Javascript

Objets, prototypes et nouveautés

CogniTIC

Building Together Better IT Solutions

Plan du cours

- Objets
- Prototypes
- Onerror
- Nouveautés : ECMAScript 6 vs ECMAScript5

OBJETS

Objets

- **Déclarer une classe**
- Créer un objet
- Accéder aux objets

Déclarer une classe (1)

```
function Utilisateur(pre, nomUt, mdp)
{
    // propriétés
    this.prenom = pre;
    this.nomUtilisateur = nomUt;
    this.motdepasse = mdp;
    // méthode qui sera créée pour chaque objet créé
    this.afficherUtilisateur = function()
    {
        document.write("<ul><li>Prénom : " + this.prenom + "</li>");
        document.write("<li>Nom d'utilisateur : " + this.nomUtilisateur + "</li>");
        document.write("<li>Mot de passe : " + this.motdepasse + "</li></ul>");
    }

}
```

Déclarer une classe (2)

```
function Utilisateur2(pre, nomUt, mdp)
{
    this.prenom = pre;
    this.nomUtilisateur = nomUt;
    this.motdepasse = mdp;
    // ajouter une méthode par après (par exemple)
    this.afficherUtilisateur2 = afficherUtilisateur2;
}
function afficherUtilisateur2()
{
    document.write("<ul><li>Prénom : " + this.prenom + "</li>");
    document.write("<li>Nom d'utilisateur : " + this.nomUtilisateur + "</li>");
    document.write("<li>Mot de passe : " + this.motdepasse + "</li></ul>");
}
```

Objets

- Déclarer une classe
- **Créer un objet**
- Accéder aux objets

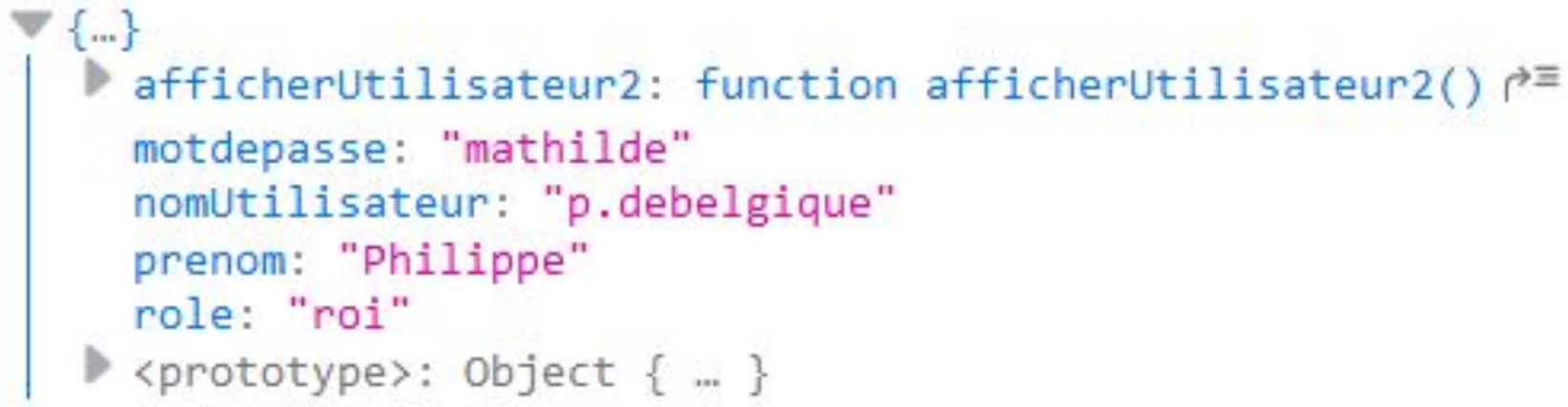
Créer un objet

```
// première manière
var utilisateur1 = new Utilisateur("Jules",
"j.cesar", "rome");
// deuxième manière
var utilisateur2 = new Utilisateur2();
   utilisateur2.prenom = "Philippe";
   utilisateur2.nomUtilisateur = "p.debelgique";
   utilisateur2.motdepasse = "mathilde";
```

Que contiennent-ils ?

Comment ajouter une propriété ?

