

# Javascript

## Introduction



# Les origines

- Créé en 1995 par Brendan Eich pour la société Netscape
- Devait au départ s'inspirer de Scheme (programmation fonctionnelle)
  - Le projet se nomme LiveScript
  - Il est conçu comme un langage serveur (en remplacement de PERL ou des applications CGI de l'époque)
- Le marketing décide que la syntaxe doit ressembler à celle de Java
  - Suite à un accord avec SUN, le projet est renommé Javascript
  - Le langage s'exécutera sur le navigateur web (côté client)



# Définition

- Langage interprété
- Typage dynamique
- Interprétation laxiste
- multi paradigme procédural, fonctionnel ou objet (sans classe)



# Historique

- 1995 : naissance de JavaScript
- 1997 : Dynamic HTML, manipulation des objets HTML avec JavaScript
- 2002 : Ajax (XMLHttpRequest)
- 2006 : jQuery
- 2008 : JavaScript 1.8, Chrome V8 engine
- 2009 : AngularJS, NodeJS
- 2012 : Apache Cordova
- 2014 : Ionic 1.0
- 2015 : JavaScript ES6



# Utilisations

Interpréteur	Applications
V8	Google Chrome, Opera, NodeJS
SpiderMonkey	Firefox, Adobe Acrobat, MongoDB
Chakra	Microsoft Edge
JScript	Microsoft Internet Explorer
JScript .NET	Microsoft .NET
Rhino	Java SE
Nashorn	Java JDK
ActionScript	Adobe Flash, Adobe Flex, Adobe Air
Adobe ExtendScript	Adobe Creative Suite
JavaScriptCore	Safari, Mac OS X



# Contextes d'utilisations

- ◉ Navigateurs Web
- ◉ Programmation serveur (NodeJS)
- ◉ Bases de données NOSQL
- ◉ Extensions, scripting des applications ou des systèmes d'exploitation
- ◉ Automatisation des tâches (Grunt, Gulp)
- ◉ Intégration dans un framework de développement (.NET, Java SE)



Javascript Web



Où placer les code ?



# Dans le fichier html

```
<script>  
  instructions javascript;  
</script>
```

uniquement pour du code  
spécifique à une page, ne  
permet pas la factorisation



# Dans un fichier séparé

```
<script src="js/mons-script.js">  
</script>
```

Permet la  
factorisation et le  
partage du code  
entre les pages,  
solution la plus  
souvent adoptée

Attention, il ne faut  
pas mettre de code  
JavaScript entre les  
ouverture et  
fermeture de balise



# Premier script

```
<script>
```

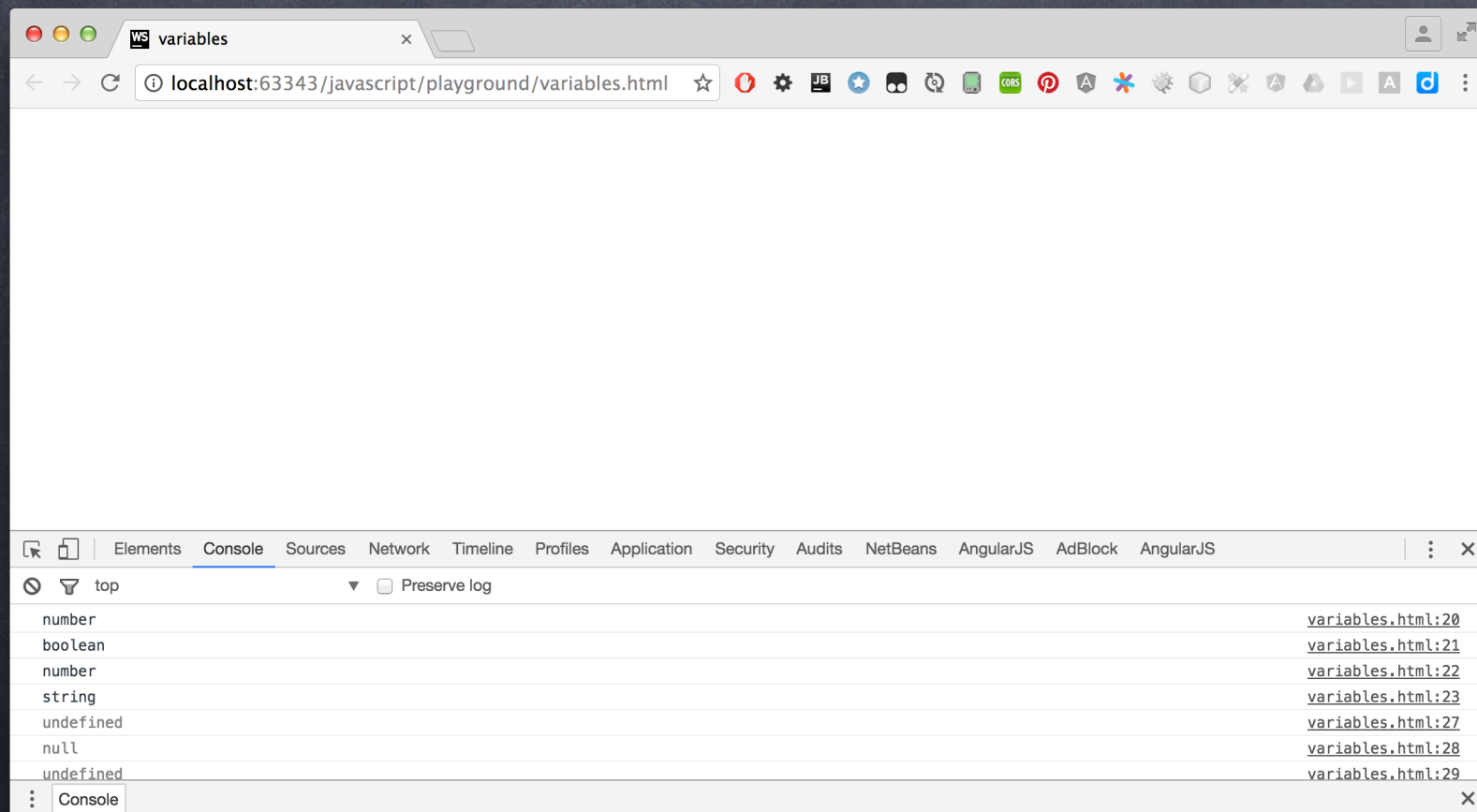
```
//affiche une boite  
//d'alerte  
alert('coucou');
```

```
</script>
```

- Le script s'exécute au chargement de la page
- Les instructions se terminent par un ; (facultatif)
- // pour un commentaire mono-ligne



# Afficher la console



Firefox

CTRL + MAJ + K

Chrome

CTRL + MAJ + J

Internet Explorer

F12

Safari

CTRL + ALT + C



Les variables



# Définition

- Une information stockée en mémoire vive pendant l'exécution d'un programme
- Au cours de l'exécution du programme, cette information peut être lue ou modifiée



# Déclaration et initialisation

```
<script>  
  var nom = "Seb";  
  var age = 40;  
</script>
```

- Déclaration avec le mot clef var (facultatif)

```
<script>  
  var nom, age  
  nom = "Seb";  
  age = 40;  
</script>
```

