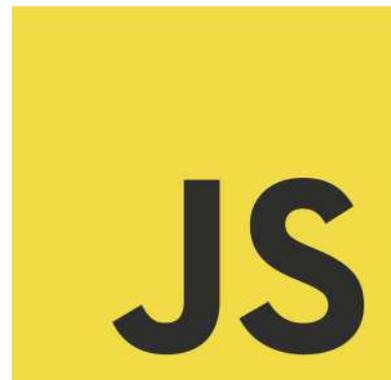


# Introduction au JavaScript

*IT - Développement*  
*JavaScript 101 - Livret de cours*



Noël Macé  
noelmace.com

## Légende

-  Entrée du glossaire
-  Abréviation
-  Référence Bibliographique
-  Référence générale

# Table des matières



<b>Introduction</b>	<b>5</b>
<b>I - Pourquoi et comment ?</b>	<b>6</b>
1. Vous avez dit langage de programmation ?	6
1.1. <i>Qu'est ce qu'un langage ?</i>	6
1.2. <i>Éléments de taxinomie des langages de programmation</i>	7
1.3. <i>Principaux langages de programmation actuels</i>	10
2. Architecture des applications web	11
2.1. <i>Environnement client-serveur</i>	11
2.2. <i>Architecture n-tiers</i>	12
3. Qu'est ce que le Javascript ?	14
3.1. <i>Histoire du Javascript</i>	14
3.2. <i>Les usages du JavaScript</i>	17
<b>II - Premiers pas</b>	<b>18</b>
1. Outils pour le développement	18
1.1. <i>Éditeurs de texte pour le développement</i>	18
1.2. <i>IDE pour JavaScript</i>	19
2. Hello World JavaScript	20
3. Syntaxe	21
3.1. <i>Comment intégrer du JavaScript à du HTML ?</i>	21
3.2. <i>Conventions</i>	23
3.3. <i>Commentaires</i>	25
<b>III - Fondamentaux</b>	<b>26</b>
1. Variables	26
1.1. <i>Déclaration de variables</i>	26
1.2. <i>Types de données</i>	26
1.3. <i>Exercice : Manipulation de variables et types primitifs</i>	27
2. Opérateurs	28
2.1. <i>Opérateurs JavaScript</i>	28
2.2. <i>Exercice : Opérateurs JavaScript : mise en pratique</i>	29
3. Structures de contrôle	30
3.1. <i>Alternatives en JavaScript</i>	30
3.2. <i>Boucles JavaScript</i>	31
3.3. <i>Exercice : Calculs mathématiques simples</i>	32
4. Fonctions	32
4.1. <i>Fonctions</i>	32

4.2. Exercice : Fonctions JavaScript : tester si un nombre est premier .....	33
--	----

## **IV - Programmation Orientée Objet 34**

1. Introduction à la programmation objet .....	34
1.1. Qu'est ce qu'un objet ? .....	34
2. Programmation orientée prototype .....	35
2.1. Introduction .....	35
2.2. Principe .....	35
3. La programmation orientée objet en JavaScript .....	35
3.1. Constructeur .....	35
3.2. Création d'un objet .....	36
3.3. Prototype .....	36
4. Exercice : TODO List .....	37

## **V - Manipulation de pages web 38**

1. Document Object Model .....	38
1.1. Qu'est ce que c'est ? .....	38
1.2. Principaux objets .....	39
2. l'objet document .....	39
2.1. Récupération d'éléments .....	39
2.2. Écriture dans le document .....	40
2.3. Propriétés .....	41
3. Elements DOM .....	41
3.1. Node .....	41
3.2. Element .....	42
3.3. HTML <code>Element</code> .....	42
4. Événements .....	43
4.1. Les différents événements .....	43
4.2. Utiliser des événements sans DOM .....	43
4.3. Gestion d'événements avec DOM2 .....	44

## **VI - Conclusion 46**

1. Exercice : TP : formulaire interactif .....	46
--	----

## **Solutions des exercices 47**

## **Glossaire 52**

## **Abréviations 53**

## **Références 54**

## **Crédits des ressources 55**

# Introduction



JavaScript 101 : les fondamentaux du langage

Pré-requis : faibles notions d'algorithmiques

Durée prévue : entre 14 et 20 heures

Ce cours n'est encore qu'en cours de rédaction et d'amélioration. Tout particulièrement, de nombreuses transitions, remarques et ajouts y sont manquant. Il ne peut donc suffire pour un apprentissage en autodidacte, et est pour l'instant destiné à l'enseignement en présentiel, dans lequel les différents éléments manquant sont présentés oralement, ou en tout cas en dehors de ce support.



