{
6   document.write('<p>' + nombre + ' : entier saisi
    impair</p>');
7 }
```



Exemple : demander un entier valide à l'internaute

```
let nombre = NaN;
do{
  nombre = window.prompt("Entrer un entier", '');
}while(nombre === '' || isNaN(nombre));
nombre = parseInt(nombre);
```

La déclaration de fonctions

- le mot-clé `function`
- un nom
- une liste (possiblement vide) de paramètres entre `( ... )`
- une suite (possiblement vide) d'instructions entre `{ ... }`

```
1  // déclaration d'une fonction de somme
2  function somme (a, b) {
3    return a+b ;
4  }
5  console.log(somme.name);        // affiche "somme"
6  console.log(somme(2000, 46)); // affiche "2046"
7
8  console.log(somme);
9  // affiche:
10 // function somme (a, b) {
11 //   length: 2
12 //   name: "somme"
13 //   prototype: Object { ... }
14 //   ...
```

Expressions de fonctions et fonctions anonymes

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

```
1 let soustraction = function(a, b) {
2   return a-b;
3 }
4 console.log(soustraction.name); // affiche "soustraction"
5 console.log(soustraction(2050, 4)); // affiche 2046
```

Expressions de fonctions avec =>

- Depuis ES6 : nouvelle notation pour les expressions de fonctions avec =>

```
1 let produit = (a,b) => {return a*b;}
2 let wkw     = ()    => {return 2046;}
3 let incrBy1 = (x)   => x+1;
4 console.log(wkw);               // affiche "function
      wkw(){...}"
5 console.log(wkw());             // affiche "2046"
6 console.log(wkw.name);          // affiche "wkw"
7 console.log(produit(31,66)); // affiche "2046"
```

Invocation immédiate

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

```
1  // f1, f2, f3 sont des nombres (pas des fonctions !)
2  let f1 = (function return42() {return 42;}) ();
3  let f2 = (function () {return 300;}) ();
4  let f3 = ((x) => 31 * x) (66);
5  // f4 est une fonction
6  let f4 = ((x) => 31 * x); // pas d'invocation immédiate
7  console.log(f1);      // affiche "42"
8  console.log(f2);      // affiche "300"
9  console.log(f3);      // affiche "2046"
10 console.log(f4(66)); // affiche "2046"
```

Arguments des fonctions

- Lors d'un appel, le nombre d'arguments peut être différent du nombre de paramètres
  - arguments manquants : valeur `undefined`
  - tous les arguments : dans le pseudo-tableau `arguments`

```
1 function f(x, y) {
2   console.log("x=" + x, "; y=" + y + "; args=");
3   console.log(arguments);
4 }
5 f(1,2);    // affiche : "x=1; y=2; args=[1,2]"
6 f(1,2,3); // affiche : "x=1; y=2; args=[1,2,3]"
7 f(1);      // affiche : "x=1; y=undefined; args=[1]"
```

Arguments des fonctions : valeur par défaut

- Depuis ES6 : possibilité de définir une valeur par défaut
- Notation explicite pour les paramètres optionnels
  - syntaxe : `argument=valeur_par_defaut`
- Notation explicite pour les arguments supplémentaires dans un tableau
  - syntaxe : `...nom_du_tableau`

```
1  // Si l'argument "y" est absent, il prend pour valeur 2046
2  // Tous les arguments supplémentaires seront stockés dans
       le tableau les_autres
3  let g = (x, y=2046, ...les_autres) =>
4  {
5    console.log("x=" + x, "; y=" + y + "; les_autres=");
6    console.log(les_autres);
7  }
8  g();       // affiche : "x=undefined; y=2046; les_autres=[]"
9  g(1);      // affiche : "x=1;         y=2046; les_autres=[]"
10 g(1,2);    // affiche : "x=1;         y=2;    les_autres=[]"
11 g(1,2,3); // affiche : "x=1;         y=2;    les_autres=[3]"
```

Fonction en paramètre (*callback*)

- Rappel : Une fonction est un objet et donc peut être passée en paramètre

```
1  // Définitions équivalentes
2  function somme1(a,b) {return a+b;}
3  let somme2 = (a,b) => a+b;
4
5  // Définition d'une fonction prenant une fonction f en
       paramètre :
6  function calculer(a, b, f) {return f(a,b);}
7
8  // On appelle calculer en lui passant une fonction en
       argument
9  console.log(calculer(20, 22, somme1));        // affiche 42
10 console.log(calculer(2000, 46, somme2));      // affiche 2046
11 // On peut passer une expression comme argument
12 console.log(calculer(31, 66, (i,j) => i*j)); // affiche 2046
```

Fonction comme résultat

- Rappel : Une fonction est un objet et donc peut être retournée comme résultat d'une fonction

```
1  function incrementeur(x) {
2    // Ici on retourne une fonction à un argument "a"
3    return (a) => { return a+x;};
4  }
5
6  let incr5 = incrementeur(5);
7  // incr5 est une fonction qui incrémente de 5
8
9  console.log(incr5(7));                  // affiche 12
10 console.log(incr5(2041));               // affiche 2046
11 console.log(incrementeur(2)(2044)); // affiche 2046
```

Portée des fonctions

- Une fonction ou une variable peut être déclarée :
  - à tout moment
  - dans n'importe quelle fonction
- Rappel sur la portée (*scope*) des variables
  - Mot-clé `var` (ES5) : ensemble de la fonction où elle est déclarée
  - Mot-clé `let` (ES6) : bloc où elle est déclarée
- Portée (*scope*) des fonctions
  - tout le bloc de déclaration, ainsi que les fonctions et blocs imbriqués
  - les références en avant sont possibles

Portée des fonctions : exemple

```
function portee_fonctions() {
  console.log('incrBy2(2046):', incrBy2(2046)); // OK

  function incrBy2(x) {return x+2 ;}
  if(true){
    console.log('a:', a);  // ReferenceError
    console.log('b:', b);  // affiche "undefined"
    let a = 10;
    console.log('incrByA(6):', incrByA(6));
    function incrByA(x) {
      console.log('incrBy2(100):', incrBy2(100));
      return x + a;
    }
    var b = 8;
    console.log('incrByA(3):', incrByA(3));
  }
  console.log('a:', a);   // ReferenceError
  console.log('b:', b);
  console.log(incrByA(5)) // ReferenceError
  console.log('incrBy2(40):', incrBy2(40));
}
```

Clôture (*closure*)

- Clôture (*closure*) d'une fonction : portée au sein de laquelle elle est déclarée
  - Inclut l'accès à tous les objets (variables) présents dans cette portée
- La clôture est toujours accessible lors de l'exécution de la fonction, même si la portée a disparu ou si la fonction est exécutée dans une portée différente !
- La fonction conserve une référence vers tous les objets présents dans sa portée initiale de déclaration

Clôture : exemple

- La portée de la fonction `f` consiste en tout le corps (lignes 1 à 4)
  - ce qui inclut la variable `i`... et l'argument `n`
- ... et consiste aussi en tout le bloc englobant `f` !

```
1  function f(n) {
2    let i=1;
3    return (x) => (x + i++) * n;
4  }
5
6  // f10 est une fonction
7  let f10 = f(10);
8  // donc on peut invoquer f10:
9  console.log(f10(5));
10 console.log(f10(5));
11 console.log(f10(5));
12 // ou créer une nouvelle fonction :
13 console.log(f(10)(5));
14 console.log(f(10)(5));
```

- La valeur de `i` est modifiée à chaque appel de `f10`

Affichages successifs :

Clôture : exemple

- La portée de la fonction `f` consiste en tout le corps (lignes 1 à 4)
  - ce qui inclut la variable `i`... et l'argument `n`
- ... et consiste aussi en tout le bloc englobant `f` !

```
1  function f(n) {
2    let i=1;
3    return (x) => (x + i++) * n;
4  }
5
6  // f10 est une fonction
7  let f10 = f(10);
8  // donc on peut invoquer f10:
9  console.log(f10(5));
10 console.log(f10(5));
11 console.log(f10(5));
12 // ou créer une nouvelle fonction :
13 console.log(f(10)(5));
14 console.log(f(10)(5));
```

- La valeur de `i` est modifiée à chaque appel de `f10`

Affichages successifs :

Clôture : exemple

- La portée de la fonction `f` consiste en tout le corps (lignes 1 à 4)
  - ce qui inclut la variable `i`... et l'argument `n`
- ... et consiste aussi en tout le bloc englobant `f` !

```
1  function f(n) {
2    let i=1;
3    return (x) => (x + i++) * n;
4  }
5
6  // f10 est une fonction
7  let f10 = f(10);
8  // donc on peut invoquer f10:
9  console.log(f10(5));
10 console.log(f10(5));
11 console.log(f10(5));
12 // ou créer une nouvelle fonction :
13 console.log(f(10)(5));
14 console.log(f(10)(5));
```

- La valeur de `i` est modifiée à chaque appel de `f10`

Affichages successifs :

Clôture : exemple

- La portée de la fonction `f` consiste en tout le corps (lignes 1 à 4)
  - ce qui inclut la variable `i`... et l'argument `n`
- ... et consiste aussi en tout le bloc englobant `f` !

```
1  function f(n) {
2    let i=1;
3    return (x) => (x + i++) * n;
4  }
5
6  // f10 est une fonction
7  let f10 = f(10);
8  // donc on peut invoquer f10:
9  console.log(f10(5));
10 console.log(f10(5));
11 console.log(f10(5));
12 // ou créer une nouvelle fonction :
13 console.log(f(10)(5));
14 console.log(f(10)(5));
```

- La valeur de `i` est modifiée à chaque appel de `f10`

Affichages successifs :

Clôture : exemple

- La portée de la fonction `f` consiste en tout le corps (lignes 1 à 4)
  - ce qui inclut la variable `i`... et l'argument `n`
- ... et consiste aussi en tout le bloc englobant `f` !

```
1  function f(n) {
2    let i=1;
3    return (x) => (x + i++) * n;
4  }
5
6  // f10 est une fonction
7  let f10 = f(10);
8  // donc on peut invoquer f10:
9  console.log(f10(5));
10 console.log(f10(5));
11 console.log(f10(5));
12 // ou créer une nouvelle fonction :
13 console.log(f(10)(5));
14 console.log(f(10)(5));
```

- La valeur de `i` est modifiée à chaque appel de `f10`

Affichages successifs :

Clôture : exemple

- La portée de la fonction `f` consiste en tout le corps (lignes 1 à 4)
  - ce qui inclut la variable `i`... et l'argument `n`
- ... et consiste aussi en tout le bloc englobant `f` !