- Initialisation avec l'opérateur =



# Syntaxe alternative

```
<script>  
  var nom = "Seb",  
      age = 40;  
</script>
```

Déclarations et  
affectations  
multiples séparées  
par des virgules



# Hoisting

```
<script>  
  nom = "Seb";  
  age = 54;  
  
  var nom, age;  
</script>
```

L'affectation d'une variable peut avoir lieu avant sa déclaration



# Nom des variables

- Pas d'espaces
- Pas d'accents
- Pas de caractères spéciaux ou de ponctuation hormis \$ et \_
- Ne peut commencer par un chiffre (mais peut en contenir après le premier caractère)
- Eviter les mots réservés de JavaScript
- Attention à la casse, age et Age sont deux variables différentes pour JavaScript



# Nomenclature

- Underscore, chaque mot est séparé par un tiret bas
- CamelCase, la première lettre est en minuscule, les mots suivants débutent par une majuscule
- PascalCase, tous les mots débutent par une majuscule



# Quelques conseils

- ◉ Noms explicites, indiquent le rôle, le contenu et éventuellement l'unité
  - ◉ `isAuthenticated`
  - ◉ `unitPrice`, `totalBeforeTax`
  - ◉ `buildingLengthInMeters`, `halfLifeInYears`
- ◉ Adopter une nomenclature et s'y tenir
- ◉ Préférer les noms en anglais



# Les types de variables

- Typage dynamique
  - Le type de la variable est déterminé par son contenu
  - Le type n'est pas fixe, il peut changer pendant l'exécution du programme



# Les types scalaires

- number : un nombre entier ou décimal
- string : une suite de caractère alphanumériques
- boolean : true ou false
- undefined : le type d'une variable non encore initialisée



# Les types composites

- array : un tableau indicé de valeurs
- object : un objet
- function : une fonction



# Le type number

```
var entier = 5;  
var decimal = 3.8;
```

//Notation scientifique

```
//54 * 10^3 = 54000  
entier = 54e3;
```

```
//54 * 10^-3 = 0.054  
entier = 54e-3;
```

```
//octal base 8  
//8 en base 10  
var a = 010;
```

```
//hétéradécimal  
//255 en base 10  
var a = 0xFF;
```



# Le type string

```
var nom = "Maloron";  
var prenom = 'Sébastien';  
  
//Chaîne vide  
var phrase= "";  
  
phrase = "il m'a dit";  
  
var code = '<p class="actif">';  
  
phrase = "Il m'a dit : \"vas\" ";
```

\n	saut de ligne
\r	retour charriot
\t	tabulation
\"	guillemet
\'	apostrophe
\\	anti-slash
\t	tabulation



# obtenir le type d'une variable

opérateur typeof

```
var nom = "Seb";  
  
//retourne string  
console.log(typeof nom);
```



# Le type undefined

```
var nom;  
  
//retourne undefined  
console.log(typeof nom);  
  
nom = "Seb";  
  
//retourne string  
console.log(typeof nom);  
  
/*la variable x n'a pas  
été déclarée, son type  
est undefined également*/  
console.log(typeof x)
```

Le type d'une variable :

- non déclarée
- déclarée mais non encore initialisée



Les entrées sorties



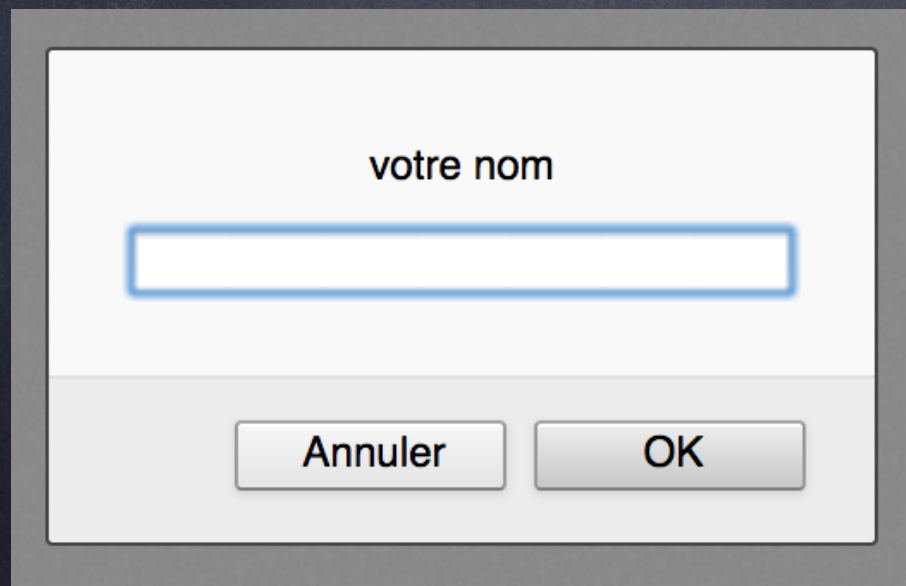
# sorties

```
<script>  
  var nom = "Seb";  
  
  //sortie dans une fenêtre popup  
  alert(nom);  
  
  //sortie dans la console  
  console.log(nom);  
</script>
```

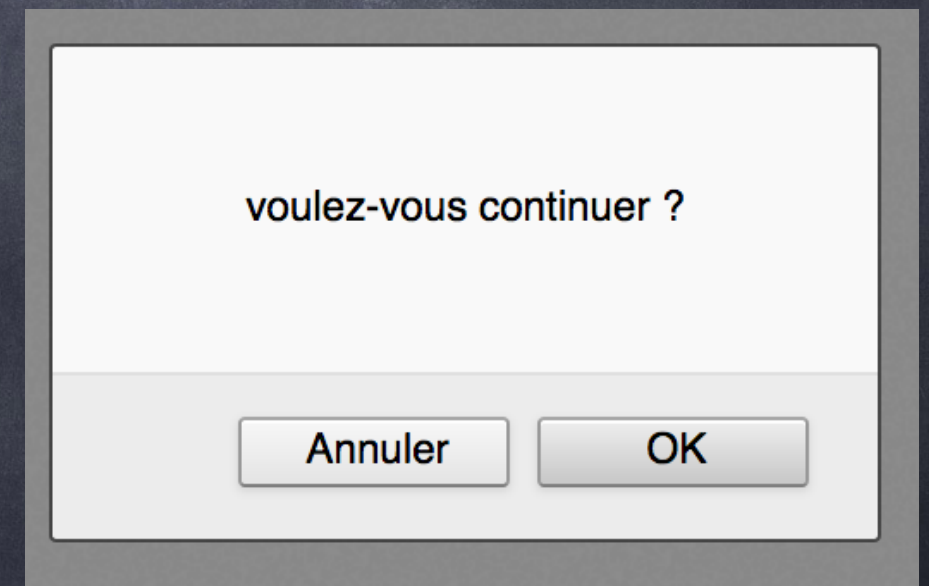


# Entrées

```
<script>  
  //fenêtre de saisie  
  var nom = prompt("votre nom");  
  
  //fenêtre de confirmation  
  var proceed = confirm("voulez-vous continuer ?");  
</script>
```



A screenshot of a JavaScript prompt dialog box. The title bar is not visible. The main area contains the text "votre nom" above a single-line text input field. At the bottom, there are two buttons: "Annuler" (Cancel) on the left and "OK" on the right.



A screenshot of a JavaScript confirm dialog box. The title bar is not visible. The main area contains the text "voulez-vous continuer ?". At the bottom, there are two buttons: "Annuler" (Cancel) on the left and "OK" on the right.