# Pourquoi et comment ?



Vous avez dit langage de programmation ?	6
Architecture des applications web	11
Qu'est ce que le Javascript ?	14

## 1. Vous avez dit langage de programmation ?

Bien entendu, vous aurez compris que Javascript est un langage de programmation ... mais savez vous exactement ce que cela signifie, et pourquoi, parmi tout les langages existant, utiliser le Javascript ? Pour vous aider à comprendre ces aspect, nous allons ici introduire quelques concepts essentiels, afin que vous puissiez savoir exactement ce qu'est un langage de programmation informatique, ses particularité, ce qui vous permettra de mieux comprendre par la suite les spécificités du Javascript.

### 1.1. Qu'est ce qu'un langage ?

#### 1.1.1. Définitions



##### Définition : Langage

« Faculté de mettre en œuvre un système de signes linguistiques, qui constituent la langue, permettant la communication et l'expression de la pensée. » - Wikitionnaire



##### Définition : Programme (Informatique)

« Ensemble d'instructions, souvent contenu dans un fichier, écrit dans un langage machine exécutable par un ordinateur. » - Wikitionnaire

#### 1.1.2. Langage de programmation

##### Conclusion

Un langage homme-machine  
Compréhensible par les deux  
Permettant donc de réaliser des programmes



##### Exemple : HelloWorld Python

```
print("Hello World !")
```

demande à l'ordinateur d'afficher (méthode print) le message "Hello World !"



## 1.2. Éléments de taxinomie des langages de programmation

Nous verrons ici les différents éléments permettant la taxinomie des langages de programmation, afin de vous donner les bases essentielles vous permettant la compréhension des caractéristiques de chacun, et donc leur comparaison et classement.

### 1.2.1. Interprété ou compilé ?



#### Définition : Langage compilé

Langage devant nécessairement faire un appel à un *compilateur* afin d'être traduit dans un autre langage (machine / binaire) afin d'être exécuté sur une machine donnée. On parle alors de *compilation*.



#### Définition : Langage interprété

Langage directement analysé, traduit et exécuté par un programmeur informatique dénommé interpréteur. Les opérations d'analyse et de traduction sont réalisées à *chaque exécution*. On parle alors d'*exécution dynamique*.

#### Avantages et inconvénients

La compilation étant faite pour une architecture particulièrement, l'usage d'un langage compilé permet d'optimiser l'usage du hardware (processeur, mémoire, etc ...), alors que l'appel à un interpréteur à chaque exécution d'un programme nécessite plus de ressources. Ainsi, l'usage d'un langage compilé permet d'obtenir de *meilleures performances* par rapport à un langage interprété. Cependant, les performances des machines actuelles et la compilation sur une plateforme souvent légèrement différente de celle sur laquelle le programme sera exécuté limitent de plus en plus cette différence.

En revanche, du fait de leur relative indépendance vis à vis de la machine sur laquelle le programme sera exécuté, les langages interprétés permettent une *meilleure portabilité*  $\Rightarrow$  *portabilité* du code, et une relative *simplification* du développement.

### 1.2.2. Paradigmes



#### Définition : Paradigme de programmation

Modèle cohérent définissant une manière de réaliser un code informatique pour répondre à un problème. Un langage est dit multi-paradigme quand plusieurs paradigmes fondamentaux sont utilisables avec celui-ci.

#### Programmation impérative

Paradigme de programmation définissant un programme en terme de séquences d'instructions exécutées pour modifier l'état du programme. Il s'agit du modèle le plus ancien, puisque le plus proche du fonctionnement réel de la quasi totalité des processeurs du marché. Il est logiquement le plus courant, et donc le plus appris. La programmation impérative peut se décliner en programmation structurée et en programmation procédurale.



## Programmation déclarative

Paradigme de programmation consistant à déclarer les données du problème, puis à demander au programme de les résoudre.