```
1  function f(n) {
2    let i=1;
3    return (x) => (x + i++) * n;
4  }
5
6  // f10 est une fonction
7  let f10 = f(10);
8  // donc on peut invoquer f10:
9  console.log(f10(5));
10 console.log(f10(5));
11 console.log(f10(5));
12 // ou créer une nouvelle fonction :
13 console.log(f(10)(5));
14 console.log(f(10)(5));
```

- La valeur de `i` est modifiée à chaque appel de `f10`

Affichages successifs :

Clôture : utilisations (1/2)

- Pour créer des variables partagées non globales

```
1 function sommeTab(tableau){
2   let somme = 0; // variable de somme
3   // f est une fonction qui incrémente somme de e
4   let f = (e) => {somme += e;};
5   tableau.forEach(f); // appel de f sur chaque élément
6   return somme;
7 }
8 console.log(sommeTab([1,2,3,4,5]));
```

Affichage dans la console :

Clôture : utilisations (1/2)

- Pour créer des variables partagées non globales

```
1 function sommeTab(tableau){
2   let somme = 0; // variable de somme
3   // f est une fonction qui incrémente somme de e
4   let f = (e) => {somme += e;};
5   tableau.forEach(f); // appel de f sur chaque élément
6   return somme;
7 }
8 console.log(sommeTab([1,2,3,4,5]));
```

Affichage dans la console :

Clôture : utilisations (2/2)

- Pour « piéger » une variable

```
1  function sequence(u0){
2    let u = u0;
3    return () => u++;
4  }
5  let suivant1  = sequence(1);
6  let suivant10 = sequence(10);
7
8  console.log(suivant1());
9  console.log(suivant10());
10 console.log(suivant10());
11 console.log(suivant1());
12 console.log(suivant10());
```

Affichages :

Clôture : utilisations (2/2)

- Pour « piéger » une variable

```
1  function sequence(u0){
2    let u = u0;
3    return () => u++;
4  }
5  let suivant1  = sequence(1);
6  let suivant10 = sequence(10);
7
8  console.log(suivant1());
9  console.log(suivant10());
10 console.log(suivant10());
11 console.log(suivant1());
12 console.log(suivant10());
```

Affichages :

Clôture : utilisations (2/2)

- Pour « piéger » une variable

```
1  function sequence(u0){
2    let u = u0;
3    return () => u++;
4  }
5  let suivant1  = sequence(1);
6  let suivant10 = sequence(10);
7
8  console.log(suivant1());
9  console.log(suivant10());
10 console.log(suivant10());
11 console.log(suivant1());
12 console.log(suivant10());
```

Affichages :

Clôture : utilisations (2/2)

- Pour « piéger » une variable

```
1  function sequence(u0){
2    let u = u0;
3    return () => u++;
4  }
5  let suivant1  = sequence(1);
6  let suivant10 = sequence(10);
7
8  console.log(suivant1());
9  console.log(suivant10());
10 console.log(suivant10());
11 console.log(suivant1());
12 console.log(suivant10());
```

Affichages :

Clôture : utilisations (2/2)

- Pour « piéger » une variable

```
1  function sequence(u0){
2    let u = u0;
3    return () => u++;
4  }
5  let suivant1  = sequence(1);
6  let suivant10 = sequence(10);
7
8  console.log(suivant1());
9  console.log(suivant10());
10 console.log(suivant10());
11 console.log(suivant1());
12 console.log(suivant10());
```

Affichages :

Clôture : utilisations (2/2)

- Pour « piéger » une variable

```
1  function sequence(u0){
2    let u = u0;
3    return () => u++;
4  }
5  let suivant1  = sequence(1);
6  let suivant10 = sequence(10);
7
8  console.log(suivant1());
9  console.log(suivant10());
10 console.log(suivant10());
11 console.log(suivant1());
12 console.log(suivant10());
```

Affichages :

Clôture : l'exemple tordu ultime

- Attention à la clôture (et donc à la portée des variables locales ou globales !)

```
// Déclarations
function incrementeurBizarre(valeur_init){
  let u = valeur_init;
  return () => {
    pas++; v++; u += pas;
    console.log("pas = " + pas + " ; v = " + v + " ; u = " + u);
  }
}

let v = 0; // variable globale
if(true){
  var pas = 0; // variable globale car `var`
}

// Tests
let iB5   = incrementeurBizarre(5);
let iB100 = incrementeurBizarre(100);

iB5();   // affiche ?
iB5();   // affiche ?
iB100(); // affiche ?
iB100(); // affiche ?
iB5();   // affiche ?
```

Outline

Objets : constructeur, propriétés, méthodes

- Éventuel constructeur :

```
let maintenant = new Date();
```

- Propriétés

```
monObjet.ma_propriete;
```

- Méthodes

```
maintenant.getHours();
```

Exemple :

```
let maintenant = new Date();
console.log("Il est " + maintenant.getHours() + "h" +
    maintenant.getMinutes());
// affiche par exemple "Il est 20h46"
```

Les objets en JavaScript

- Objets natifs
  - Exemples : `Math`, `String`, `Array`, `Date`, `Number`...
- Objets liés aux pages Web
  - Exemples : `document`, `navigator`, `window`...
- Objets créés par le script

Un exemple d'objet natif : l'objet `Math`

- Quelques propriétés

```
1 Math.E; // le nombre d'Euler
2 Math.PI;
```

- Quelques méthodes
  - `floor` : Renvoie le plus grand nombre entier plus petit ou égal à un nombre
  - `max` : Renvoie le maximum d'un ensemble de nombres
  - `min` : Renvoie le minimum d'un ensemble de nombres
  - `random` : Renvoie un nombre pseudo-aléatoire dans $[0, 1]$
  - `round` : Renvoie la valeur d'un nombre arrondi à l'entier le plus proche.
  - Et aussi : `abs`, `ceil`, `cos`, `sin`, `exp`, `sqrt`, `log`, `pow`, ...

Exemple :

```
1 console.log("Meilleure note arrondie par défaut : " +
    Math.floor(Math.max(2.7, 17.4, 9.7))); // affiche "17"
```

Un exemple d'objet natif : l'objet `String`

- Exemple de propriété : `length`
  - Longueur en caractères
- Quelques méthodes
  - `charAt` : Renvoie le caractère à la position spécifiée
  - `substring` : Renvoie les caractères d'une chaîne entre deux positions dans celle-ci
  - `toLowerCase` : Renvoie la valeur de la chaîne appelante convertie en minuscules
  - `toUpperCase` : Renvoie la valeur de la chaîne appelante convertie en majuscules

Exemple :

```
1 let nom = window.prompt ("Votre nom ?");
2 // Je tape "André"
3
4 console.log("Nom en majuscules : " + nom.toUpperCase() + ",
    première lettre : " + nom.charAt(0) + ", longueur : " +
    nom.length); // Affiche "Nom en majuscules : ANDRÉ,
    première lettre : A, longueur : 5"
```

Créer ses propres objets

- ES5 : pas de classes... mais des objets !
- Ajout dynamique de propriétés à tout moment

```
var film = {
  titre          : "Reservoir Dogs",
  equipe_technique : {
    montage    : "Sally Menke",
    costumes   : "Betsy Heimann"
  },
  sortie         : 1992,
  realisation : "Quentin Tarantino"
};
// Ajout dynamique d'une propriété
// NOTE: un peu bizarre d'un point de vue programmation objet !
film.distinction = "Festival d'Avignon";
// Accès aux propriétés
console.log("Le film " + film.titre + " a été réalisé par " +
    film.realisation + ", monté par " +
    film.equipe_technique.montage + " et a reçu un prix au " +
    film.distinction); // affiche "Le film Reservoir Dogs a été
    réalisé par Quentin Tarantino, monté par Sally Menke et a reçu
    un prix au Festival d'Avignon"
```

Méthodes en ES5

- Méthode : propriété d'un objet dont la valeur est une fonction

```
var velo = {
  marque    : "Giant",
  bequille  : true,
  couleur   : "bleu",
  vitesse   : 0,

  demarrer  : function()  { this.bequille = false;},
  accelerer : function(x) { if (!this.bequille)
    this.vitesse +=x ; },
  repeindre : function(c) { this.couleur = c;}
};
velo.demarrer();
console.log(velo.bequille) ; // affiche "false"
velo.accelerer(20);
console.log(velo.vitesse) ;  // affiche "20"
```

Méthodes en ES6

- Méthode : nouvelle notation dédiée

```
let velo = {
  marque    : "Giant",
  bequille  : true,
  couleur   : "bleu",
  vitesse   : 0,

  demarrer()   { this.bequille = false;},
  accelerer(x) { if (!this.bequille) this.vitesse +=x; },

  repeindre(c) { this.couleur = c;}
};
velo.demarrer();
console.log(velo.bequille); // affiche "false"
velo.accelerer(20);
console.log(velo.vitesse);  // affiche "20"
```

Constructeurs

- Constructeur JavaScript : fonction utilisant `this` pour initialiser le nouvel objet

```
let v = new Object(); // Équivalent à v = {}
v.couleur      = 'rouge';
v.nb_vitesses  = 7;
v.proprietaire = {prenom :'Alice', nom : 'Martin'};

// Mieux : constructeur
function Velo(c,n,p) {
  this.couleur       = c;
  this.nb_vitesses   = n;
  this.proprietaire  = p;
}
let v1 = new Velo('rouge', 7, {prenom :'Alice', nom :
    'Martin'});
let v2 = new Velo('bleu', 21, {prenom :'Pierre', nom :
    'Dupont'});
```

Création d'une classe

- Mot-clé `class`, et présence d'un `constructor`

```
class Velo {
  constructor(m, c) {
    this.marque    = m;
    this.couleur   = c;
    this.bequille  = true; // initialement à l'arrêt
    this.vitesse   = 0;
  }
  demarrer()   { this.bequille = false;}
  accelerer(x) { if(!this.bequille){this.vitesse +=x;}}
  repeindre(c) { this.couleur = c;}
  afficher()   { return "Vélo de marque " + this.marque + " et de
    couleur " + this.couleur;}
}
const velo = new Velo("Giant", "bleu"); // instance de la classe
console.log(velo.bequille); // affiche "true"
velo.demarrer();
console.log(velo.bequille); // affiche "false"
velo.accelerer(20);
console.log(velo.vitesse);  // affiche "20"
console.log(velo.afficher()); // affiche "Vélo de marque Giant et
    de couleur bleu"
```

Classes et héritage

- Mot-clé `extends`
- appel à la classe parente par `super`