Les expressions



# Définition

- Une ou plusieurs données (opérandes)
- Une ou plusieurs opérations (opérateur)
- retourne un résultat



# Exemple

opérande

5 + 2

opérande

opérateur

résultat ou évaluation  
de l'expression : 7



# Opérateurs arithmétiques

opérateur	description	exemple pour a=5	résultat
+	addition	$a + 2$	7
-	soustraction	$a - 2$	3
/	division	$a / 2$	2.5
*	multiplication	$a * 2$	10
%	modulo reste de la division entière	$a \% 2$	1



# Opérateurs d'affectation

opérateur	description	exemple pour $a = 5$	résultat
$=$	affectation	$a = 2$	2
$+=$	affectation addition	$a += 2$	7
$/=$	division	$a /= 2$	2.5
$*=$	multiplication	$a *= 2$	10
$\%=$	modulo reste de la division entière	$a \%= 2$	1



# Opérateurs d'affectation

opérateur	description	exemple pour a = 5	résultat
++	incrémententation	a++	6
--	décrémententation	a--	4

## post-incrémententation

```
var a = 1;  
/* retourne 2 car a est  
incrémenté après l'évaluation  
de l'expression */  
console.log(a++ * 2);
```

## pré-incrémententation

```
var a = 1;  
/* retourne 4 car a est  
incrémenté avant l'évaluation  
de l'expression */  
console.log(++a * 2);
```



# Opérateurs de comparaison

opérateur	description	exemple pour a = 5	résultat
==	égal	a == 5	true
===	identique	a === "5"	false
>	strictement supérieur	a > 5	false
>=	supérieur ou égal	a >= 5	true
<	strictement inférieur	a < 5	false
<=	inférieur ou égal	a <= 5	true
!=	différent	a != 5	false
!==	non identique	a !== "5"	true



# Opérateurs Logiques

opérateur	description	exemple pour a = true et b = false	résultat
&&	ET logique	a && b	false
	OU logique	a    b	true
!	Négation logique	! (a && b)	true



# condition ternaire

condition ? valeur si vrai : valeur si faux

```
var total = 120;  
//10% de remise si total > 100  
var remise = total > 100 ? 0.1 : 0;
```

```
var total = 120;  
//10% de remise si total > 100  
var remise = total > 100 ? 0.1 : 0;
```



# Concaténation

- combinaison de deux chaîne de caractères
- même opérateur que pour l'addition



# Problème

```
//Prompt retourne un entier  
var nombre = prompt("Entrer un nombre");  
var addition = nombre + 1;  
  
//Pour nombre = 1  
//affiche 11 soit la concaténation  
//de "1" et 1  
console.log(addition)
```



# Conversions de chaînes en nombres

conversion	fonction	exemple pour a = 2.5	résultat
en entier	<code>parseInt(string, base)</code>	<code>parseInt(a)</code>	2
en décimal	<code>parseFloat(string)</code>	<code>parseFloat(a)</code>	2.5

chaîne commence par	base par défaut
0	8
0x	16
tout autre chiffre	10



# Problème résolu

```
//Prompt retourne un entier  
var nombre = prompt("Entrer un nombre");  
var addition = parseInt(nombre) + 1;  
  
//Pour nombre = 1  
//affiche 2  
console.log(addition)
```



# Précédence des opérateurs

ordre décroissant

type	opérateurs
regroupement	(..)
membre	. []
invocation/instanciation	() new
négation, signe, incrémentation	! - + ++ --
unaire	typeof void delete
arithmétiques	* / % + -
relationnel	< <= > >= in instanceof
égalité	== != === !==
logique	&&
condition ternaire	... ? ... : ...
affectation	= += -= *= /= %=
virgule	,



# Applications

Demander un nom, un prénom et un âge,  
afficher :

"bonjour <prenom> <nom> vous avez <age> ans"  
dans la console

Afficher true lorsque l'âge est supérieur ou égal  
à 18



# Applications

Demander un montant HT et afficher dans la console le montant TTC (tva 20%).

Demander le diamètre d'une cercle, afficher dans la console son périmètre et son aire.  
(Math.PI donne une valeur de PI)



# Applications

Demander une température en degrés celsius et convertir en degrés Fahrenheit.

$$\text{fahrenheit} = (\text{celsius} * 9/5) + 32$$

Demander une taille en mètres et convertir en pieds et en pouces.

$$1 \text{ pied} = 30.48 \text{ cm}, 1 \text{ pouce} = 2.54 \text{ cm}$$



# Structures de contrôle



# Condition

```
if(condition) {  
    instructions;  
}
```

La condition est  
une expression  
booléenne

```
if(condition) {  
    //condition == true  
    instructions;  
} else {  
    //condition == false  
    instructions  
}
```

La clause else est  
invoquée lorsque la  
condition est fausse



# Exemple

```
msg = "Entrer votre âge";  
var age = prompt(msg);  
  
if(age < 18){  
    console.log("mineur");  
} else {  
    console.log("majeur")  
}
```



# Erreur classique

```
msg = "Entrer votre âge";  
var age = prompt(msg);  
  
//Cette assertion est toujours vraie  
if(age = 18){  
    console.log("vous avez 18 ans");  
}  
  
//Affiche 18  
console.log(age);
```



# Expression booléenne multiple

```
msg = "Entrer votre âge";  
var age = prompt(msg);
```

```
if(userName == "admin" &&  
    password == "pass"  
) {  
    console.log("autorisé");  
} else {  
    console.log("interdit")  
}
```



Dans une expression booléenne multiple  
avec l'opérateur &&  
si la première assertion est fausse  
les autres ne sont pas évaluées

Faux

ET

Vrai

=

Faux

Faux

ET

Faux

=

Faux



# Évaluation à false

```
var age = parseInt(
  prompt("Votre age");
)
if(age ){
  console.log("valide");
}

//age n'est pas true (ni false)
if(age == true){

}
```

valeurs évaluées  
comme false

0

chaîne vide

undefined

null



# Conditions multiples

```
if(condition1){  
  
    if(condition2){  
        instructions;  
    }else{  
        instructions;  
    }  
  
} else {  
    instructions;  
}
```

KO

```
if(condition1){  
    instructions;  
  
else if(condition2){  
    instructions;  
  
}else{  
    instructions;  
}
```

OK



# Switch

```
switch(note) {  
  case 1:  
    msg = "Nul";  
    break;  
  
  case 2:  
    msg = "Mauvais";  
    break;  
  
  ...  
  
  default:  
    msg = "notes de 1 à 5";  
}
```

Evalue les  
différentes  
valeurs d'une  
expression

Performances  
plutôt déplorables  
en Javascript



# Applications

- Valider la saisie d'un âge (nombre entre 7 et 77 ans)
- Demander une quantité de produit commandé (le prix unitaire étant de 12 € HT). Afficher le total TTC en appliquant une remise de 5% si le total HT est supérieur à 400 et 10% s'il est supérieur à 1200.  
Note : la remise s'applique sur le montant HT