Il existe plusieurs sous-ensembles de la programmation déclarative :

- la programmation descriptive, ne permettant que de décrire des structures de données (comme en HTML par exemple) et à ce titre, souvent considéré comme n'étant pas réellement de la programmation
- la programmation fonctionnelle, plus proche de la logique mathématique, puisque définissant un programme en terme de fonctions et rejetant le changement d'état et la mutation de données, contrairement à la programmation impérative. L'exemple le plus connus de langage à paradigme fonctionnel est Scheme, mais ce paradigme est sans doute le second plus courant derrière la programmation impérative, et est donc supporté par de nombreux langages multi-paradigmes.
- la programmation logique, où les problèmes sont présentés sous forme de prédicats, comme en Prolog par exemple.
- la programmation par contraintes, où les problèmes sont modélisés à l'aide de variables de décision et de contrainte, selon le principe de "Problème de satisfaction de contrainte" ou CSP/CSP<sup>A</sup>. Il s'agit d'un paradigme complexe, utilisé pour la résolution de problèmes combinatoires de grande taille. Il est généralement intégré à un langage via une bibliothèque, comme par exemple Cream pour Java ou python-constraint pour Python.

## Programmation orientée objet (POO)

Paradigme dans lequel un programme est découpé en entités appelées "objets", représentant un concept ou une entité du monde physique, défini par une structure interne, un comportement et une interaction et hiérarchisation avec les autres objets. Ce paradigme est toujours associé à un autre, et ne définit pas en soit la méthode de résolution du problème, mais plutôt la structure du programme. C'est un modèle extrêmement courant. Par exemple, le C++ est, globalement, du C modifié pour s'orienter vers ce paradigme.

### 1.2.3. Typage

Le typage concerne les éléments permettant de déterminer la valeur pouvant être prise par une donnée. Il existe de nombreux termes permettant de distinguer les formes de typages, et donc les langages. Voyons en quelques uns

#### statique / dynamique

Il existe deux méthodes pour déterminer le type pouvant être employé au sein d'une variable :

- dynamiquement : le type est déclaré implicitement, et donc automatiquement déterminé à l'exécution. On dit alors qu'il est déterminé « à la volée. » Cela permet une plus grande souplesse lors du développement, mais entraîne aussi un plus grand risque d'erreurs.
- statique : le type est déclaré explicitement lors de la déclaration d'une variable, et donc déterminé lors de la compilation. Cela permet ainsi une meilleure sûreté de typage, rendre possible l'optimisation du code en fonction de ces informations, ainsi qu'un gain de mémoire.

#### faible / fort

Aujourd'hui, on peut dire que les termes typage faible et typage fort ont été utilisés à tort et à travers, parfois de manière contradictoire, ce qui les a substantiellement vidés de leur sens. Cependant, cela reste un terme très utilisé. En général, on pourra dire qu'un typage est fort si :

- la compilation ou l'exécution peuvent détecter des erreurs de typage
- les conversions implicites de types sont formellement interdites

Si l'une de ces conditions n'est pas remplie, on parlera donc de typage faible.



### 1.2.4. Facilité de prise en main et lisibilité

Bien entendu, tous les langages ne sont pas aussi simples à prendre en main, et tous ne permettent pas une bonne lisibilité. Comme contre exemple ultime, on peut citer le Brainfuck, sorte de "blague d'informaticien" consistant à créer le langage le plus minimaliste mais surtout le plus complexe à utiliser et comprendre, puisque ses instructions sont écrites grâce à des caractères et non des "mots réservés".



#### Exemple : Hello World en Brainfuck

```
+++++++[>++++++>++++++>++++>+<<<<-]>++.>+.++++++..+
```

Heureusement, cela était un contre exemple extrême, mais il demeure que tous les langages ne sont pas égaux sur ce point. Leur structure, leurs concepts, permettent à chacun plus ou moins de simplicité de prise en main et pousse à une écriture plus ou moins respectueuse de certains standards permettant une meilleure lisibilité. Cela reste cependant une guerre de cloché assez vide de sens, et même si certains langages (comme par exemple Python), sont reconnus pour leur simplicité, et d'autres (comme par exemple C), pour leur complexité, vous trouverez toujours des personnes prêtes à vous tenir de longs discours sur la simplicité du C ou la complexité du Python, parfois avec de très bons arguments. Retenez donc que quelque soit le langage, l'essentiel est votre rigueur dans l'écriture, l'indentation, et surtout dans les commentaires, et bien entendu l'étendue de travail que vous êtes prêt à lui consacrer, et donc votre motivation.

### 1.2.5. Usages possibles

Il vous sera également nécessaire de déterminer exactement quel usage vous voulez faire de ce langage dans le cadre de votre/vos projet(s). Car, même si certains langages (comme le C) sont plus ou moins capables de réaliser tout et n'importe quoi, tous ne sont pas orientés vers cet usage, et il est parfaitement possible de gaspiller beaucoup de temps et d'efforts à vouloir réaliser une certaine tâche dans un certain langage alors que sa mise en œuvre aurait pu être bien plus rapide dans un langage mieux orienté vers cet usage. Par exemple, le C peut permettre de la programmation pour le web dynamique, mais un langage orienté à cet usage, comme PHP ou même Java, sera sans doute (pas toujours cependant) plus adapté.