```
1  class VeloElectrique extends Velo {
2    constructor(m, c, p) {
3      super(m, c);
4      this.puissance = p;
5    }
6    afficher() { return super.afficher() + " de puissance " +
       this.puissance; }
7  }
8
9  const ve = new VeloElectrique("Giant", "vert", 50);
10 ve.demarrer();
11 ve.accelerer(20);
12 console.log(ve.vitesse);  // affiche "20"
13 console.log(ve.afficher()); // affiche "Vélo de marque
       Giant et de couleur vert de puissance 50"
```

Classes et méthodes statiques

- Mot-clé `static`
- Une méthode statique ne peut pas être invoquée sur un objet, mais sur une classe
  - classique en programmation objet

```
1  class Velo {
2    constructor(m, c) {
3      this.marque    = m;
4      this.couleur   = c;
5      this.bequille  = true; // initialement à l'arrêt
6      this.vitesse   = 0;
7    }
8    afficher()   { return "Vélo de marque " + this.marque + " et de
       couleur " + this.couleur;}
9    static definition() { return "Un vélo est un véhicule à 2 roues
       qui va plus vite qu'une voiture à 4 roues";}
10 }
11 const v = new Velo("Btwin", "rose"); // instance de la classe
12 console.log(v.afficher()); // affiche "Vélo de marque Btwin et de
       couleur rose"
13 console.log(Velo.definition()); // appel sur LA CLASSE et non un
       objet
```

Outline

Les tableaux en JavaScript

- Tableau : séquence d'éléments accessibles grâce à un indice entier

```
1  // Création :
2  let reals = ['Wong', 'Sciamma', 'Tarantino', 'Ducournau']
3
4  // Accès
5  console.log(reals[1]); // affiche "Sciamma"
6
7  // Modification
8  reals[2] = 'Hamaguchi';
9  console.log(reals); // affiche "["Wong", "Sciamma",
       "Hamaguchi", "Ducournau"]"
10
11 // Propriétés
12 console.log(reals.length); // affiche "4"
13 reals.length = 3;
14 console.log(reals); // affiche "["Wong", "Sciamma",
       "Hamaguchi"]"
15 reals.length = 4;
16 console.log(reals); // affiche "["Wong", "Sciamma",
       "Hamaguchi", <1 empty slot>]"
```

Absence de typage des tableaux

- On peut mettre toutes sortes de choses dans un tableau

```
1  let reals = ['Wong', 'Sciamma', 'Tarantino', 'Ducournau']
2
3  // Indice: 0   1   2   3     4     5          6
4  let tab = [1, 2, 3, {}, 'a', (x)=>x+2, reals];
5
6  console.log(tab[0]); // affiche "1"
7
8  tab[4] = false;
9  console.log(tab[4]); // affiche "false"
10
11 tab[3].nom = 'Lee';
12 console.log(tab[3]); // affiche "Object {nom: "Lee"}"
13
14 let v = tab[5](8);
15 console.log(v); // affiche "10"
16
17 let l = tab[6][1];
18 console.log(l) ; // affiche "Sciamma"
```

L'objet `Array`

- Déclaration et création

```
let nomDuTableau = new Array(taille); // taille =
    nombre d'éléments du tableau
```

Utilisation : Les éléments sont indicés de 0 jusqu'à `taille - 1`

```
let tab = new Array(10);
tab[3]  = "Hou Hsiao-hsien";
```

- Propriété `length` : nombre d'éléments du tableau
- Quelques méthodes
  - `concat` : `res=tab1.concat(tab2)` concatène `tab1` et `tab2` et met le résultat dans `res`
  - `join` : `chaine=tab.join(car)` regroupe en une seule chaîne tous les éléments de `tab` séparés par le caractère séparateur `car` (par défaut `car =','`)
  - `sort` : trie les éléments par ordre alphabétique
  - `reverse` : inverse l'ordre des éléments
  - `indexOf` : retourne le premier (plus petit) indice d'un élément égal à la valeur passée en paramètre à l'intérieur du tableau, ou -1 si aucun n'a été trouvé

Plus de méthodes

- `push` : ajoute un élément

```
let reals = ['Wong', 'Sciamma', 'Lee', 'Ducournau']
reals.push("Hou");
console.log(reals); // affiche ['Wong', 'Sciamma',
    'Lee', 'Ducournau', 'Hou']
```

- `pop` : retire le dernier élément

```
reals.pop();
console.log(reals); // affiche ['Wong', 'Sciamma',
    'Lee', 'Ducournau']
```

- `shift` : retire le premier élément

```
reals.shift();
console.log(reals); // affiche ['Sciamma', 'Lee',
    'Ducournau']
```

- `slice` : retourne un tableau avec les éléments entre deux indices

```
let qq_reals = reals.slice(1,3);
console.log(qq_reals); // affiche ['Lee', 'Ducournau']
```

Parcours de tableau : boucle `for`

- Syntaxe classique
  - Déjà vue dans de nombreux autres langages

```
let reals = ['Wong', 'Sciamma', 'Tarantino',
    'Ducournau']
let i;
for(i = 0; i < reals.length; i++) {
  console.log(reals[i]);
}
// Affiche:
// Wong
// Sciamma
// Tarantino
// Ducournau
```

... mais pas le plus habituel (ni le plus simple) en JavaScript

Parcours de tableau : syntaxe `for ... of`

```
let reals = ['Wong', 'Sciamma', 'Tarantino', 'Ducournau']
for (let real of reals) {
  console.log(real);
}
// Affiche:
// Wong
// Sciamma
// Tarantino
// Ducournau
```

Parcours de tableau : syntaxe `forEach`

- Meilleure syntaxe : `forEach`
  - reçoit un *callback* et l'applique à chaque élément du tableau
  - le *callback* reçoit en paramètre un élément du tableau

```
1 let reals = ['Wong', 'Sciamma', 'Tarantino', 'Ducournau']
2 reals.forEach( (real) => { console.log(real); });
3
4 let nombres = [1, 2, 4, 5, 7, 10, 13, 2004]
5 let somme = 0;
6 nombres.forEach( (nb) => { somme += nb; });
7 console.log(somme); // affiche 2046
```

Outline

Les templates de chaînes dans une page Web (1/2)

```
1  <!DOCTYPE html>
2  <html lang="fr">
3    <head>
4      <meta charset="utf-8" >
5      <title>Exemple : HTML + JavaScript</title>
6    </head>
7    <body>
8    <!-- Mauvaise pratique! -->
9      <script>
10       let nom       = 'Sciamma';
11       let prenom    = 'Céline';
12       let naissance = 1978;
13       let annee_courante = new Date().getFullYear();
14       let phrase = `La réalisatrice ${prenom + ' '+ nom} a cette
     année ${annee_courante - naissance} ans`;
15       console.log(phrase); // affichage dans le terminal
16     </script>
17   </body>
18 </html>
```

Exemple téléchargeable depuis Moodle

Les templates de chaînes dans une page Web (2/2)

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="utf-8" >
    <title>Exemple : HTML + JavaScript</title>
  </head>
  <body>
  <!-- Mauvaise pratique! -->
    <script>
      let nom        = 'Sciamma';
      let prenom     = 'Céline';
      let naissance = 1978;
      let annee_courante = new Date().getFullYear();
      let phrase = `La réalisatrice ${prenom + ' '+ nom} a cette
    année ${annee_courante - naissance} ans`;
      document.write(phrase); // affichage dans le corps de la
    page Web
    </script>
  </body>
</html>
```

Exemple téléchargeable depuis Moodle

Utilisation d'une fonction JavaScript dans une page HTML

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="utf-8" >
    <title>Exemple : HTML + JavaScript</title>
  <!-- Mauvaise pratique!  -->
    <script>
      function decubique(){
        return Math.ceil(6*Math.random());
      }
    </script>
  </head>
  <body>
    <h1>Tirage d'un dé à 6 faces</h1>
    <script>
      document.write('<p>Tirage = ' + decubique() + '</p>');
    </script>
  </body>
</html>
```

Exemple téléchargeable depuis Moodle

Chargement de programmes JavaScript en HTML

Inclusion du code JavaScript dans la page HTML :

- balise `<script>`
- à placer dans la partie `<head>`
- ou (très fortement recommandé) : en fin de `<body>`
  Avantages
  - permet de ne pas ralentir l'affichage de la page avec le téléchargement du JavaScript
  - permet de s'assurer que la page est entièrement chargée lorsque le code JavaScript va accéder à des éléments de la page

```
1  <!DOCTYPE html>
2  <html lang="fr">
3    <head>
4    ...
5    </head>
6    <body>
7    ....
8      <script src="monscript.js"></script>
9    </body>
10 </html>
```

Import d'une fonction JavaScript dans une page HTML

```html
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="utf-8" >
    <title>Exemple : HTML + JavaScript</title>
    <script src="decubique.js"></script>
  </head>
  <body>
    <h1>Tirage d'un dé à 6 faces</h1>
    <!-- Pratique bof bof ! -->
    <script>document.write('<p>Tirage = ' + decubique() + '</p>');</script>
  </body>
</html>
```

Fichier `decubique.js` :

```javascript
'use strict'
// Fonction retournant un tirage aléatoire entre 1 et 6
function decubique(){
  return Math.ceil(6*Math.random());
}
```

Démonstration

Exemple téléchargeable depuis Moodle

Bonne pratique

✔ Séparation du JavaScript

Le code JavaScript devrait être (quasi) intégralement séparé du HTML.
Dans l'idéal, la seule modification du code HTML est l'insertion du script grâce à la balise `<script>`.

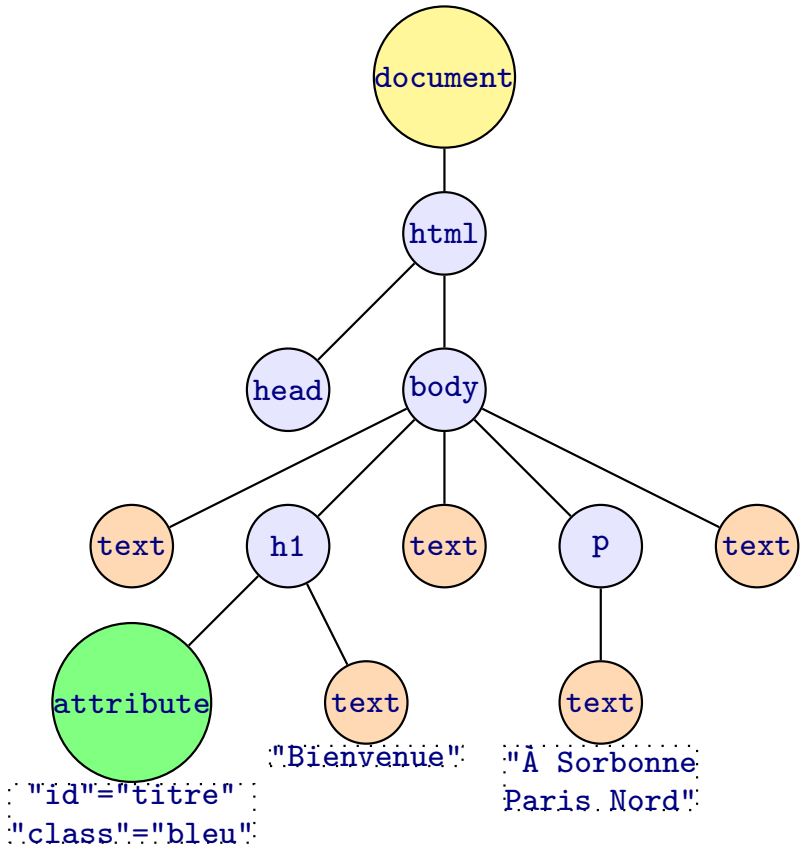
Outline

JavaScript et le DOM

- Tous les navigateurs disposent d'un interprète JavaScript embarqué
- Interaction de JavaScript avec le document : DOM
  - Rappel : DOM = *Document Object Model*
- Réaction suite à des actions de l'internaute : évènements

DOM : rappel