# Applications

En fonction du diagramme sur la diapo suivante, faire un programme de jeu d'aventure qui demande les choix du joueur et affiche le résultat (qui peut être de nouveaux choix)



vous rencontrez un troll  
dans la forêt

action

Fuir

rapide

Non

Oui

Le troll vous  
rattrape et  
vous dévore

Vous vous  
sauvez de  
justesse

Combattre

fort

Non

Oui

Le troll vous  
assomme et  
vous dévore

Vous  
massacrez  
ce pauvre  
troll

Se cacher

furtif

Non

Oui

Le troll vous  
trouve et  
vous dévore

Le troll ne  
vous voit pas  
et passe son  
chemin



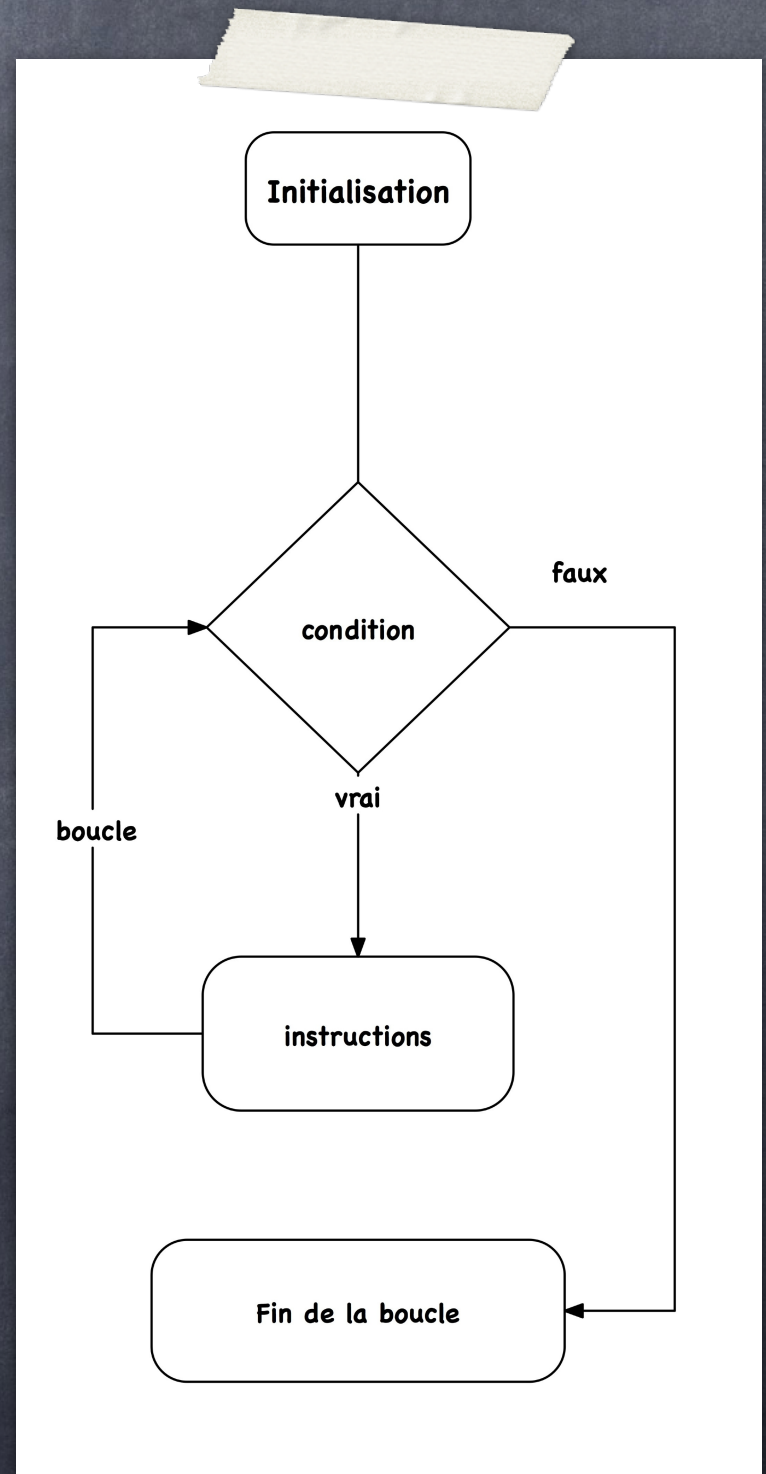
# Boucle

Exécuter des instructions  
plusieurs fois



# Tant que

```
while(condition){  
  instructions;  
}
```

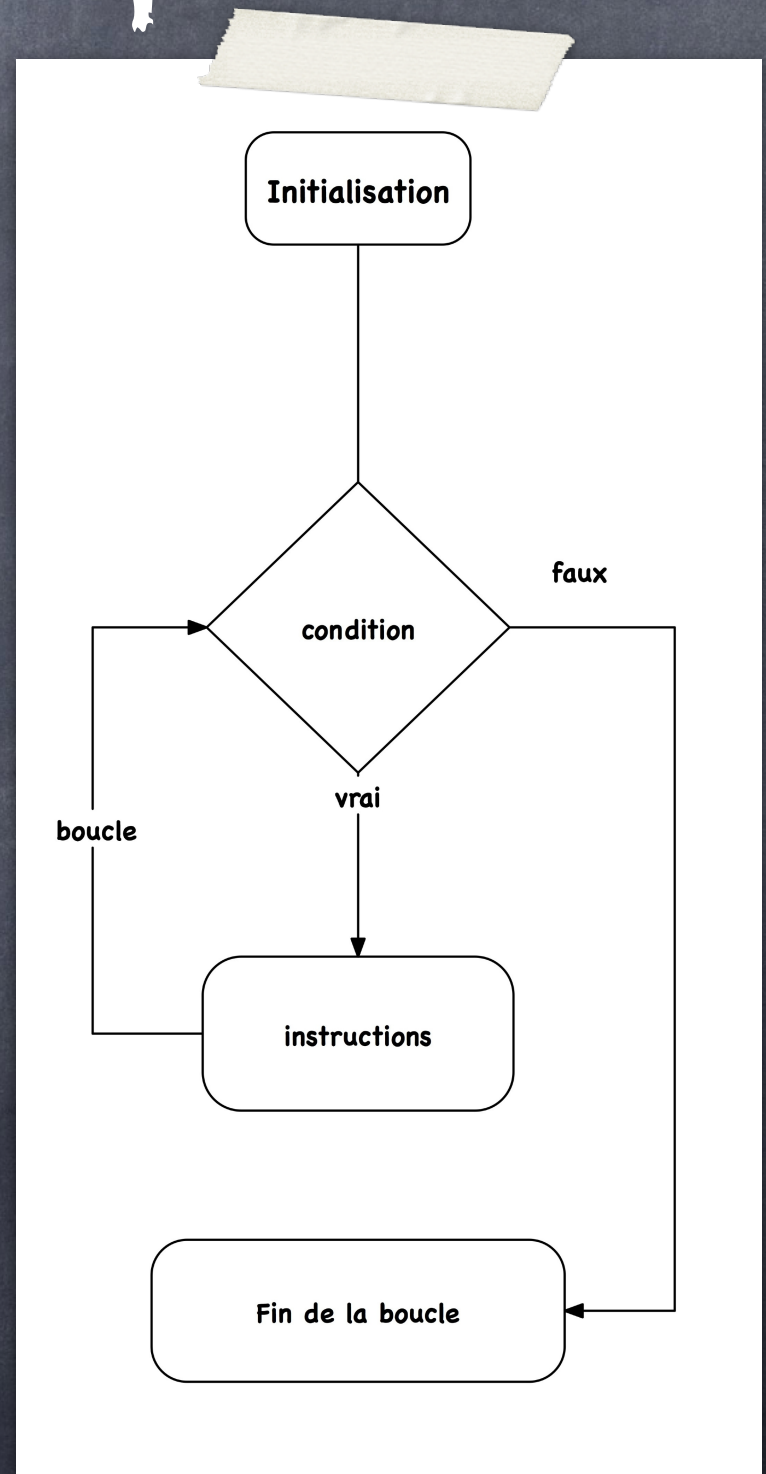




# Jusqu'à ce que

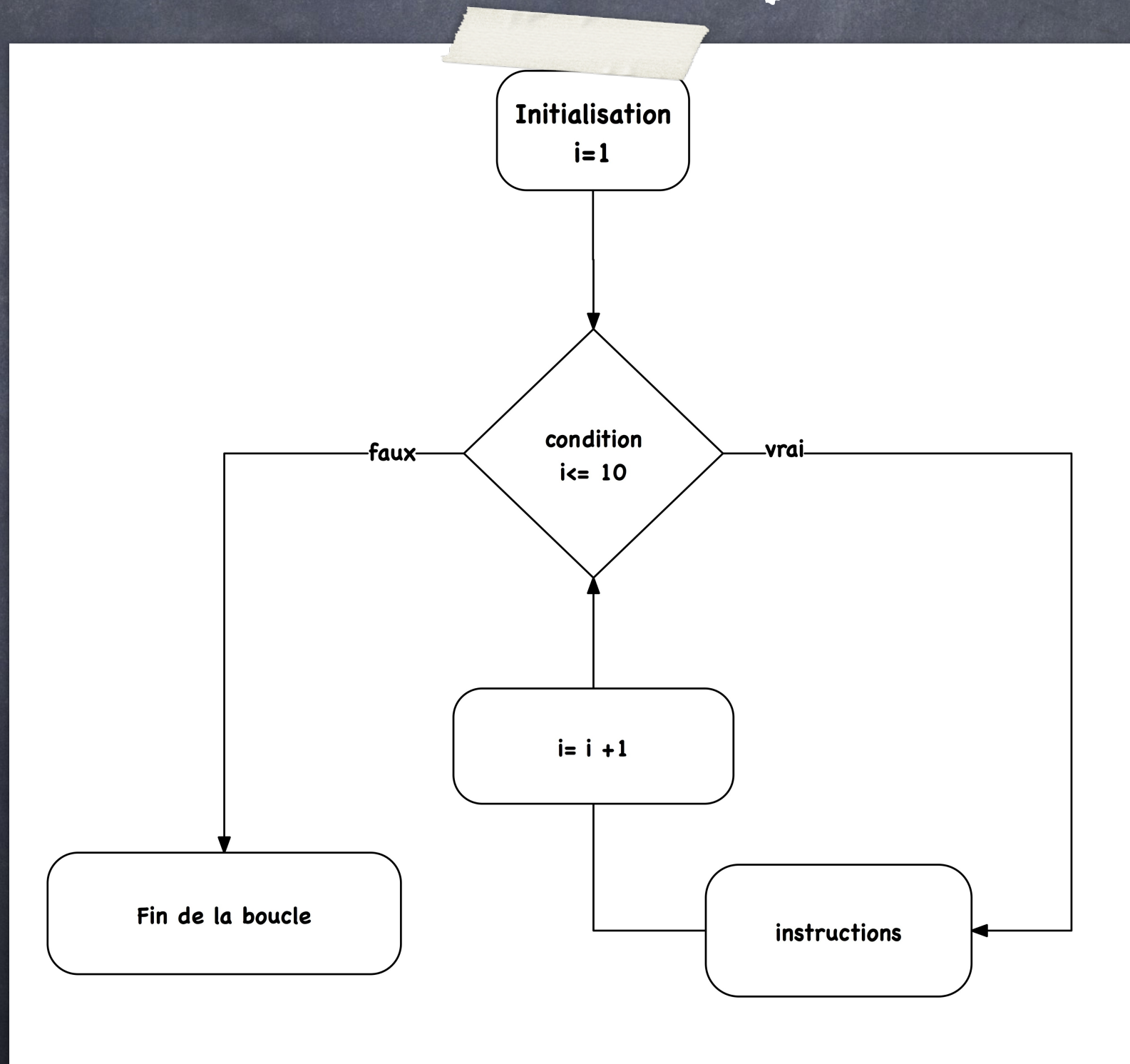
```
do {  
  instructions;  
} while(condition);
```

Les instructions sont  
exécutées au moins  
une fois





# Avec compteur





# Syntaxe

```
for(initialisation; condition; expression){  
    instructions;  
}
```

```
for(var i = 1; i <=10; i++){  
    instructions;  
}
```

```
for(var i = 10; i >0; i-=2){  
    instructions;  
}
```



# Application

Écrire un programme qui affiche les formes suivantes dans la console

```
*****
```

```
*****  
*****  
*****  
*****
```

```
*  
  
**  
  
***  
  
****  
  
*****
```



# Application

Écrire un programme qui demande la saisie d'un entier dans une boucle. Si la saisie n'est pas un entier afficher une boîte de dialogue demandant à l'utilisateur s'il veut quitter l'application. Si tel est le cas afficher la somme des entiers saisis sinon demander à nouveau la saisie d'un entier.



# Application

Écrire un programme qui génère un nombre aléatoire entre 1 et 100 puis qui demande à l'utilisateur de saisir un nombre et ce tant que le nombre aléatoire n'a pas été trouvé. A chaque tentative, le programme indiquera si le nombre saisi est plus petit ou plus grand que le nombre à deviner.