### 1.2.6. Standardisation

La standardisation d'un langage est également un point à aborder, même si secondaire face aux autres sujets ici. Utiliser un langage trop exotique et absolument pas standardisé exclura souvent votre projet de la communauté des développeurs. Mieux vaut donc privilégier un langage standardisé, ou en tout cas répandu, pour s'ouvrir à un maximum de contributeurs. Bien entendu, j'oriente, comme à mon habitude, mon discours dans un cadre "libre et open source" mais cela est également valable si vous souhaitez faire du propriétaire (enfin, venez en parler quand même avant). Bref, dans tous les cas (et même si vous refusez l'ouverture donc), le choix d'un langage hors standards vous privera de nombreux talents, n'utilisant pas ce langage par choix ou méconnaissance.

Comme langage largement standardisé, l'exemple par excellence est surtout le C, qui a donné lieu à plusieurs ISO/CEI/IEC, ce qui n'est pas étranger à sa très large adoption. Cependant, d'autres langages peu ou pas standardisés, comme Python, ne sont pas non plus à exclure, puisque ouverts (nous allons en parler) et très populaires. Gardez à l'esprit que contrairement aux protocoles réseaux, les langages sont peu sujets à la standardisation, et que certains langages standardisés, comme le Fortran, ne sont plus toujours très répandus, ou, comme le C#, très fermés.

### 1.2.7. Licence / Ouverture

Pour conclure, je ne reviendrais pas durant des heures sur le sujet du "logiciel libre et open source", mais sachez que la licence et l'ouverture d'un langage est également primordial. En effet, comment savoir si le langage utilisé par votre projet sera toujours d'actualité, maintenu et utilisé dans des années, voir des décennies. Comme remédier aux problèmes de performance, de fonctionnalités obsolètes, et j'en passe, si son développement est totalement fermé, privé et obscur ? Et enfin, comment pourriez vous véritablement apprendre en profondeur le fonctionnement d'un langage, si vous ne pouvez que vous basez sur les documentations qu'auraient bien voulu vous fournir ses mainteneurs ? Et encore, même si de nombreux efforts sont fait pour créer des plateformes libres pour des langages conçus comme fermés, cette fermeture s'accompagne très souvent de problèmes de compatibilité (le rétro-ingéniering ne pouvant pas tout faire), et de violation de brevets. Bref, je vous recommanderais de fuir comme la peste les .net et autres WLangage (WinDev).

### 1.3. Principaux langages de programmation actuels

#### *Informations générales*

	Langage	Création	Licence	Simplicité	Popularité
1	C	1973	Libre de droit	Faible	18%
2	Java	1991	GNU GPL	Moyenne	16,5%
3	Objective-C	1983	Libre	Moyenne	11%
4	C++	1982	Libre de droit	Faible	7,5%
5	C#	2001	Propriétaire	Moyenne	6%
6	PHP	1994	Libre (licence PHP)	Forte	4,5%
7	Visual Basic	1991	Propriétaire	Faible	3%
8	Python	1991	PSF/PSF <sup>^A</sup> Licence (libre)	Forte	2,5%
9	Javascript	1995	Libre de droit (trademark Oracle)	Forte	1,5%

Popularité basée sur l'index TIOBETIOBE ↻ 2014



*Informations techniques*

Langage	Type	Paradigmes	POOPOO <sup>AA</sup>	Typage
C	compilé	procédural	Non	Statique
Java	semi-interprété (compilation en bytecode)	procédural	Oui	Statique (fort)
Objective-C	compilé	procédural	Oui	Dynamique
C++	compilé	procédural	Oui	Statique
C#	compilé	procédural	Oui	Statique (fort)
PHP	compilé à la volée ou interprété	multi (fonctionnel / procédural )	Depuis la version 5	Dynamique (faible)
Visual Basic	compilé ou interprété	événementiel	Basique	Statique (fort)
Python	interprété	multi (procédural / fonctionnel)	Oui	Dynamique (fort)
Javascript	interprété	multi (procédural / fonctionnel)	Oui (orientée prototype)	Dynamique (faible)

## 2. Architecture des applications web

### 2.1. Environnement client-serveur

*Définition*

---

mode de communication  
à travers un réseau (ou un système)  
entre deux entités (application ou hôte) de rôle distinct