```
1 <!DOCTYPE html>
2 <html lang="fr">
3   <head> ... </head>
4   <body>
5     <h1 id="titre"
      class="bleu">Bienvenue</h1>
6     <p>À Sorbonne Paris Nord</p>
7   </body>
8 </html>
```



Manipuler le DOM avec JavaScript

- JavaScript fournit une interface pour manipuler le DOM :
  - sélectionner et accéder à des éléments de l'arbre DOM, naviguer dans le DOM
  - créer, insérer, supprimer, modifier des éléments dans le DOM
  - écouter des évènements produits par des éléments dans le DOM

Outline

Sélectionner des éléments du DOM

- `document.getElementById("id")`
  Retourne le nœud (unique) dont l'identifiant est `id`

- `document.getElementsByTagName("mabalise")`
  Retourne les nœuds dont la balise est `mabalise`

- `document.getElementsByClassName("maclasse")`
  Retourne les nœuds dont la classe est `maclasse`

Sélectionner des éléments du DOM : exemple

```html
<html lang="fr">
  <head> ... </head>
  <body>
    <section class="main">
      <p id="p1">paragraphe 1</p>
      <p id="p2">paragraphe 2</p>
      <p id="p3">paragraphe 3</p>
    </section>
  </body></html>
```

```javascript
let premier = document.getElementById("p1");
console.log(premier.nodeName, premier.id); // affiche "P p1"

let paragraphes = document.getElementsByTagName("p");
console.log(paragraphes[1].nodeName, paragraphes[1].id); //
    affiche "P p2"

let mains = document.getElementsByClassName("main");
console.log(mains[0].nodeName, mains[0].className); //
    affiche "SECTION main"
```

Sélectionner des éléments du DOM : requêtes

- `element.querySelector(selecteur)`
  Retourne parmi les descendants du nœud sur lequel on l'invoque le 1er élément qui correspond au `selecteur`
  - `element` : peut être `document` ou un nœud quelconque
  - `selecteur` : sélecteur CSS

- `element.querySelectorAll(selecteur)`
  Retourne parmi les descendants du nœud sur lequel on l'invoque la liste contenant tous les éléments qui correspondent au `selecteur`

Sélectionner des éléments du DOM par requête : exemple

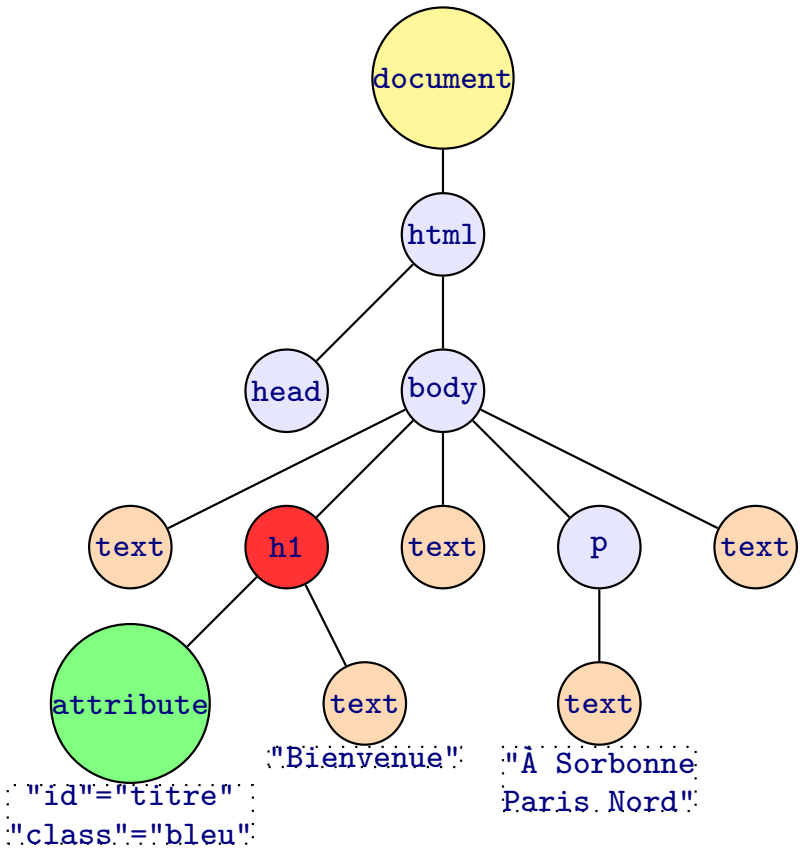
```html
<html lang="fr">
  <head> ... </head>
  <body>
    <section class="main">
      <p id="p1">paragraphe 1</p>
      <p id="p2">paragraphe 2</p>
      <p id="p3">paragraphe 3</p>
    </section>
  </body></html>
```

```javascript
let premier = document.querySelector("#p1");
console.log(premier.nodeName, premier.id); // affiche "P p1"

let paragraphes = document.querySelectorAll("p");
console.log(paragraphes[1].nodeName, paragraphes[1].id); //
    affiche "P p2"

let mains = document.querySelectorAll(".main");
console.log(mains[0].nodeName, mains[0].className); //
    affiche "SECTION main"
```

Outline

Se déplacer dans le DOM : nom du nœud courant

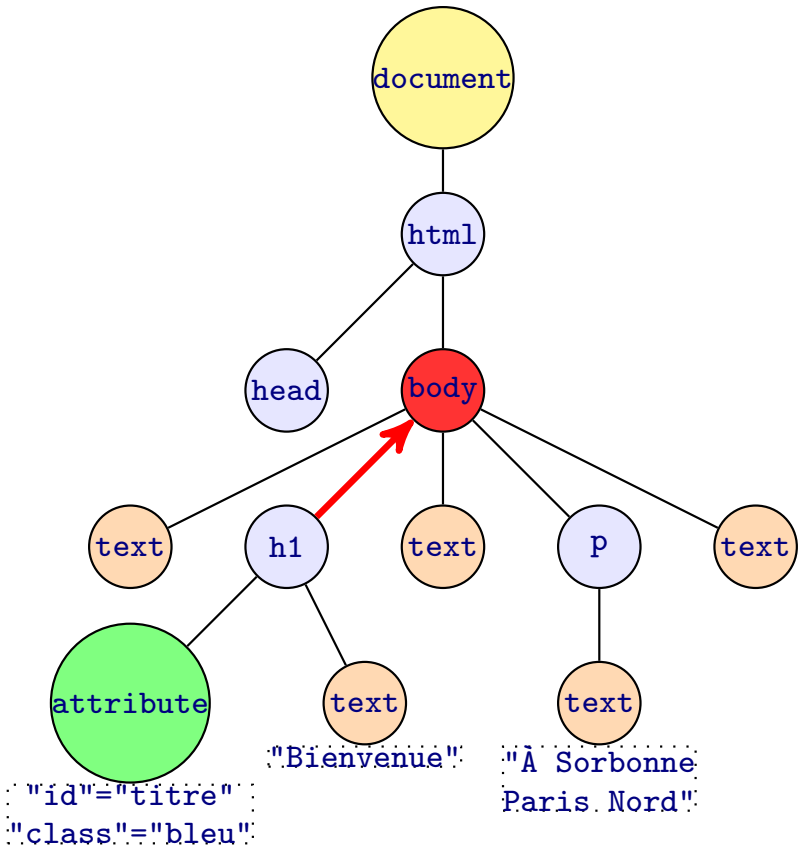
- Nom du nœud courant : `nodeName`




```
let noeud =
    document.querySelector("#titre")
console.log(noeud.nodeName); //
    affiche "H1"
```

Se déplacer dans le DOM : nœud parent

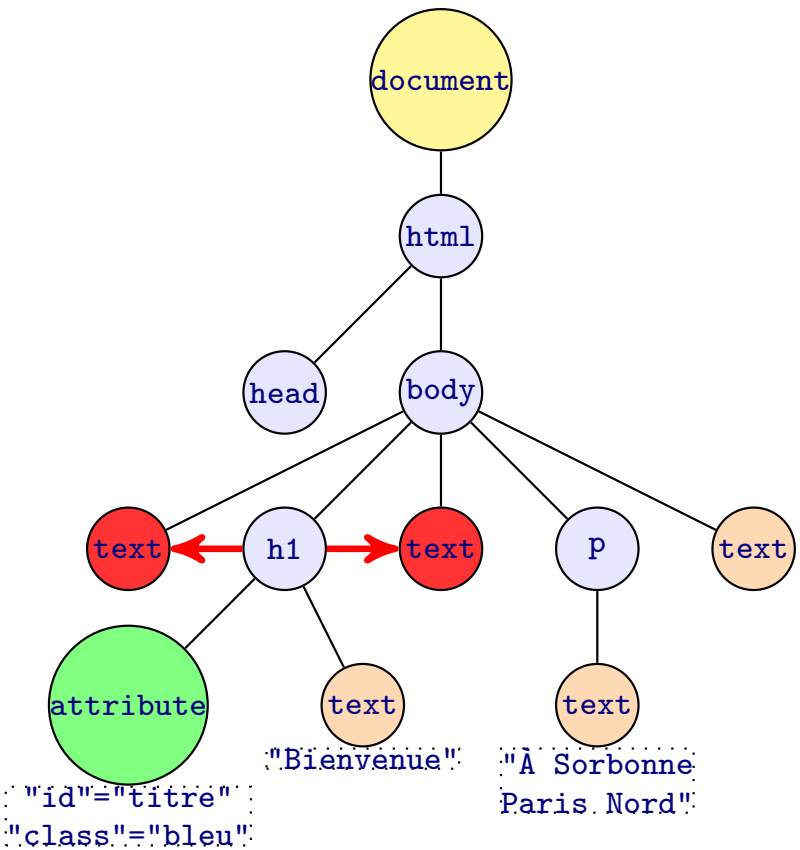
- Accès au nœud parent : `parentNode`