A la fin du jeu, le programme affichera le nombre de tentatives qui ont été nécessaires pour gagner.

(`Math.random()` retourne un aléatoire entre 0 et 1)



Tableaux



# Présentation

indice	valeur
0	pomme
1	poires
2	oranges
3	coings
4	groseille

- Liste de données indexées par une clef numérique
- Premier indice 0



# Déclaration

```
var fruits = new Array(  
  "pommes", "poires"  
);
```

OU

```
var fruits = ["pommes", "poires"];
```

un tableau peut contenir  
des données de types différents  
y compris des types composites  
(tableaux, objets, fonctions)



# Accès aux éléments

```
var fruits = ["pommes", "poires", "oranges"];

//Affiche le deuxième élément
console.log(fruits[1]);

//Modifie la valeur du troisième élément
fruits[2] = "figues";

//Ajoute un élément
fruits[3] = "mangues";
```



# Opérations sur les tableaux

opération	code
ajouter un élément à la fin d'un tableau	<code>array.push(valeur)</code>
ajouter un élément au début du tableau	<code>array.unshift()</code>
obtenir le nombre d'éléments d'un tableau	<code>array.length</code>
retourner et supprimer le dernier élément	<code>el = array.pop()</code>
retourner et supprimer le premier élément	<code>el = array.shift()</code>



# Parcourir un tableau

```
var fruits = ["pommes", "poires", "oranges"];  
  
var fruitsSize = fruits.length;  
  
for(var i= 0; i < fruitsSize; i++) {  
    console.log(fruits[i]);  
}
```



# Parcourir un tableau

```
var fruits = ["pommes", "poires", "oranges"];  
  
for(var item in fruits) {  
    console.log(item);  
}
```



# Rechercher une valeur

```
var fruits = ["poires", "oranges", "poires"];  
var search = "poires";
```

```
//Retourne l'indice de la première occurrence  
//de la valeur recherchée  
//ou -1 si la valeur est absente  
console.log(fruits.indexOf(search));
```

```
//indice de la dernière occurrence  
console.log(fruits.lastIndexOf(search));
```