#### 2.1.1. Entités

*Serveur*

---

en "écoute"  
attente de connexions clients  
le plus souvent sur un "port réseau" (ex : TCP 80 pour un serveur HTTP)  
rôle : "offrir le service", des informations, effectuer des traitement lourds, etc ...  
peu répondre aux requêtes de nombreux clients

*Client*

---

"actif"  
établi une demande de connexion au serveur  
rôle : interface / interaction avec l'utilisateur (ou le système client)



## 2.1.2. Types de client

### *Client léger*

---

minimum de traitement

traitements importants effectués sur le serveur

ex : navigateur web, où les traitements sont minimalistes (javascript) du fait de la "pauvreté" de l'HTML

### *Client lourd*

---

application autonome

prend en charge la quasi-totalité des traitements

ne fait généralement appel au serveur que pour échange de données

### *Client riche*

---

offre une interface graphique plus évoluée qu'avec un client léger

avec des fonctionnalités similaires à un client lourd

traitements majoritairement effectués sur le serveur

le client riche effectuant les traitements de présentation et de "finalisation"

ex de standards pour la définition de clients riches : XUL (Mozilla) ou Flex (Adobe)

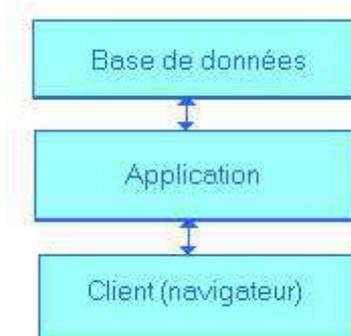
description de l'interface via XML

## 2.2. Architecture n-tiers

Architecture en couche

Généralement en 3 tiers :

- Présentation
- Métier
- Accès aux données



*Architecture 3-Tiers*

### 2.2.1. Trois Tiers

#### *Présentation (des données)*

---

affichage

restitution sur le poste de travail

dialogue avec l'utilisateur

ex : comment afficher un message

#### *Métier (traitement des données)*

---

mise en œuvre de la logique application

ex : comment définir un utilisateur et réaliser son authentification

#### *Accès aux données*

---

Persistance des données

Conservation pérenne des données

ex : base de donnée MySQL

## 3. Qu'est ce que le Javascript ?

### 3.1. Histoire du Javascript

#### *Brendan Eich*

---

Créateur du Javascript

développeur Américain

Netscape de 1995 à la fin (juillet 2003, développement stoppé par AOL)

Membre fondateur de Mozilla :

- architecte du projet Mozilla.org en 1998
- membre du conseil d'administration de la Fondation Mozilla depuis 2003
- tout premier directeur technique de Mozilla Corporation (août 2005)
- créateur du langage Rust



*Brendan Eich*

#### 3.1.1. Premiers pas

##### *Prémises*

---

LiveScript : langage de script coté serveur

développé par Brendan Eich

##### *Création*

---

en décembre 1995 par Brendan Eich pour Netscape

à la manière d'un LiveScript coté client

simple renommage, en accord avec Sun Microsystems

### *Premières implémentations*

---

Netscape Navigator 2.0 est bien entendu le premier à intégrer un moteur JavaScript, en mars 1996.

Ecrit en C par Brendan Eich sous le nom de SpiderMonkey.

(sera par la suite "libéré", et encore aujourd'hui maintenu par la Fondation Mozilla, sous licence MPL)

Rapide succès, du fait de la popularité du navigateur.

Microsoft s'en inspirera pour créer JScript, qu'il intègre à Internet Explorer 3.0 en août 1996.

## 3.1.2. Standardisation

### *Mise en œuvre*

---

Dés l'apparition du JScript chez Microsoft, Netscape décide de réagir, en soumettant JavaScript à Ecma International pour standardisation.

Travail effectué de novembre 1996 à juin 1997.

Résultant à l'adoption du nouveau standard ECMAScript (Standard ECMA-262).