```
let noeud =
    document.querySelector("#titre")
let parent = noeud.parentNode;
console.log(parent.nodeName); //
    affiche "BODY"
```

Se déplacer dans le DOM : nœuds frères

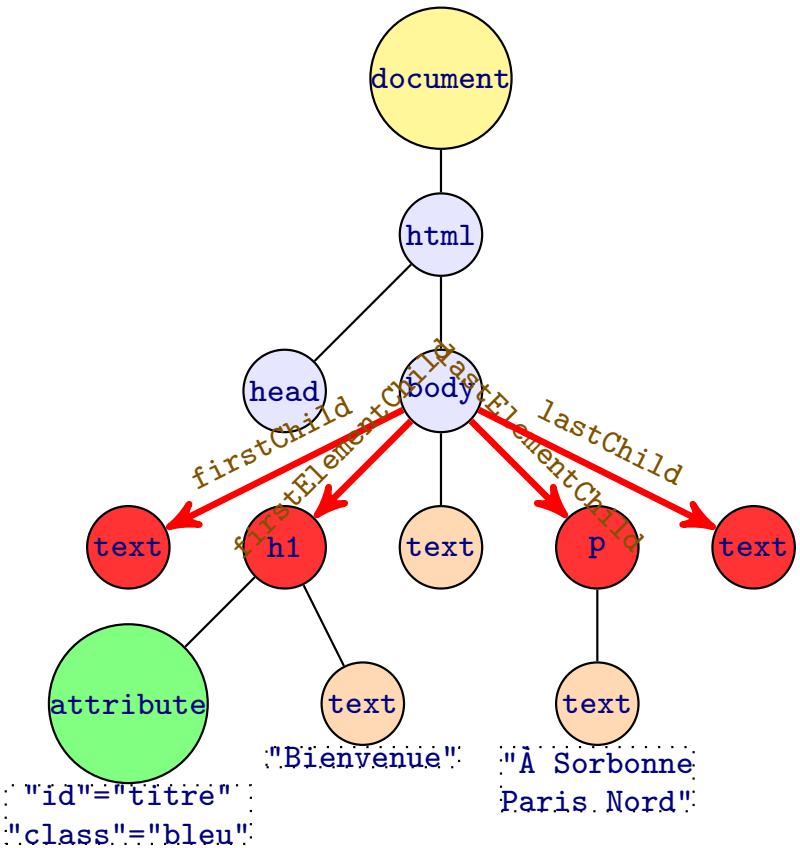
- Accès au nœud frère antérieur : `previousSibling`
- Accès au nœud frère postérieur : `nextSibling`
- *siblings* : nœuds enfants du parent du nœud courant




```
let noeud =
    document.querySelector("#titre")
let frere_precedent =
    noeud.previousSibling;
let frere_suivant   =
    noeud.nextSibling;
console.log(frere_precedent.nodeNam
    // affiche "#text"
console.log(frere_suivant.nodeName)
      // affiche "#text"
```

Se déplacer dans le DOM : nœuds enfants (1/2)

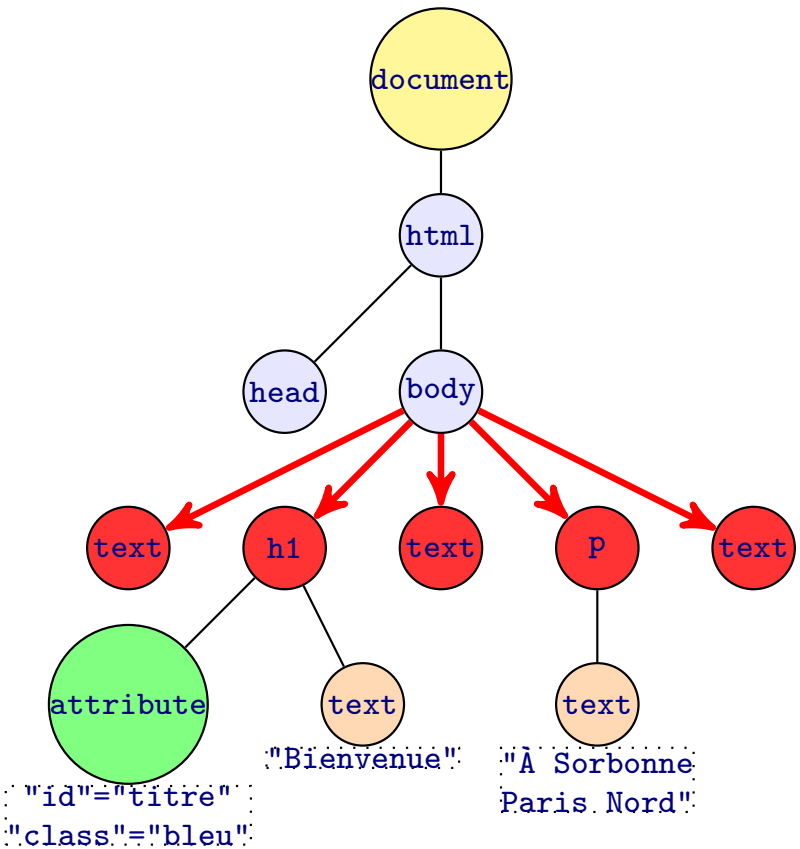
- Accès au premier enfant : `firstChild`
- Accès au premier enfant qui est un élément : `firstElementChild`
- Accès au dernier enfant : `lastChild`
- Accès au dernier enfant : `lastElementChild`




```
1 let noeud =
      document.querySelector("body");
2 let fs = noeud.firstChild ;
3 console.log(fs.nodeName); //
      affiche "#text"
4 console.log(noeud.firstElementChild
5   .nodeName); // affiche "H1"
6 console.log(noeud.firstElementChild
7   .firstChild.textContent); //
      affiche "Bienvenue"
```

Se déplacer dans le DOM : nœuds enfants (2/2)

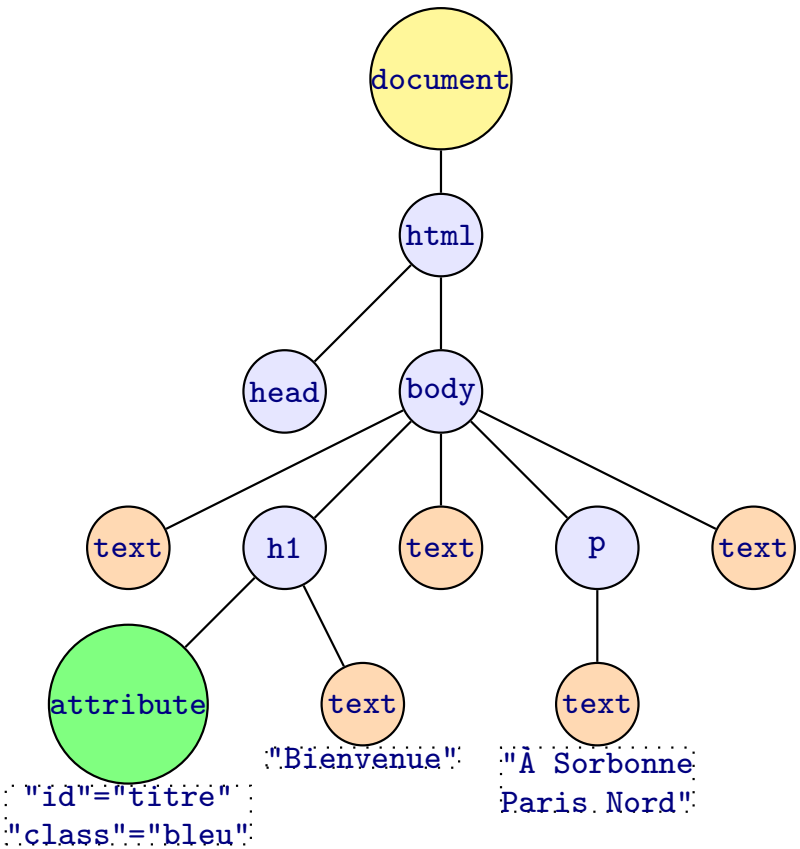
- Liste de tous les nœuds enfants : `childNodes`
- Nombre de nœuds enfants qui sont des éléments : `childElementCount`




```
let noeud =
    document.querySelector("body");
let enfants = noeud.childNodes;
console.log(enfants[3].nodeName);
    // affiche "P"
console.log(enfants.length); //
    affiche 5
console.log(noeud.childElementCount
    // affiche 2
```

Valeur texte d'un nœud

La valeur textuelle d'un nœud de type « texte » peut être récupérée par l'attribut `textContent`




```
1 let noeud =
      document.querySelector("body");
2
3 console.log(noeud.firstElementChild
4   .firstChild.textContent); //
      affiche "Bienvenue"
```

Outline

Modifier les attributs CSS du document

- Modification de l'apparence d'un document : en ajoutant ou en retirant des classes CSS sur un élément
  - `elt.classList` : liste des classes portées par `elt` (propriété non modifiable directement)
  - Ajout : `elt.classList.add("nouvelle")` : ajoute de la classe `"nouvelle"` à l'élément `elt`
  - Suppression : `elt.classList.remove("suppr")` : supprime la classe `"suppr"` de l'élément `elt`
  - Inversion : `elt.classList.toggle("classe")` : ajoute/supprime la classe `"classe"` de l'élément `elt`
  - Test : `elt.classList.contains("classe")` : rend `vrai` si l'élément `elt` porte la classe `"classe"`

Modifier le document

- Méthodologie
  1. créer des nœuds, puis
  2. les insérer dans l'arbre, puis
  3. supprimer des nœuds de l'arbre (le cas échéant)

Création de nœuds

- `createElement` : création d'un nœud de type élément

```
let b = document.createElement("b"); // création d'une balise "<b>" (texte gras)
```

- `createTextNode` : création d'un nœud de type texte

```
let t = document.createTextNode('Bienvenue');
```

- `setAttribute` : affectation à un nœud d'un attribut avec sa valeur

```
b.setAttribute("id", "tresgras");
```

Insertion de nœuds dans le DOM

- `appendChild` : insertion d'un nœud comme dernier enfant

```
parent.appendChild(nouveau); // insère le noeud nouveau
    comme dernier enfant de "parent"
```

- `insertBefore` : insertion d'un nœud avant le nœud passé en paramètre

```
parent.insertBefore(nouveau, noeud_suivant); // insère
    le noeud nouveau avant noeud_suivant dans les
    enfants de "parent"
```

- `replaceChild` : remplace un nœud

```
parent.replaceChild(nouveau, ancien); // remplace
    ancien par nouveau dans les enfants de "parent"
```

- `removeChild` : supprime un nœud des enfants