# Extraire un sous-tableau

```
var fruits = ["figues", "oranges", "poires", "mangues"];

//Extrait de la position 1 à la position 3 (non inclus)
//["oranges", "poires"]
var subset = fruits.slice(1, 3);

//Extrait de la position 1 à la fin du tableau
//["oranges", "poires", "mangues"]
var subset = fruits.slice(1);

//Extrait les deux derniers éléments
//["poires", "mangues"]
var subset = fruits.slice(-2);
```



# Supprimer des éléments

```
var fruits = ["figues","oranges","poires", "mangues"];  
  
//supprime 2 éléments à partir de la position 1  
//et retourne un tableau des éléments supprimés  
//["oranges","poires"]  
  
var deleted = fruits.splice(1,2);
```



# Insérer des éléments

```
var fruits = ["figues", "oranges", "poires", "mangues"];

//insère 2 éléments à partir de la position 1
fruits.splice(1, 0, "kiwis", "citrons");

//supprime 1 élément à partir de la position 2
//et en insère 1
fruits.splice(2, 1, "coings");
```



# Joindre des tableaux

```
var fruits = ["figues", "oranges", "poires", "mangues"];  
var legumes = ["courges", "carottes", "aubergine"];  
var legumineuses = ["soja", "pois chiche", "fèves"];  
  
//insère 2 éléments à partir de la position 1  
var aliments = fruits.concat(legumes, legumineuses);
```



# Application

Écrire un programme qui élimine les valeurs en doublons dans le tableau suivant

```
var fruits = [  
  "pommes", "oranges",  
  "poires", "oranges",  
  "pommes", "figues",  
  "poires"  
];
```



# Application

Écrire un programme qui affiche la position et la valeur du plus grand élément du tableau suivant.

Afficher également la somme et la moyenne des éléments

```
var fruits = [  
  5, 8, 12,  
  1, 48, 3  
];
```



# Application

Écrire un programme qui permute la position des valeurs "oranges" et "figues"

```
var fruits = [  
  "pommes", "oranges",  
  "poires", "figues"  
];
```



# Application

Écrire un programme qui compose une phrase en sélectionnant aléatoirement un sujet, un verbe, un complément d'objet direct et un complément circonstanciel. Pour cela vous pouvez utiliser le fichier `cadavre_exquis.js`.



Les tableaux associatifs  
ou  
objets littéraires



# Présentation

clef	valeur
nom	Maloron
prenom	Sébastien
age	45

Liste de données  
indexées par une clef  
alphanumérique



# Déclaration

```
var person = {  
  nom: 'Maloron',  
  prenom: 'Sébastien',  
  age: 45  
};
```

```
console.log(person.nom);  
console.log(person['age']);
```



# Parcours

```
for(var key in person){  
    console.log(key + "=" + person[key]);  
}
```

Attention : La performance de la boucle for in est bien plus faible que celle des autres types de boucle



# Combinaisons

```
var person = {  
  nom: 'Maloron',  
  prenom: 'Sébastien',  
  age: 45,  
  
  skills: ['Java', 'Python', 'PHP'],  
  
  address: {  
    street: '5 rue Orfila',  
    zipCode: '75020',  
    city: 'Paris'  
  }  
};
```



# Combinaisons

```
var persons =  
[  
  {nom: 'Maloron', prenom: 'Sébastien'},  
  {nom: 'Maloron', prenom: 'Vincent'},  
  {nom: 'Maloron', prenom: 'Clémence'},  
];
```



# Clefs dynamiques

```
var person = {};
```

```
//Le fait d'affecter une valeur  
//génère automatiquement une clef  
person.nom = 'Maloron';  
person.prenom = 'Sébastien';
```

```
console.log(person);
```



# Tester l'existence d'une clef

```
var person = {};
```

```
//avec l'opérateur in  
if('age' in person){  
  console.log('la clef age existe');  
}
```

```
//méthode alternative  
if(person.hasOwnProperty('nom')){  
  console.log('la clef nom existe');  
}
```



# valeur par défaut

```
var person = {};
```

```
//avec une expression ternaire  
var age = 'age' in person?  
    person.age:  
    20;
```

```
//méthode alternative  
var nom = person.nom || 'toto';
```



# Application

Créer une structure de données modélisant un panier de commerce électronique.

Écrire un programme qui calcule le total du panier



# Application

A partir de la source de données du fichier livres.js

Écrire un programme qui retourne :

- tous les livres dont le prix est inférieur à 20
- tous les livres possédant le tag : Base de données
- tous les livres qui ont plus d'un auteur
- la somme du prix des livres parlant d'informatique



# Application

A partir de la source de données du fichier livres.js

Écrire un programme qui :

- supprime tous les livres de Jean-Marie Defrance
- ajoute un livre à la fin du tableau
- ajoute un livre au début du tableau
- augmente le prix des livres d'informatique de 10%



# Les fonctions



# Définition

- Regroupe une série d'instructions sous une dénomination
- Ces instructions peuvent être exécutées en invoquant la fonction
- Il est possible d'invoquer plusieurs fois une fonction
- Une fonction peut elle même invoquer d'autres fonctions



# Utilité

- Organiser et simplifier le code
  - décomposer un problème global en unités simples
- Factoriser, éviter la duplication du code
- Auto-documenter le code, améliorer la lisibilité
- Améliorer la fiabilité du code
- Améliorer notre productivité en tant que développeur



# Création et invocation

```
function hello(){  
  console.log('hello');  
}
```

```
var hello = function(){  
  console.log('hello');  
}
```

```
//Hello 5 fois  
for(var i = 1; i<= 5; i++) {  
  hello();  
}
```



# Arguments

```
function hello(name){  
  console.log('hello ' + name);  
}  
  
hello('world');
```



# Nombre variable d'arguments

```
function sum(){  
  var total = 0;  
  var argNumber = arguments.length;  
  for(var i = 0; i < argNumber; i++){  
    total += arguments[i];  
  }  
  console.log(total);  
}  
  
sum(5,3,8);
```



# Valeur de retour

```
function centimetersToInches(cm) {  
    return cm / 2.54;  
}
```

```
console.log(  
    centimetersToInches(45)  
);
```



# Retour multiple

```
function centimetersToImperial(cm) {  
    var feet = parseInt(cm/30.48);  
    var remains = cm % 30.48;  
    var inches = parseInt(remains / 2.54);  
  
    return {  
        pouces: inches,  
        pieds: feet  
    }  
}
```



# Scope

```
//portée globale  
var greet = "Hello";  
  
function hello(){  
  //portée locale  
  var name = "world";  
  
  return greet + " " + name;  
}
```



# Closures

```
function greet(greeting){  
    //Retourne une fonction anonyme  
    return function(name){  
        //greeting est passé à la fonction  
        return greeting + " " + name;  
    }  
}  
//fonction expression  
var greetInFrench = greet("Bonjour");  
  
console.log(greetInFrench("Seb"));
```



# Callbacks

```
var persons = [];  
function addPerson(infos, callback){  
    persons.push(infos);  
    callback();  
}  
  
//Callback exécuté à la fin du traitement  
//de la fonction addPersons  
function showPersons(){  
    console.log(persons);  
}  
  
addPerson({name:"Jean"}, showPersons);
```



# IIFE

Immediately Invoked Function Expression

```
(function () {  
    var n = 5;  
    console.log(n);  
})();
```

```
//Erreur, n n'est pas dans le scope global  
console.log(n);
```



# IIFE

passage d'arguments

```
(function (w, input) {  
    //w est un alias de l'objet window  
    console.log(w);  
  
    //input vaut 15  
    console.log(input);  
})(window, 15);
```



# Module

```
var module = (function () {  
    var moduleName = "monModule";  
  
    var process = function(){  
        console.log("private process");  
    };  
  
    return {  
        getName: function(){  
            return moduleName;  
        }  
    }  
})();
```



# Application

- ▶ Écrire une fonction qui retourne la circonférence d'un cercle en fonction de son diamètre
- ▶ Écrire une fonction qui retourne un entier aléatoire entre des valeurs min et max passées en argument
- ▶ Écrire une fonction nommé stats. Cette fonction admettra un tableau ordinal de nombres en argument et retournera un tableau associatif contenant les clefs somme, moyenne, min, max avec les valeurs correspondantes



# Application

- ▶ Écrire une fonction qui retourne le nombre et le type de pièces à rendre en fonction du prix et du montant inséré (en euros) dans l'automate. Cette fonction retournera un tableau associatif dont les clefs seront les valeurs des pièces acceptées en centimes (5, 10, 20, 50, 100, 200).