### *Conséquences*

---

ECMAScript (qu'on pourrait désigner comme le "vrai nom" du JavaScript) est dès lors la base de langages très proches :

- JavaScript bien sûr (chez Netscape puis Mozilla)
- JScript de Microsoft
- ActionScript d'Adobe

*Versions*

5 éditions à ce jour :

Version	Date de publication	Description	Javascript	JScript
1	Juin 1997		directement inspiré de Javascript 1.2 (Netscape 4.0)	3.0 (IE 4.0, octobre 1997)
2	Juin 1998	Simple modification rédactionnelle pour se conformer au standard international ISO/IEC 16262.	1.3 (Netscape 4.5 - octobre 1998)	
3	Décembre 1999	Nombreuses améliorations (Expressions rationnelles plus puissantes, meilleure manipulation des chaînes de caractères, levée des exceptions avec le mécanisme try/catch, meilleure gestion des erreurs...)	1.5 (Netscape 6.0, nov 2000; mais aussi plus tard les versions de Netscape et Mozilla)	5.5 (IE 5.5, juillet 2000)
4	Abandonnée	Abandonnée au projet d'ES5. Travaux très critiqués car destinés à abandonner le modèle orienté prototype au profit d'un model plus classique (à la manière du Java ou du C++). Une première version du référentiel d'implémentation d'ECMAScript 4 est disponible depuis juin 2007.		
5	Décembre 2009	Améliore la rétrocompatibilité en annulant certains points de la version 4. Ajout du "strict mode"		
5.1	Juin 2011	Version pleinement compatible avec le standard international ISO/IEC 16262:2011	1.8.5 (Gecko, Firefox, SpiderMonkey et Rhino, avec ajouts de fonctionnalités de la version 6, ou sans pour V8/Chrome)	
6	En cours de développement	Vise à introduire divers nouveaux concepts.	2.0 (travail en cours)	



## 3.2. Les usages du JavaScript

### 3.2.1. Dynamisation de page Web

#### *Pourquoi ?*

---

Chaque navigateur web intègre aujourd'hui un interpréteur JavaScript (ou plutôt ECMAScript), afin de permettre la "dynamisation" du code HTML.

L'HTML est un langage descriptif. Il ne sert donc qu'à afficher des informations, statiquement.

Le JavaScript va, lui, permettre de modifier dynamiquement le comportement de cette page HTML.



#### *Exemple : Quelques exemples*

---

- Afficher/masquer du texte
- Faire défiler des images
- Créer un diaporama avec un aperçu « en grand » des images
- Créer des infobulles
- voir même créer des jeux et des "logiciels" complets (surtout avec HTML5)

### 3.2.2. Autres usages

#### *Le future ? !*

---

Javascript est aujourd'hui, contrairement au passé (et à une certaine fausse image qui demeure dans les esprits), bien plus qu'un simple langage de scripting coté client pour la dynamisation web. C'est un langage en pleine évolution, qui connaît aujourd'hui de nombreuses améliorations et nouvelles application, ce qui en fait sans doute un des langages les plus novateurs et prometteurs.

#### *Exécution coté serveur*

---

Avec node.js, javascript connaît aujourd'hui une croissante popularité pour une exécution coté serveur. Cela n'est cependant pas l'objectif de ce cours, et je vous invite, pour en savoir plus, à aller consulter *cette page*.

#### *Langage d'extension*

---

JavaScript est également largement utilisé, comme le Python, comme langage privilégié pour la création d'extensions de nombreux logiciels. Vous pouvez par exemple utiliser JavaScript pour modifier le comportement de l'interface de Firefox

#### *Mobilité et portabilité*

---

Avec l'arrivée de HTML5 et CSS3, offrant de très nombreuses possibilités, javascript devient également aujourd'hui (et c'est sans doute un de ses aspects les plus prometteurs), un langage permettant enfin de réaliser des projets semi-légers, standards, portables et ouverts. C'est notamment le cas grâce aux projets Cordova / PhoneGAP, ou encore Mozilla Open Web App (mis sur le devant de la scène grâce à FirefoxOS), mais aussi et surtout grâce à l'énorme travail de standardisation réalisé par le W3C sur le sujet.

# Premiers pas



Outils pour le développement	18
Hello World JavaScript	20
Syntaxe	21

## 1. Outils pour le développement

### 1.1. Éditeurs de texte pour le développement

Dans l'ensemble, tout éditeur de texte peu vous permettre de développer dans la très grande majorité des langages. Cependant, certains sont plus adaptés à cet usage. Bien entendu, nous ne verrons pas ici les logiciels propriétaire (vous trouverez sans doute d'autres ressources à ce sujet sur le web, mais leur intérêt, même objectivement, n'est pas bien grand).