```
parent.removeChild(noeud); // supprime noeud de la
    liste des enfants de "parent"
```

Création et insertion de nœuds : exemple

```
window.addEventListener("load", () => {
    document.addEventListener("keypress", (e) => {
      console.log("key: ", e.key, "keycode: ", e.keyCode);
      if(e.key === 'm'){
        const body = document.querySelector("body");
        const nouveaup = document.createElement("p");
        nouveaup.setAttribute("id", "p4");
        const t = document.createTextNode("Et à Villetaneuse");
        nouveaup.appendChild(t);
        body.appendChild(nouveaup);
        // insérer 1 ligne après le h1
        const p = document.querySelector("body>p");
        body.insertBefore(document.createElement("hr"), p);
      } // end if
    }); // end listener
});
```

 Démonstration

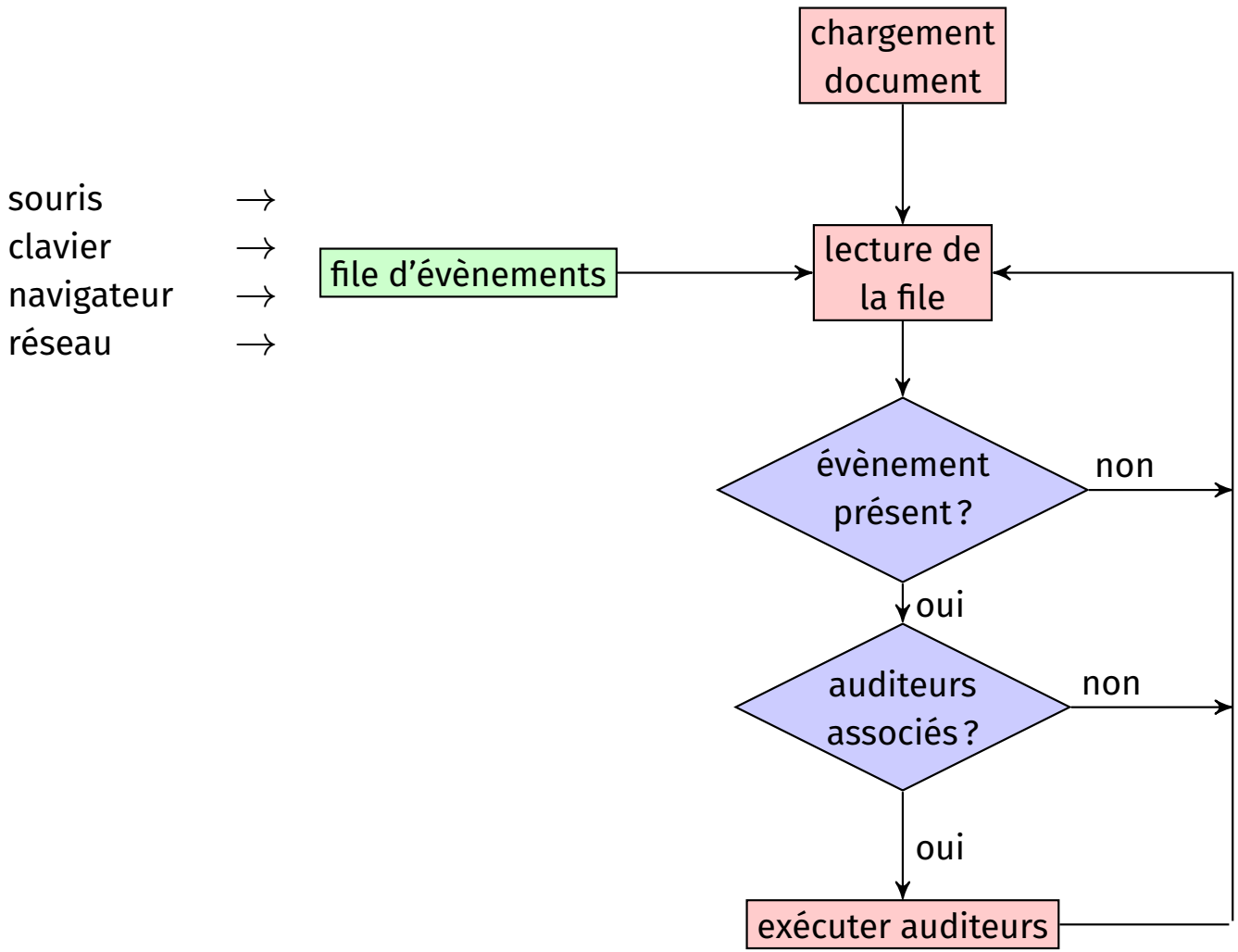
Outline

Outline

Les évènements en JavaScript

- Évènement : action de l'internaute sur l'interface (ou du réseau, ou ...)
- Pour réagir à ces actions : on lie une fonction à un évènement
  - *handler* (gestionnaire d'évènements)

La boucle d'évènements JavaScript



⚠ Un seul auditeur est executé à la fois ! (*monothread*)

Les évènements et le DOM

- Les objets du DOM génèrent des évènements correspondant souvent à des actions de l'internaute sur l'interface (ils sont observables)
- Une application JavaScript dans le navigateur est une application conduite par des évènements (*event driven*)
- La boucle d'évènements est réalisée par l'interprète JavaScript

Quelques évènements du DOM en JavaScript

Évènement	Type de nœud	Action détectée
`load`	`document`, `frameset`	Chargement complet du DOM
`click`	`element`	Clic de souris
`mousedown`	`element`	Bouton de souris enfoncé
`mouseup`	`element`	Bouton de souris relâché
`mouseover`	`element`	Passage du pointeur au-dessus
`mouseout`	`element`	Sortie du pointeur
`keypress`	`element`	Touche de clavier pressée
`blur`	`element`	Perte du focus (utile dans un formulaire)
`focus`	`element`	Acquisition du focus
`change`	`element`	Modification de la valeur
`resize`	`document`, `element`	Changement de taille
`scroll`	`document`, `element`	Utilisation de la molette

Outline

Enregistrer le gestionnaire d'évènement sur un nœud

- Trois méthodes principales pour enregistrer la fonction gestionnaire (*handler*) sur un nœud pour un évènement
  1. Via la propriété `on<event>` dans le HTML (⚠ mauvaise pratique !)
  2. Via la propriété `on<event>` des nœuds
  3. Via un auditeur dédié (souvent la meilleure pratique)

- Quand un évènement `e` survient : `e.target` contient la cible de l'évènement (propriété de `Event`)
  - Documentation : `https://developer.mozilla.org/fr/docs/Web/API/Event`

Méthode 1 (nulle) : propriété on<event> du HTML

- Associer un appel JavaScript à un évènement directement dans le HTML

```
1  <!DOCTYPE html>
2  <html lang="fr">
3    <head>
4      <meta charset="utf-8">
5      <title>Gestionnaire d'évènement méthode 2</title>
6    </head>
7    <body>
8      <p onclick="alert('Sorbonne !');">Sorbonne Paris
       Nord</p>
9      <p onmouseover="console.log('Galilée');">Institut
       Galilée</p>
10   </body>
11 </html>
```

Méthode 1 (nulle) : propriété on<event> du HTML

- Associer un appel JavaScript à un évènement directement dans le HTML

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Gestionnaire d'évènement méthode 2</title>
  </head>
  <body>
    <p onclick="alert('Sorbonne !');">Sorbonne Paris
    Nord</p>
    <p onmouseover="console.log('Galilée');">Institut
    Galilée</p>
  </body>
</html>
```

⚠ Mauvaise pratique

Cette pratique est une mauvaise pratique (dépréciée), et ne devrait pas être utilisée

Méthode 2 (moche) : propriété `on<event>` des nœuds

- Donner une valeur à la propriété `on<evt>` d'un nœud du DOM
  - `load` ⇒ `onload`
  - `click` ⇒ `onclick`
  - etc.

- Lorsque l'évènement se produit, cette valeur est utilisée comme gestionnaire d'évènement

```
1 node.onclick = () => { ... };
2 node.onmouseover = animateButton;
3 window.onload = () => { ... };
```

⚠ Un seul gestionnaire

Cette méthode ne permet pas d'enregistrer plusieurs gestionnaires (*handlers*) sur le même évènement

Méthode 2 (moche) : exemple 1

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Gestionnaire d'évènement méthode 2</title>
    <script>
        document.onclick = () => {console.log("Clic dans la page détecté !")};
        document.onkeypress = affToucheClavier;

        function affToucheClavier(e){
          console.log("Touche pressée : ", e.key, "Code : ", e.keyCode);
        }
    </script>
  </head>
  <body>
    <h1>Évènements clavier et clic</h1>
  </body>
</html>
```

Méthode 2 (moche) : exemple 2

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Gestionnaire d'évènement méthode 2</title>
  </head>
  <body>
    <p id="SPN">Sorbonne Paris Nord</p>
    <p id="IG">Institut Galilée</p>
    <script>
      let spn = document.getElementById("SPN");
      let ig  = document.querySelector("#IG");

      spn.onclick     = (event) => {alert("Sorbonne !");};
      ig.onmouseover = (event) => {alert ("Galilée !");};
   </script>
  </body>
</html>
```

🖥️ Démonstration

Méthode 3 (bien) : auditeur

- Enregistrement d'un auditeur (ou *listener* ou écouteur) sur un nœud pour un évènement

```
node.addEventListener("evt", handler);
// "evt"    : l'évènement concerné
// handler : une fonction recevant un objet event en
    paramètre
```

- Il est possible d'enregistrer plusieurs auditeurs sur le même nœud pour un même évènement

```
lien.addEventListener("click", (event) => {alert("clic
    !");});
lien.addEventListener("click", (event) =>
    {console.log("clic !");});
```

Méthode 3 : exemple de déclaration

```html
<html lang="fr">
  <head> ... </head>
  <body>
    <p id="SPN">Sorbonne Paris Nord</p>
    <p id="IG">Institut Galilée</p>
    <script src="handler-methode3.js"></script>
  </body>
</html>
```

```javascript
let spn = document.getElementById("SPN");
let ig  = document.querySelector("#IG");

spn.addEventListener("click", (event) => {alert("Sorbonne !");});
spn.addEventListener("click", (event) =>
    {console.log("Sorbonne !");});
ig.addEventListener("mouseover", (event) =>
    {console.log("Galilée !");});
```