# Application

Écrire une fonction nommée `htmlTag` qui admet deux arguments :

- le nom de la balise

- le contenu de la balise

La fonction retourne le contenu encadré par des balise ouvrantes et fermantes.

Tester l'imbrication de balises en utilisant un appel de fonction comme argument.

```
htmlTag("div", htmlTag("h1", "titre"));
```



L'objet string



# Quelques méthodes

Fonction	Description
<code>str.length</code>	nombre de caractères dans la chaîne <code>str</code>
<code>str.charAt(pos)</code>	retourne le caractère à l'index <code>pos</code> . Sur les navigateurs récents on peut également utiliser <code>str[pos]</code>
<code>str.charCodeAt(pos)</code>	retourne le code du caractère à l'index <code>pos</code>
<code>str.fromCharCode(code)</code>	retourne le caractère correspondant au code passé en argument
<code>str.trim()</code>	retourne la chaîne <code>str</code> sans les espaces éventuels aux extrémités
<code>str.toLowerCase()</code>	converti la chaîne <code>str</code> en minuscule
<code>str.toUpperCase()</code>	converti la chaîne <code>str</code> en majuscule



# Recherche

Fonction	Description
<code>str.indexOf(search)</code>	retourne la position de la première occurrence de la chaîne <code>search</code> au sein de <code>str</code>
<code>str.lastIndexOf(search)</code>	retourne la position de la dernière occurrence de la chaîne <code>search</code> au sein de <code>str</code>
<code>str.includes(search)</code>	retourne <code>true</code> si la chaîne <code>search</code> est inclus dans <code>str</code>
<code>str.startsWith(search)</code>	retourne <code>true</code> si la chaîne <code>str</code> commence par la chaîne <code>search</code>
<code>str.endsWith(search)</code>	retourne la chaîne <code>str</code> sans le espaces éventuels aux extrémités



# Extraction

Fonction	Description
<code>str.slice(start,end)</code> <code>str.substring(start, end)</code>	retourne une sous chaîne de str depuis l'index inclus)
<code>str.substr(start, length)</code>	retourne l'index
<code>str.replace(search, value)</code>	remplace toutes les occurrences de au sein de str pas la chaîne



# Tableaux

Fonction	Description
<code>str.split(separator)</code>	découpe la chaîne str à chaque occurrence de chaînes de caractères
<code>array.join(separator)</code>	contraire de split, constitue une chaîne de caractère en concaténant toutes les valeurs d'un tableau et en ajoutant la chaîne



# Application

- ▶ Écrire une fonction qui converti une notation underscore en camelcase. ex : `nom_du_client` sera converti en `nomDuClient`.
- ▶ Écrire une fonction qui fait l'inverse, convertir `nomDuClient` en `nom_du_client`



# Application

Écrire une fonction qui génère un mot de passe aléatoire en passant en argument le nombre de caractères du mot de passe. La chaîne retournée devra comporter au moins un chiffre, un caractère en minuscule et un en majuscule.



L'objet Date



# Définition

- Une date est définie sous la forme d'un entier représentant le nombre de milli-secondes écoulées depuis le premier janvier 1970



# Construction

```
//Date et heure en cours  
var d1 = new Date();  
//Nombre de millisecondes  
var d2 = new Date(864000000);  
  
//année, mois, jour (janvier = 0)  
var d3 = new Date(2016,12,5);  
  
//ajout des heures, minutes, secondes, millisecondes  
var d4 = new Date(2016,12,5, 15,5,20,0);  
  
var d5 = new Date("October 13, 2016 12:15:00");
```



# Affichage

<code>d.toString();</code>	Mon Dec 05 2016
<code>d.toLocaleDateString();</code>	05/12/2016
<code>d.toLocaleString();</code>	05/12/2016 à 15:05:20
<code>d.toLocalTimeString();</code>	15:05:20
<code>d.toISOString();</code>	2016-12-05T14:05:20.000Z
<code>d.toJSON();</code>	2016-12-05T14:05:20.000Z
<code>d.toUTCString();</code>	Mon, 05 Dec 2016 14:05:20 GMT
<code>d.getTimeString();</code>	15:05:20 GMT+0100 (CET)



# getters

<code>d.getFullYear();</code>	L'année sur 4 chiffres
<code>d.getMonth();</code>	Le mois de 0 à 11
<code>d.getDate();</code>	Le jour du mois de 1 à 31
<code>d.getDay();</code>	Le jour de la semaine de 0 à 6 0 = dimanche
<code>d.getHours();</code>	L'heure de 0 à 59
<code>d.getMinutes();</code>	La minute de 0 à 59
<code>d.getMilliseconds();</code>	Les millisecondes de 0 à 999
<code>d.getTime()</code>	Nombre de milisecondes depuis le 01/01/1970
<code>d.getTimezoneOffset();</code>	Décalage en minutes entre la date locale et la date UTC



# setters

<code>d.setFullYear(year);</code>	L'année sur 4 chiffres
<code>d.setMonth(month);</code>	Le mois de 0 à 11
<code>d.setDate(date);</code>	Le jour du mois de 1 à 31
<code>d.setDay(day);</code>	Le jour de la semaine de 0 à 6 0 = dimanche
<code>d.setHours(hours);</code>	L'heure de 0 à 59
<code>d.setMinutes(min);</code>	La minute de 0 à 59
<code>d.setMilliseconds(ms);</code>	Les millisecondes de 0 à 999



# Calculs

```
//Date et heure en cours  
var d = new Date(2016, 11, 10);  
  
d.setDate(d.getDate() + 25);  
  
//affiche 04/01/2017  
console.log(d.toLocaleDateString());
```



# Application

Calculer le nombre de secondes écoulées entre le début et la fin d'un script

Ajouter 6 mois et 5 jours à la date du jour

Afficher la date du premier lundi du mois prochain

Calculer le nombre de mois écoulés entre deux dates (en considérant qu'un mois commencé est écoulé)



Timers



# Méthodes

<code>setInterval(callback, delay)</code>	Exécute la fonction callback toutes les delay millisecondes
<code>setTimeout(callback, delay)</code>	Exécute la fonction callback au bout de delay millisecondes
<code>clearInterval(handle)</code> <code>clearTimeout(handle)</code>	Supprime le timer identifié par handle



# Horloge

```
var timer;

function updateTime(){
    var now = new Date();

    console.log(now.toLocaleTimeString());
}

function stopClock(){
    clearInterval(timer);
}

//Arrêt de l'horloge au bout de 20 secondes
setTimeout(stopClock, 20* 1000);
//Mise à jour de l'horloge toutes les secondes
timer = setInterval(updateTime, 1000);
```