- utilisateur2.role = "roi";



Objets

- Déclarer une classe
- Créer un objet
- **Accéder aux objets**

Accéder aux objets

```
// accéder aux propriétés
document.write("Rôle de ut2 : " + utilisateur2.role + "<br>");
// Role de ut2 : roi
document.write("Rôle de ut1 : " + utilisateur1.role + "<br>");
// Role de ut1 : undefined

// utiliser une méthode d'un objet
utilisateur1.afficherUtilisateur();
/*
• Prénom : Jules
• Nom d'utilisateur : j.cesar
• Mot de passe : rome
*/
```

Accéder aux objets : with

```
// pour accéder directement aux propriétés et méthodes d'un objet sans devoir le spécifier à chaque fois.
with(utilisateur2)
{
    document.write("<p>Le nom d'utilisateur est : "+ nomUtilisateur);
    document.write(". Le prénom de l'utilisateur : " + prenom + "</p>");
    afficherUtilisateur2();
}

/*  Le nom d'utilisateur est : p.debelgique. Le prénom de l'utilisateur : Philippe
• Prénom : Philippe
• Nom d'utilisateur : p.debelgique
• Mot de passe : mathilde
*/
```

Exercices

- Créer un objet chien qui a :
  - Un nom
  - Une race
  - Une couleur
  - Une méthode aboie qui écrit à l'écran "Waff"
  - Une méthode medaille qui écrit à l'écran :
    - "Nom : *nomChien*"
    - "Race : *raceChien*"
    - "Couleur : *couleurChien*"

PROTOTYPES

Pourquoi utiliser les prototypes ?

Le prototype est la "classe" (c#) / fonction qui a créé l'objet

- Avantages : économiser de la mémoire

car dans l'exemple (page 5), chaque utilisateur va avoir sa méthode afficherUtilisateur()

OR elle fait la même chose à chaque fois.

DONC pour économiser de la mémoire, on ajoute la méthode uniquement à son prototype.

Comment ajouter une méthode au prototype ? (1)

```
function Utilisateur3(pre, nomUt, mdp)
{
   this.prenom = pre;
   this.nomUtilisateur = nomUt;
   this.motdepasse = mdp;

   Utilisateur3.prototype.afficher = function(){
      document.write("<ul><li>" + this.prenom + "</li>");
      document.write("<li>" + this.nomUtilisateur + "</li>");
      document.write("<li>" + this.motdepasse + "</li></ul>");
   };
}
```

Comment ajouter une méthode au prototype ? (2)

```
function Utilisateur4(pre, nomUt, mdp)
{
	this.prenom = pre;
	this.nomUtilisateur = nomUt;
	this.motdepasse = mdp;
}

Utilisateur4.prototype.afficher = function(){
	document.write("<ul><li>" + this.prenom + "</li>");
	document.write("<li>" + this.nomUtilisateur + "</li>");
	document.write("<li>" + this.motdepasse + "</li></ul>");
};
```

Utiliser une méthode sur un prototype

```
var ut4 = new Utilisateur4("Jules", "j.cesar", "rome");
ut4.afficher();
console.log(ut4);
```

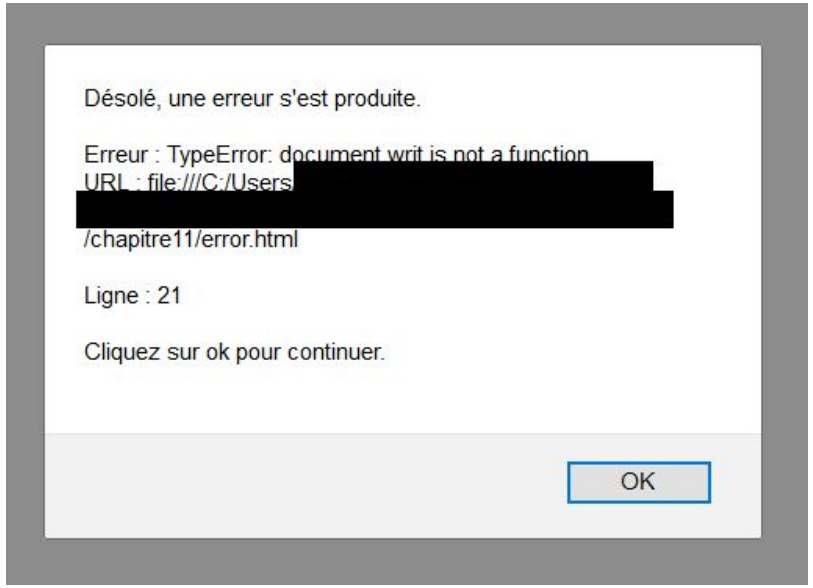
Exercices

- Ajouter une méthode **addRole** au prototype de l'utilisateur qui permet d'ajouter une propriété **role** à la classe Utilisateur3