#### 1.1.1. Les classiques

##### *VIM*

---

VIMVIM ↵ VIM ^ VIM ↵ est un éditeur original, et, grâce à son propre langage d'extension, très complet. Du fait d'une structure basée sur un concept de "modes" (la touche i, par exemple, active le mode insertion, pour écrire dans le document), même si son apprentissage peu paraître un peu complexe aux débutant, mais est fortement simplifié par la présence d'un tutoriel intégré et très complet (accessible via la commande vimtutor sour GNU/Linux), il procure une grande efficacité une fois celui-ci maîtrisé. Conçu en 1991 par Braam Moolenaar comme une amélioration libre de Vi, un éditeur de référence créé par Bill Joy en 1976 et dès lors présent sur tout les systèmes Unix (bien que non-libre), VIM c'est rapidement imposé comme une référence sur les systèmes Unix actuel (bien que rarement installé par défaut, la plupart des utilisateurs préférerons l'installer en remplacement du Vi minimaliste déjà présent).

Même si vim est à l'origine un éditeur texte en CLI, vous pourrez également utiliser gvim, son pendant en GUI.

##### *Emacs*

---

Même si le terme Emacs représente toute une famille d'éditeurs issus du premier Emacs développé par James Gosling en 1981, le plus connus reste aujourd'hui GNU Emacs, développé en 1984 par Richard Stallman. C'est un éditeur puissant, extensible, et extrêmement personnalisable, très reconnu dans le monde Unix.



##### *Complément : Encore une occasion de troll \o/*

---

« Vim est un éditeur ; il ne cherche pas à inclure "tout sauf l'évier de la cuisine" mais vous pouvez nettoyer le vôtre avec Vim. »

« Emacs est un très bon système d'exploitation auquel il ne manque qu'un bon éditeur de texte. »

## La guerre d'éditeurs

Comme le montrent les citations précédents (et peut être votre réaction à celle-ci), la "guerre d'éditeur", sorte de troll massif, éternel et immortel consistant à essayer de déterminer lequel, de VIM ou GNU Emacs, est le meilleur, est bien toujours d'actualité, même aujourd'hui (bien que dans un petit cercle restreint).

La réelle raison de ce trollage massif ? Les réflexes conditionnés ! En effet, notre cerveau, habitué à réaliser certaines opérations à la suite, définit ce qu'on appelle en neurologie cognitive, un "Task Set", c'est à dire un enchaînement automatique d'actions et de réflexes destinés à réagir promptement à une situation donnée. De ce fait, n'importe qui ayant l'habitude de l'usage d'un outil, quel qu'il soit (VIM, Emacs, Nano, Firefox, Debian, etc ...) possède ces Task Set, qui continuent à intervenir dans un contexte semblable, donc même si vous, utilisateur GNU Emacs acharné, vous retrouvez face à un VIM temporairement. Et que cela donne-t-il ? De l'énerverment, et de la frustration face à un outil qui nous semble "mal fait", "peu ergonomique", et j'en passe ... source de troll si vous souhaitez l'exprimer. Car même si vos affirmations vous semblent alors objectives et fondées, elles ne sont que le fruit de votre premier choix, et de vos habitudes.

Bon après, vous trouverez toujours des gens pour me contredire, mais ça fait partie du jeu !)

## Comparaison "objective"

Avantages réels de Emacs :

- usage plus "naturel" pour un débutant
- donne un style "hacker" ^^
- ça fait plaisir à Stallman

Avantages réels de Vim :

- évite les troubles musculosquelettiques
- plus léger, plus rapide
- garanti par les standards POSIX
- évite les conflits de touches de contrôle

### 1.1.2. Les plus simples

#### gedit

Disponible aussi bien sur les systèmes Unix, y compris sur Mac OS X, que sous Windows, Gedit est un éditeur libre, graphique, et simple d'usage, vous proposant plusieurs fonctionnalités pour la programmation, comme la colorisation syntaxique. Il s'agit de l'éditeur par défaut de nombreux environnements de bureau GNU/Linux.

#### notepad++

Disponible uniquement sur Windows, notepad++ est un bon éditeur de texte libre orienté programmation. Il devrait être suffisant pour la plupart des usages sous Windows, en offrant une très bonne alternative aux éditeurs de texte par défaut que sont le blocnote et notepad.

## 1.2. IDE pour JavaScript

### Eclipse

Un IDE de référence, très complet, car extensible et paramétrable.

Assez gourmand en ressources.

Plus adapté pour des langages plus complexes, comme le Java.

Libre.

Nombreuses extensions pour l'HTML, le JavaScript et le CSS.

Aptana : installable en tant qu'IDE autonome ou extension Eclipse (cf [aptana.com](http://aptana.com))



## *Komodo Edit*

---

Simple et léger.

Très orienté développement web (navigateur intégré, etc ...)

Libre également (licence MPL)

Site officiel : <http://www.activestate.com/komodo-edit>

## *Bluefish*

---

Très simple et léger.