# Tableaux fonctions avancées



# Liste des fonctions

<code>array.sort([fn])</code>	trie les éléments d'un tableau éventuellement selon la fonction fn
<code>array.find(fn)</code> <code>array.findIndex(fn)</code>	Retourne la première valeur passant un test défini dans la fonction fn. findIndex retourne l'index au lieu de la valeur
<code>array.every(fn)</code>	Vérifie si tous les éléments du tableau passent le test défini dans la fonction fn
<code>array.some(fn)</code>	Vérifie si au moins un élément du tableau passe le test défini dans la fonction fn



# Liste des fonctions

<code>array.filter(fn)</code>	Retourne un nouveau tableau contenant les éléments du tableau original qui passent le test défini dans la fonction <code>fn</code>
<code>array.forEach(fn)</code>	Appelle la fonction <code>fn</code> pour chaque élément du tableau
<code>array.map(fn)</code>	Retourne un tableau résultant de l'exécution de la fonction <code>fn</code> sur chaque élément du tableau
<code>array.reduce(fn)</code>	Réduit un tableau à une seule valeur (agrégation). Parcourt le tableau du premier indice au dernier.
<code>array.reduceRight(fn)</code>	Réduit un tableau à une seule valeur (agrégation). Parcourt le tableau du dernier indice au premier.



# Tri simple

```
var fruits = ["pommes", "poires", "bananes"];  
//["bananes", "poires", "pommes"]  
fruits.sort();
```

```
//La fonction sort traite les éléments  
//comme des chaînes de caractères  
var list = [50, 111, 9, 200, 38];  
//[111, 200, 38, 50, 9]  
list.sort();
```



# Tri amélioré

```
var list = [50, 111, 9, 200, 38];  
//[111, 200, 38, 50, 9]  
list.sort(function(a, b){  
    return b-a;  
});
```

La fonction anonyme de comparaison  
doit retourner un entier  
positif, négatif ou égal à zéro



# Tri sur un tableau de tableaux associatifs

```
var persons = [  
    {nom: 'toto', age: 50},  
    {nom: 'titi', age: 12},  
    {nom: 'tata', age: 28}  
];  
  
persons.sort(function (a, b) {  
    return b.age - a.age;  
});
```



# Recherche

```
var persons = [  
  {nom: 'toto', age: 50},  
  {nom: 'titi', age: 12},  
  {nom: 'tata', age: 28}  
];  
  
var p = persons.find(function (person) {  
  return person.age < 20;  
});
```



# Arguments

1	currentValue	Obligatoire : la valeur de l'élément en cours
2	index	Facultatif : l'index de l'élément en cours
3	array	Facultatif : le tableau source

valable pour les fonctions  
find, findIndex, forEach, every, some, filter et map



# Parcours

```
var persons = [  
    {name: 'toto', age: 50},  
    {name: 'titi', age: 12},  
    {name: 'tata', age: 28}  
];  
  
persons.forEach(function (person, pos) {  
    console.log(  
        (pos +1) + " " +  
        person.name + "(" + person.age + ")"  
    );  
});
```



# Filtre

```
var persons = [  
    {name: 'toto', age: 50},  
    {name: 'titi', age: 12},  
    {name: 'tata', age: 28}  
];  
  
var p = persons.filter(function (person) {  
    return person.age < 50;  
});
```



# Some

```
var persons = [  
    {name: 'toto', age: 50},  
    {name: 'titi', age: 12},  
    {name: 'tata', age: 28}  
];  
  
//true  
var test = persons.some(function (person) {  
    return person.age < 50;  
});
```



# Every

```
var persons = [  
    {name: 'toto', age: 50},  
    {name: 'titi', age: 12},  
    {name: 'tata', age: 28}  
];  
  
//false  
var test = persons.every(function (person) {  
    return person.age < 50;  
});
```



# Map

```
var persons = [  
    {name: 'toto', age: 50},  
    {name: 'titi', age: 12},  
    {name: 'tata', age: 28}  
];  
  
//false  
var p = persons.map(function (person) {  
    person.name = person.name.toUpperCase();  
});
```



# Reduce

```
var persons = [  
  {name: 'toto', age: 50},  
  {name: 'titi', age: 12},  
  {name: 'tata', age: 28}  
];  
  
var total = persons.reduce(function (total, p) {  
  return total + p.age;  
});
```



# Arguments

1	total	La valeur initiale ou celle retournée par le précédent appel à la fonction
2	currentValue	Obligatoire : la valeur de l'élément en cours
3	index	Facultatif : l'index de l'élément en cours
4	array	Facultatif : le tableau source

valable pour les fonctions  
reduce et reduceRight



# Reduce et reduceRight

```
var list = [1,2,3];

function subtract(total, value) {
    return total - value;
}

// -4 (1-2-3)
var n = list.reduce(subtract);

// 0 (3-2-1)
n = list.reduceRight(subtract);
```



# Application

A partir de la source de données du fichier livres.js

Écrire un programme qui retourne :

- la somme du prix des livres parlant d'économie
- le prix moyen des livres d'informatique
- tous les livres possédant les tags :  
Base de données et PHP
- le nombre de livres par tag sous la forme d'un tableau associatif {PHP: 5, Ajax: 1}



# La gestion des erreurs



# Trois types d'erreurs

- Syntaxe
- Exécution (runtime)
- Logique



# Erreurs de syntaxe

## source

Mon code n'est pas syntaxiquement correct

- J'ai oublié de fermer une parenthèse, une accolade
- J'utilise une fonction inconnue de mon interpréteur



# Erreurs de syntaxe

## solutions

- Être plus attentif
- Utiliser un bon IDE



# Erreurs d'exécution

## source

La valeur d'une variable à un instant  $t$  provoque une erreur

- La saisie de l'utilisateur est incorrecte
- Le type d'une variable a changé



# Erreurs d'exécution

## solutions

- Tester les cas marginaux
- Valider les données avant de les utiliser (saisie utilisateur, arguments fonctions, autres données externes)
- Éviter le transtypage



# Erreurs de logique

## source

Mon code est correct mais ne fais pas ce que je veux

- Mon algorithme n'est pas approprié
- J'ai mal compris le problème



# Erreurs de logique

## solutions

- Spécifier le code (comportement attendu)
- Tester le code avec des valeurs d'entrées et de sortie connues



# Capture des exceptions

```
try {  
    instructions;  
} catch(error) {  
    gestion de l'erreur  
}
```



# Utilité

Capturer des erreurs exceptionnelles (erreur d'exécution) lorsque l'on ne peut être certain de la validité du code car :

- Le code dépend de données externes
- Le code fait appel à des bibliothèques ou fonctions externes



# Example

```
var jsonData = "test";  
var data;  
try  
{  
    data = JSON.parse(jsonData);  
}  
catch(error)  
{  
    console.log("Exception");  
    console.log(error.name);  
    console.log(error.message);  
}
```



# Lever une exception

- Signaler une erreur sans la traiter immédiatement
- Très utile pour créer des bibliothèques de fonctions



# Exemple

```
function calcul(n, divisor){  
  if(! divisor){  
    throw {  
      message:"Division pas zéro interdite",  
      name:"InvalidArgumentException"};  
    }  
  return n / divisor;  
}
```

```
try  
{  
    calcul(5,0)  
}  
catch(error)  
{  
    console.log(error);  
}
```