ONERROR

ONERROR

```
function errorHandler(message, url, ligne)
{
	out = "Désolé, une erreur s'est produite. \n\n";
	out += "Erreur : " + message + "\n";
	out += "URL : " + url + "\n\n";
	out += "Ligne : " + ligne + "\n\n";
	out += "Cliquez sur ok pour continuer.\n\n";
	alert(out);
	return true;
}
onerror = errorHandler
document.writ("Bienvenue") // erreur
```



ECMAScript 6 vs ECMAScript 5

NOUVEAUTÉS

Constantes

ECMAScript 6

```
const PI = 3.141593
PI > 3.0
```

ECMAScript 5

```
Object.defineProperty(typeof
global === "object" ? global :
window, "PI", {
   value: 3.141593,
   enumerable: true,
   writable: false,
   configurable: false
 })
PI > 3.0;
```

La portée des variables

ECMAScript 6

```
for (let i = 0; i < a.length; i++) {
     let x = a[i] …
}
for (let i = 0; i < b.length; i++) {
     let y = b[i] …
}
let callbacks = []
for (let i = 0; i <= 2; i++) {
     callbacks[i] = function () {
          return i * 2
     }
}
callbacks[0]() === 0
callbacks[1]() === 2
callbacks[2]() === 4
```

ECMAScript 5

```
var i, x, y;
for (i = 0; i < a.length; i++) {
     x = a[i]; …
}
for (i = 0; i < b.length; i++) {
     y = b[i]; …
}
var callbacks = [];
for (var i = 0; i <= 2; i++) {
     (function (i) {
          callbacks[i] = function() {
               return i * 2;
          };
     })(i);
}
callbacks[0]() === 0;
callbacks[1]() === 2;
callbacks[2]() === 4;
```

La portée des fonctions

ECMAScript 6

```
{
    function foo () {
        return 1
    }
    foo() === 1 {
        function foo () {
            return 2
        }
        foo() === 2
    }
    foo() === 1
}
```

ECMAScript 5

```
(function () {
    var foo = function () {
        return 1;
    }
    foo() === 1;
    (function () {
        var foo = function () {
            return 2;
        }
        foo() === 2;
    })();
    foo() === 1;
})();
```

Fonctions fléchées : corps

ECMAScript 6

```
impairs = evens.map(v => v + 1)
pairs = evens.map(v => ({
    even: v,
    odd: v + 1 })
)
nums = evens.map((v, i) => v + i)
```

ECMAScript 5

```
impairs = evens.map(function (v) {
    return v + 1;
});
pairs = evens.map(function (v) {
    return {
        even: v,
        odd: v + 1
    };
 });
nums = evens.map(function (v, i) {
    return v + i;
});
```

Fonctions fléchées : this

ECMAScript 6

```
this.nums.forEach((v) => {
   if (v % 5 === 0)
      this.fives.push(v)
})
```

ECMAScript 5

```
// variante 1
var self = this;
this.nums.forEach(function (v) {
   if (v % 5 === 0)
      self.fives.push(v);
});
 // variante 2
this.nums.forEach(function (v) {
   if (v % 5 === 0)
      this.fives.push(v);
}, this);
```

Gestion des paramètres

ECMAScript 6

```
function f (x, y = 7, z = 42) {
   return x + y + z
}
f(1) === 50
```

ECMAScript 5

```
function f (x, y, z) {
   if (y === undefined)
      y = 7;
   if (z === undefined)
      z = 42;
   return x + y + z;
 };
f(1) === 50;
```

Gestion des paramètres

ECMAScript 6

```
function f (x, y, ...a) {
    return (x + y) * a.length
}
f(1, 2, "hello", true, 7) === 9
```

ECMAScript 5

```
function f (x, y) {
    var a =
Array.prototype.slice.call(arguments,
2);
    return (x + y) * a.length;
};
f(1, 2, "hello", true, 7) === 9;
```

Gestion des paramètres

ECMAScript 6

```
var params = [ "hello", true, 7 ]
var other = [ 1, 2, ...params ]
// [ 1, 2, "hello", true, 7 ]
function f (x, y, ...a) {
   return (x + y) * a.length
}
f(1, 2, ...params) === 9
var str = "foo"
var chars = [ ...str ]
// [ "f", "o", "o" ]
```

ECMAScript 5

```
var params = [ "hello", true, 7
];
var other = [ 1, 2
].concat(params);
// [ 1, 2, "hello", true, 7 ]
function f (x, y) {
    var a =
Array.prototype.slice.call(argume
nts, 2);
   return (x + y) * a.length;
};
```