🖥️ Démonstration

Méthode 3 : chargement du DOM

⚠ Chargement du DOM

Il est important de s'assurer que le DOM est entièrement chargé avant de sélectionner des nœuds et de déclarer des auditeurs : insérer les déclarations d'auditeurs dans… un auditeur sur l'évènement `"load"`

```
1 'use strict'
2 window.addEventListener("load", () => {
3   let spn = document.getElementById("SPN");
4   let ig  = document.querySelector("#IG");
5
6   spn.addEventListener("click", (event) => {alert("Sorbonne
      !");});
7   spn.addEventListener("click", (event) =>
      {console.log("Sorbonne !");});
8   ig.addEventListener("mouseover", (event) =>
      {console.log("Galilée !");});
9 });
```

Propriétés et méthodes des évènements

- Un auditeur reçoit comme seul argument un objet `event`
- Selon le type de l'évènement, on a accès à différentes propriétés/méthodes :
  - pour accéder à la cible (nœud du DOM ayant reçu l'évènement) :
    - `event.target` : dans tous les cas
    - `this` : dans les auditeurs définis avec `function()`
  - pour annuler l'évènement : `event.preventDefault()`

Auditeur : exemple plus élaboré

```
'use strict'
window.addEventListener("load", () => {
  document.querySelector("#IG").addEventListener("click", (e) =>
    {e.target.classList.toggle("vert");});

  document.querySelector("h1").addEventListener("mouseover", (e)
    => {console.log("Souris passée sur le titre");
    document.querySelectorAll(".discipline").forEach( (d) =>
    {d.classList.toggle("bleu");}); });

  document.addEventListener("mousedown", (e) => {
    console.log("Clic @ x:", e.clientX, "y:", e.clientY); });
  document.addEventListener("keypress", (e) => {
    console.log("key: ", e.key, "keycode: ", e.keyCode);
    switch (e.key) {
      case 'b': document.body.style.backgroundColor = 'white';
    break
      case 'o': document.body.style.backgroundColor = 'orange';
    break
    }
  });
});
```

 Démonstration

Bonne pratique : structure d'un code JavaScript

```
1  'use strict';
2
3  /*************************************************************/
4  // Constantes
5  /*************************************************************/
6  const CHAINE_CLIC = 'clic !';
7
8  /*************************************************************/
9  // Variables
10 /*************************************************************/
11 let DOM_contenu;
12 let DOM_titre;
13
14 /*************************************************************/
15 // Fonctions
16 /*************************************************************/
17 function auditeur(){
18   const par   = document.createElement("p");
19   const texte = document.createTextNode(CHAINE_CLIC);
20   par.appendChild(texte);
21   DOM_contenu.appendChild(par);
22 }
23
24 /*************************************************************/
25 // Appel principal
26 /*************************************************************/
27 // On définit tout dans un event listener appelé lorsque la page est chargée
28 window.addEventListener('load', function() {
29   DOM_contenu = document.querySelector('main');
30   DOM_titre   = document.querySelector('header h1');
31   DOM_titre.addEventListener('click', auditeur);
32 });
33 // ...QUE FAIT CE SCRIPT ?
```

Outline

Dessiner dans une page HTML

- Utilisation de l'élément HTML `canvas`
  - se comporte comme une image

```
<canvas id="canevas" width="300" height="300">
  Oups, votre navigateur ne prend pas en charge canvas
</canvas>
```

- Sur un objet associé à `canvas` :
  - Méthode `getContext` donne accès au contexte graphique du canevas et permet ensuite de dessiner
  - Zone de dessin : origine $(0, 0)$ en haut à gauche du canevas

Canevas : exemple basique (HTML)

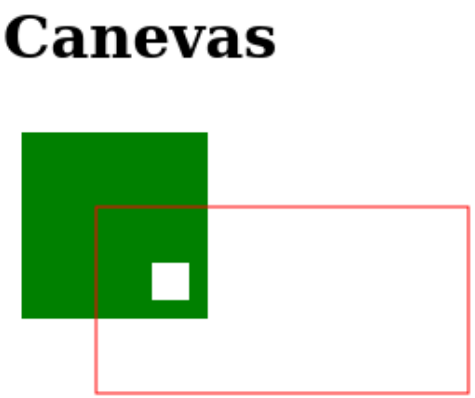
```html
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Exemple de canevas basique</title>
  </head>
  <body>
    <h1>Canevas</h1>
      <canvas id="canevas" width="300" height="300">
        Désolé, votre navigateur ne prend pas en charge les canevas.
      </canvas>
    <script src="canevas-basique.js"></script>
  </body>
</html>
```

Canevas : exemple basique (JavaScript)

```
1 const canvas   = document.getElementById('canevas');
2 const contexte = canvas.getContext('2d');
3 contexte.fillStyle = 'green';
4 contexte.fillRect(10, 10, 100, 100);
5 contexte.strokeStyle = 'red';
6 contexte.strokeRect(50, 50, 200, 100);
7 contexte.clearRect(80, 80, 20, 20);
```

Canevas : exemple basique (JavaScript)

```
1 const canvas   = document.getElementById('canevas');
2 const contexte = canvas.getContext('2d');
3 contexte.fillStyle = 'green';
4 contexte.fillRect(10, 10, 100, 100);
5 contexte.strokeStyle = 'red';
6 contexte.strokeRect(50, 50, 200, 100);
7 contexte.clearRect(80, 80, 20, 20);
```



Démonstration

Exemple téléchargeable depuis Moodle

Formes géométriques : rectangles

- `fillRect` : Dessine un rectangle rempli

```
fillRect(x, y, largeur, hauteur)
```

- `strokeRect` : Dessine un contour rectangulaire

```
strokeRect(x, y, largeur, hauteur)
```

- `clearRect` : Efface la zone rectangulaire spécifiée, la rendant complètement transparente

```
clearRect(x, y, largeur, hauteur)
```

Formes géométriques : trajets, lignes, arcs

Quatre étapes

1. création du trajet : `beginPath()`
2. séquence d'instructions de dessin pour dessiner sur le trajet (lignes, arcs…)
   - `moveTo(x,y)` : déplacement en `(x,y)` sans trace
   - `lineTo(x,y)` : trace une ligne depuis la position courante jusqu'au point `(x,y)`
   - `arc(x, y, rayon, angleInitial, angleFinal, antihoraire)` : dessine un arc de cercle
3. fermer le trajet : `closePath()`
4. le tracer (`fill()`) ou le remplir (`stroke()`) pour le faire apparaître

Formes géométriques : arcs et cercles

- Arc de cercle : `arc(x, y, rayon, angleInitial, angleFinal, antihoraire)`
  - dessine un arc de cercle centré sur la position `(x, y)`, de rayon `r`, commençant à `angleInitial` et finissant à `angleFinal` en allant dans le sens antihoraire ssi `antihoraire === true` (par défaut : horaire)

- Cercle : utiliser `arc` avec `angleInitial = 0, angleFinal = Math.PI * 2`

```
1 // Dessine un disque rouge plein de centre (75,75) et
      de rayon 50
2 beginPath();
3 contexte.arc(75, 75, 50, 0, Math.PI * 2, true);
4 closePath();
5 contexte.fillStyle = "red";
6 fill();
```

Formes géométriques : exemple (1/3)

```
function dessiner() {
  const canevas = document.getElementById('canevas');
  if (canevas.getContext) {
    const contexte = canevas.getContext('2d');

    // Triangle plein
    contexte.beginPath();
    contexte.moveTo(25, 25);
    contexte.lineTo(105, 25);
    contexte.lineTo(25, 105);
    contexte.closePath();
    contexte.fillStyle = 'blue';
    contexte.fill();
    // Triangle filaire
    contexte.beginPath();
    contexte.moveTo(125, 125);
    contexte.lineTo(125, 45);
    contexte.lineTo(45, 125);
    contexte.closePath();
    contexte.strokeStyle = 'red';
    contexte.stroke();
```

Formes géométriques : exemple (2/3)

```
    // Cercle
    contexte.beginPath();
    contexte.arc(150, 50, 20, 0, 2*Math.PI, 0);
    contexte.closePath();
    contexte.strokeStyle = 'green';
    contexte.stroke();

    // Demi-disque
    contexte.beginPath();
    contexte.arc(150, 100, 20, Math.PI/4, 5/4*Math.PI, 0);
    contexte.closePath();
    contexte.fillStyle = 'orange';
    contexte.fill();
  }
}

window.addEventListener('load', function() {
  dessiner();
});
```

Formes géométriques : exemple (3/3)

Rendu :

- Exemple téléchargeable depuis Moodle

Les couleurs

- Deux propriétés importantes du contexte de canevas
  - `fillStyle(couleur)` : définit la couleur utilisée lors du remplissage
  - `strokeStyle(couleur)` : définit la couleur utilisée lors du tracé
  - … où `couleur` est une chaîne représentant une couleur CSS, un objet gradient ou un objet motif

- Définition des couleurs : nom, valeur hexadécimale, RGB, RGBA

```
1 // Définitions possibles de la couleur orange
2 contexte.fillStyle = "orange";
3 contexte.fillStyle = "#FFA500";
4 contexte.fillStyle = "rgb(255,165,0)";
5 contexte.fillStyle = "rgba(255,165,0,1)";
```

- Documentation sur les couleurs :

  `https://developer.mozilla.org/fr/docs/Web/API/Event`

Les couleurs : exemple

```
1  function dessiner() {
2    const canevas =
       document.getElementById('canevas');
3    if (canevas.getContext) {
4      const contexte = canevas.getContext('2d');
5
6      for (let i = 0; i < 6; i++) {
7        for (let j = 0; j < 6; j++) {
8          contexte.fillStyle = 'rgb(' +
       Math.floor(255 - 42.5 * i) + ',' +
       Math.floor(255 - 42.5 * j) + ',0)';
9          contexte.fillRect(j * 25, i * 25, 25, 25);
10       }
11     }
12   }
13 }
14
15 window.addEventListener('load', function() {
16   dessiner();
17 });
```

Exemple téléchargeable depuis Moodle

Outline

Formulaires et JavaScript : l'objet `form`

```
<form id="form1" name="monForm"
    action="mailto:toto@uspn.fr" method="post"
    enctype="text/plain">
  Saisie du texte : <input type="text" name="saisie">
    <input type="submit" value="Valider le formulaire">
</form>
```

Plusieurs manières d'accéder à l'objet JavaScript `form` :

```
const f1 = document.getElementById("form1"); //recherche
    avec id (MÉTHODE RECOMMANDÉE)
const f2 = document.forms[0] ; // indice 0 du tableau forms
const f3 = document.forms['monForm'] ; // indice name
const f4 = document.monForm ; //accès direct avec name
```

Exemple téléchargeable depuis Moodle

Quelques propriétés de l'objet `form`

- `name` : nom du formulaire
- `action` : `mailto` ou adresse de la page vers laquelle est envoyé le formulaire lors de sa soumission (par appui sur le bouton HTML `submit` ou l'appel de la méthode JavaScript `submit()`)
- `method` : méthode de soumission : `post` ou `get`
- `enctype` : type d'encodage du contenu, par ex `'plain/text'`
- `elements` : tableau contenant tous les objets correspondants aux composants du formulaire

Quelques méthodes de l'objet `form`

- `reset()` : réinitialise le formulaire
- `submit()` : soumet le formulaire

Quelques propriétés des composants des formulaires

Certaines de ces propriétés sont communes à tous les composants ; d'autres sont spécifiques à certains composants

- `type` : type de l'élément (`"text"`, `"button"`, `"radio"`, `"checkbox"`, ...)
- `name` : nom de l'élément (rappel HTML : même `name` pour des boutons radio groupés)
- `value` : contenu du champ pour une zone de saisie
- `id` : identifiant de l'élément
- `disabled` : booléen à `true` si le champ de saisie est grisé et non accessible
- `checked` : booléen à `true` si une case à cocher ou un bouton radio est coché
- `selectedIndex` : indice de l'élément sélectionné dans une liste

Quelques méthodes des composants des formulaires

- `blur()` : fait perdre le focus à un composant
- `focus()` : met le focus sur un composant
- `change()` : déclenche un changement dans un composant
- `click()` : déclenche un clic de souris sur le composant
- `select()` : sélectionne le contenu d'un élément de formulaire

Exemple : boîtes à cocher (HTML)

```html
<!DOCTYPE html>
<html lang="fr">
  <head>...</head>
  <body>
    <h1>Quiz cinéma</h1>
      <p>Parmi ces noms, qui a déjà gagné la Palme d'or ?</p>
      <form id="formulaire">
      <p><input type="checkbox" id="besson"><label>Luc
    Besson</label></p>
      <p><input type="checkbox" id="ducournau"><label>Julia
    Ducournau</label></p>
      <p><input type="checkbox" id="imamura"><label>Shōhei
    Imamura</label></p>
      <p><input type="checkbox" id="tarantino"><label>Quentin
    Tarantino</label></p>
      <p><input type="checkbox" id="vansant"><label>Gus Van
    Sant</label></p>
      <p><button id="verifier">Vérifier</button></p>
      <p><button id="reinitialiser">Réinitialiser</button></p>
      </form>
    <script src="formulaire-checkbox.js"></script>
  </body>
</html>
```

Exemple : boîtes à cocher (JavaScript)

```
function avertissement(e) {
  if(e.target.checked){alert("N'importe quoi !");}}
function verification(e) {
  let c1 = document.querySelector("#besson");
  let c2 = document.querySelector("#ducournau");
  let c3 = document.querySelector("#imamura");
  let c4 = document.querySelector("#tarantino");
  let c5 = document.querySelector("#vansant");
  if (!c1.checked && c2.checked && c3.checked && c4.checked &&
    c5.checked){
    alert("Bonne réponse !");
  }else{
    alert("Faux, retournez au cinéma !");
  }
}
window.addEventListener('load', function() {
  document.querySelector("#verifier").addEventListener('click',
    verification);

    document.querySelector("#reinitialiser").addEventListener('click'
    (e) => {document.getElementById('formulaire').reset();});
  document.querySelector("#besson").addEventListener('click',
    avertissement);
}
```

Exemple : boîtes à cocher (rendu)

Démonstration

Exemple téléchargeable depuis Moodle

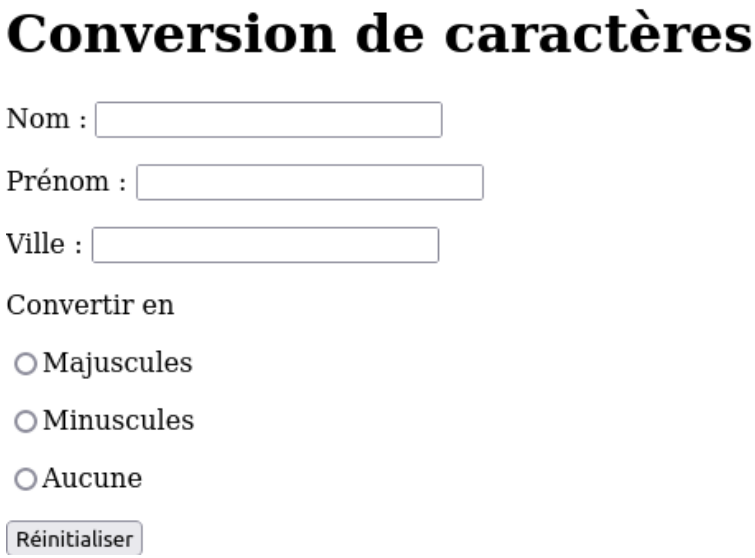
Exemple : boutons radio et zone de saisie (HTML)

```
<h1>Conversion de caractères</h1>

<form id="formulaire">
  <p>Nom :     <input type="text" name="nom"     size=20
    ></p>
  <p>Prénom : <input type="text" name="prenom" size=20
    ></p>
  <p>Ville :  <input type="text" id="ville"
    name="ville" size=20 ></p>
  <p>Convertir en</p>
  <p><input type="radio" name="conversion"
    id="maj">Majuscules</p>
  <p><input type="radio" name="conversion"
    id="min">Minuscules</p>
  <p><input type="radio" name="conversion"
    id="aucune">Aucune</p>
  <p><button id="reinitialiser">Réinitialiser</button></p>
</form>
<script src="formulaire-radio.js"></script>
```

Exemple : boutons radio et zone de saisie (JavaScript)

```
function convertirChamp(idChamp) {
  if (document.querySelector("#maj").checked)
    idChamp.value = idChamp.value.toUpperCase();
  else if (document.querySelector("#min").checked)
    idChamp.value = idChamp.value.toLowerCase();
}
function convertirTout(e) {
  let formulaire = document.getElementById('formulaire');
  convertirChamp(formulaire.nom);
  convertirChamp(formulaire.prenom);
  convertirChamp(formulaire.ville);
}
window.addEventListener('load', function() {
  document.querySelector("#maj").addEventListener('click',
    convertirTout);
  document.querySelector("#min").addEventListener('click',
    convertirTout);
  document.querySelector("#ville").addEventListener('change', (e)
    => convertirChamp(e.target));

    document.querySelector("#reinitialiser").addEventListener('click'
    (e) => {document.getElementById('formulaire').reset();});
});
```

Exemple : boutons radio et zone de saisie (rendu)



Démonstration

Exemple téléchargeable depuis Moodle

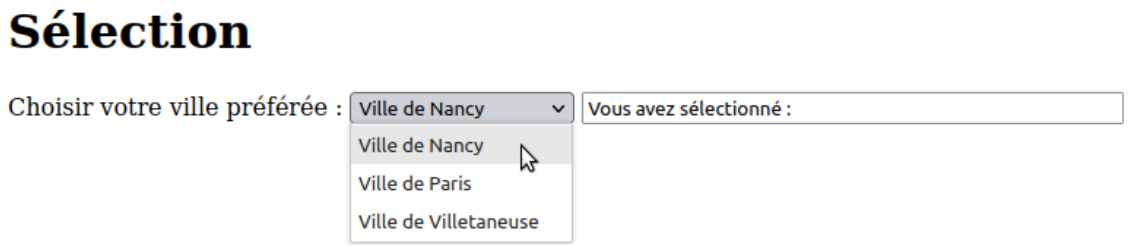
Exemple : liste de sélection (HTML)

```html
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Exemple de formulaire avec sélection</title>
  </head>
  <body>
    <h1>Sélection</h1>

      <form id="formulaire">
        <label for="liste">Choisir votre ville
    préférée :</label>
        <select id="liste" name="liste">
          <option value="nancy">Ville de Nancy</option>
          <option value="paris">Ville de Paris</option>
          <option value="villetaneuse">Ville de
    Villetaneuse</option>
        </select>
        <input type="text" readonly size=40 id="resultat"
    value="Vous avez sélectionné :">
      </form>
    <script src="formulaire-select.js"></script>
  </body>
</html>
```

Exemple : liste de sélection (JavaScript)

```
1  'use strict'
2
3  function modifier(e){
4    let el = e.target;
5    document.getElementById("resultat").value =
       document.getElementById("resultat").value + " " +
       el.options[el.selectedIndex].value;
6  }
7
8  window.addEventListener('load', function() {
9    document.querySelector("#liste").addEventListener('change',
       modifier);
10 });
```

Exemple : liste de sélection (rendu)



Démonstration

Exemple téléchargeable depuis Moodle

Sources et références

Sources et références

Références

- Frédéric Delobel : JavaScript et jQuery – La programmation Web par la pratique
- Robin Nixon : Développer un site web en PHP, MySQL JavaScript jQuery, CSS3 et HTML5 : Un guide étape par étape pour créer des sites Web dynamiques (O'Reilly)

Sources

- Ce cours est largement inspiré des supports de cours de Slim Ouni, Gérôme Canals, Isabelle Debled-Rennesson et Étienne André (Université de Lorraine)

Crédits

Source des images utilisées I

Titre : warning sign
Auteur : ?
Source : https://commons.wikimedia.org/wiki/File:Warning_sign.png
Licence : PUBLIC DOMAIN

Titre : Check Mark CSS Green
Auteur : Pigeon43
Source : https://commons.wikimedia.org/wiki/File:Check_Mark_CSS_Green.svg
Licence : CC BY

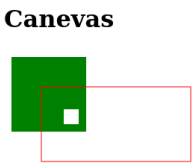
Titre : small-n-flat
Auteur : paomedia
Source : https://commons.wikimedia.org/wiki/File:Device-computer-icon.png
Licence : PUBLIC DOMAIN

Titre : Nuvola filesystems folder download
Auteur : David Vignoni
Source : https://commons.wikimedia.org/wiki/File:Nuvola_filesystems_folder_download.png
Licence : LGPL Free Software Free as in Freedom

Titre : prompt
Auteur : Étienne André

Source des images utilisées II

Source :
Licence : 



Titre : canevas basique
Auteur : Étienne André
Source :
Licence : 

Titre : canevas et formes
Auteur : Étienne André
Source :
Licence : 

Titre : canevas et couleurs
Auteur : Étienne André
Source :
Licence : 

Titre : Formulaire
Auteur : Étienne André
Source :
Licence : 

Source des images utilisées III

Titre : Formulaire
Auteur : Étienne André
Source :
Licence : 

Titre : Formulaire
Auteur : Étienne André
Source :
Licence : 

Licence de ce document

Auteur : Étienne André

(Source LaTeX disponible aux enseignant·e·s sur demande)

UNIVERSITÉ SORBONNE PARIS NORD

Institut GALILÉE
Université Sorbonne Paris Nord

Version : 29 avril 2024