Concaténation

ECMAScript 6

```
var customer = { name: "Foo" }
var card = {
   amount: 7,
   product: "Bar",
   unitprice: 42
}
var message = `Hello
${customer.name},
want to buy ${card.amount}
${card.product} for a total of
${card.amount * card.unitprice}
bucks?`
```

ECMAScript 5

```
var customer = { name: "Foo" };
var card = {
   amount: 7,
   product: "Bar",
   unitprice: 42
};
var message = "Hello " +
customer.name + ",\n" + "want to
buy " + card.amount + " " +
card.product + " for\n" + "a
total of " + (card.amount *
card.unitprice) + " bucks?";
```

Concaténation

ECMAScript 6

```
get`http://example.com/foo?bar=
${bar + baz}&quux=${quux}`
```

ECMAScript 5

```
get([
"http://example.com/foo?bar=",
"&quux=", "" ],bar + baz,
quux);
```

Concaténation

ECMAScript 6

```
function quux (strings, ...values) {
    strings[0] === "foo\n"
    strings[1] === "bar"
    strings.raw[0] === "foo\\n"
    strings.raw[1] === "bar"
    values[0] === 42
}
quux`foo\n${ 42 }bar` String.raw`foo\n${
42 }bar` === "foo\\n42bar"
```

ECMAScript 5

```
// no equivalent in ES5
```

Objets

ECMAScript 6

```
var x = 0, y = 0
obj = { x, y }
```

ECMAScript 5

```
var x = 0, y = 0;
obj = {
   x: x,
    y: y
};
```

Objets : noms des propriétés calculés

ECMAScript 6

```
let obj = {
   foo: "bar",
   [ "baz" + quux() ]: 42
}
```

ECMAScript 5

```
var obj = {
   foo: "bar"
};
obj[ "baz" + quux() ] = 42;
```

Objets : noms des propriétés calculés

ECMAScript 6

```
obj = {
    foo (a, b) {
        …
    },
    bar (x, y) {
        …
    },
    *quux (x, y) {
    …
    }
}
```

ECMAScript 5

```
obj = {
    foo: function (a, b) {
        …
    },
    bar: function (x, y) {
        …
    },
    // quux: no equivalent in ES5 …
};
```

Assignation déstructurée

ECMAScript 6

```
var list = [ 1, 2, 3 ]
var [ a, , b ] = list
[ b, a ] = [ a, b ]
```

ECMAScript 5

```
var list = [ 1, 2, 3 ];
var a = list[0], b = list[2];
var tmp = a;
a = b;
b = tmp;
```

Assignation déstructurée

ECMAScript 6

```
var obj = { a: 1 }
var list = [ 1 ]
var { a, b = 2 } = obj
var [ x, y = 2 ] = list
```

ECMAScript 5

```
var obj = { a: 1 };
var list = [ 1 ];
var a = obj.a;
var b = obj.b === undefined ? 2 : obj.b;
var x = list[0];
var y = list[1] === undefined ? 2 :
list[1];
```

Assignation déstructurée

ECMAScript 6

```
function f ([ name, val ]) {
   console.log(name, val)
}
function g ({ name: n, val: v }) {
   console.log(n, v)
}
function h ({ name, val }) {
console.log(name, val)
}
f([ "bar", 42 ])
g({ name: "foo", val: 7 })
h({ name: "bar", val: 42 })
```

ECMAScript 5

```
function f (arg) {
   var name = arg[0];
   var val = arg[1];
   console.log(name, val);
};
```

Assignation déstructurée

ECMAScript 6

```
var list = [ 7, 42 ]
var [ a = 1, b = 2, c = 3, d ] = list
a === 7
b === 42
c === 3
d === undefined
```

ECMAScript 5

```
var list = [ 7, 42 ];
var a = typeof list[0] !== "undefined"
? list[0] : 1;
var b = typeof list[1] !== "undefined"
? list[1] : 2;
var c = typeof list[2] !== "undefined"
? list[2] : 3;
var d = typeof list[3] !== "undefined"
? list[3] : undefined;
a === 7;
b === 42;
c === 3;
d === undefined;
```

Classes : Définition

ECMAScript 6

```
class Shape {
    constructor (id, x, y) {
        this.id = id
        this.move(x, y)
    }
    move (x, y) {
        this.x = x
        this.y = y
    }
}
```

ECMAScript 5

```
var Shape = function (id, x, y) {
    this.id = id;
    this.move(x, y);
};
Shape.prototype.move = function (x, y)
{
    this.x = x;
    this.y = y;
};
```

Classes : Héritage

ECMAScript 6

```
class Rectangle extends Shape {
    constructor (id, x, y, width, height) {
        super(id, x, y)
        this.width = width
        this.height = height
    }
}
class Circle extends Shape {
    constructor (id, x, y, radius) {
    super(id, x, y)
    this.radius = radius }
}
```

ECMAScript 5

```
var Rectangle = function (id, x, y, width, height) {
     Shape.call(this, id, x, y);
     this.width = width;
     this.height = height;
};
Rectangle.prototype = Object.create(Shape.prototype);
Rectangle.prototype.constructor = Rectangle;
var Circle = function (id, x, y, radius) {
     Shape.call(this, id, x, y);
     this.radius = radius;
};
Circle.prototype = Object.create(Shape.prototype);
Circle.prototype.constructor = Circle;
```

Générateurs de fonctions

ECMAScript 6

```
function* range (start, end, step) {
    while (start < end) {
        yield start
        start += step
    }
}
for (let i of range(0, 10, 2)) {
    console.log(i)
    // 0, 2, 4, 6, 8
}
```

ECMAScript 5

```
function range (start, end, step) {
    var list = [];
    while (start < end) {
        list.push(start);
        start += step;
    }
    return list;
}
var r = range(0, 10, 2);
for (var i = 0; i < r.length; i++) {
    console.log(r[i]);
    // 0, 2, 4, 6, 8
}
```

Méthodes de recherche sur des tableaux

ECMAScript 6

```
[ 1, 3, 4, 2 ].find(x => x > 3)
// 4

[ 1, 3, 4, 2 ].findIndex(x => x > 3)
// 2
```

ECMAScript 5

```
[ 1, 3, 4, 2 ].filter(function (x) { return x
> 3; })[0];
// 4

// pas d'équivalent en ES5
```

Méthodes de recherche sur des tableaux

ECMAScript 6

```
let depth = 2
let chaine1 = "*".repeat(4 * depth)
console.log(chaine1)
// "********"

let chaine2 = "foo".repeat(3)
console.log(chaine2)
// "foofoofoo"
```

ECMAScript 5

```
var depth = 2;
var chaine1 = Array((4 * depth)+ 1).join("*");
console.log(chaine1);
// "********"

var chaine2 = Array(3 + 1).join("foo");
console.log(chaine2)
// "foofoofoo"
```

Méthodes de recherche sur des strings

ECMAScript 6

```
"hello".startsWith("ello", 1) // true
"hello".endsWith("hell", 4) // true
"hello".includes("ell") // true
"hello".includes("ell", 1) // true
"hello".includes("ell", 2) // false
```

ECMAScript 5

```
"hello".indexOf("ello") === 1; // true
"hello".indexOf("hell") === (4 -
"hell".length); // true
"hello".indexOf("ell") !== -1; // true
"hello".indexOf("ell", 1) !== -1; // true
"hello".indexOf("ell", 2) !== -1; // false
```

Promesses

ECMAScript 6

```
function msgAfterTimeout (msg, who, timeout) {
    return new Promise((resolve, reject) => {
        setTimeout(() => resolve(`${msg} Hello ${who}!`), timeout)
    })
}
msgAfterTimeout("", "Foo", 100).then((msg) =>
     msgAfterTimeout(msg, "Bar", 200) ).then((msg) => {
        console.log(`done after 300ms:${msg}`)
    }
)
```

Promesses

ECMAScript 5

```
function msgAfterTimeout (msg, who, timeout, onDone) {
    setTimeout(function () {
        onDone(msg + " Hello " + who + "!");
    }, timeout);
}
msgAfterTimeout("", "Foo", 100, function (msg) {
    msgAfterTimeout(msg, "Bar", 200, function (msg) {
        console.log("done after 300ms:" + msg);
    });
});
```

Format de date

ECMAScript 6

```
var l10nEN = new Intl.DateTimeFormat("en-US")
var l10nFR = new Intl.DateTimeFormat("fr-FR")
var dateEN = l10nEN.format(new
Date("2015-01-02"))
console.log(dateEN)
// "1/2/2015"
var dateFR = l10nDE.format(new
Date("2015-01-02"))
console.log(dateFR)
// "02/01/2015"
```

ECMAScript 5

```
// pas d'équivalent en ES5
```

La liste complète est disponible sur : http://es6-features.org/