Moins "sexy" mais d'autant plus efficace.

Multilingue.

Libre (licence GPL).

Site officiel : <http://bluefish.openoffice.nl/index.html>

## 2. Hello World JavaScript

Le Hello World est l'exemple classique par lequel démarrer l'apprentissage de tout langage (ou presque). Il consiste simplement en un unique programme permettant d'afficher "Bonjour !" ou "Hello World !" d'une certaine manière (dans la console, dans une fenêtre, etc ...).

Voyons donc ensemble comment réaliser ce premier programme.



### *Exemple : Afficher une boîte de dialogue*

---

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4    <title>Hello World!</title>
5  </head>
6
7  <body>
8
9    <script>
10
11      alert('Hello world!');
12
13    </script>
14
15  </body>
16 </html>

```

*Hello World JS HTML*

### *Ce qu'il faut retenir*

---

la méthode `alert()` permet d'afficher une boîte de dialogue

un script JavaScript peut être intégré à une page html entre des balises `<script>` et `</script>`

une instruction JavaScript se termine par un point virgule

une chaîne de caractère peut s'écrire entre simples quotes



## 3. Syntaxe

### 3.1. Comment intégrer du JavaScript à du HTML ?

#### 3.1.1. Directement dans la page

##### *Balise script*

---

`<script> </script>`

cette balise peu être utilisée pour n'importe quel langage de scripting (comme du VBScript par exemple)

hormis avec HTML5, il est donc recommandé de préciser l'attribut `type="text/javascript"`



##### *Exemple : Hello World !*

---

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Hello World!</title>
5  </head>
6
7  <body>
8
9      <script>
10
11         alert('Hello world!');
12
13     </script>
14
15 </body>
16 </html>

```

*Hello World JS HTML*



##### *Remarque : Commentaires d'encadrement*

---

Insoler le script afin de respecter les standards du W3C.

Ex : l'usage du signe `>` pour la comparaison serait sans cela considéré comme une balise mal fermée, ce qui invalide la page

```

1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
2  <html>
3  <head>
4      <title>Javascript XHTML</title>
5  </head>
6
7  <body>
8
9      <script type="text/javascript">
10
11         <!--
12         if (2 > 0){
13             alert('2 est bien supérieur à 0');
14         }
15         <!-->
16     </script>
17
18 </body>
19 </html>

```

*JavaScript dans XHTML*

### 3.1.2. Fichier séparé

#### *Syntaxe*

---

Attribut src de la balise source.

Le fichier .js utilisé ne nécessite aucune syntaxe particulière.



#### *Exemple*

---

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4    <title>Hello World!</title>
5  </head>
6
7  <body>
8
9    <script src="helloworld.js"> </script>
10
11 </body>
12 </html>

```

*Hello World avec fichier .js*



#### *Remarque : Pourquoi ?*

---

Un fichier externe vaudra toujours mieux qu'un code intégré volumineux, car ceux-ci sont mis en cache par le navigateur, ce qui accélère le chargement.

### 3.1.3. Où positionner la balise script ?

#### *Comportement courant*

---

Très souvent, il est conseillé de placer vos balises scripts dans l'entête du code HTML (head). Ce n'est pas forcément le meilleur choix.



#### *Conseil : En fin de page*

---

Une page web est analysée de manière séquentielle. Donc si un javascript volumineux est déclaré en début de page, celui-ci va être chargé avant les éléments afficher, ce qui risque de ralentir la navigation. Il est donc recommandé de placer vos balises script en fin de page.



#### *Remarque : Mais ...*

---

Les navigateurs modernes ont tendance à, de toute manière, charger le javascript en dernier, quelque soit sa position dans la page ... mais vous n'êtes pas sans savoir que nous sommes dans l'obligation d'encore et toujours prendre en compte les navigateurs obsolètes. Tenez vous en donc à la recommandation précédente.

## 3.2. Conventions

### 3.2.1. Enchaînement

#### *Syntaxes possibles*

---

Le caractère permettant de séparer les instructions pour les enchaîner est, en javascript, le *point virgule*, ou le *retour à la ligne*. Toutes les syntaxes suivantes sont donc "correctes" :

```
instructionA
instructionB
```

```
instructionA ;instructionB
```

#### *Oui mais ...*

---

Cependant, il est courant, dans le développement JavaScript, de "*compresser*" le code, c'est à dire de le transformer (automatiquement) en un fichier quasi illisible, mais bien moins volumineux, afin de réduire les temps de chargement. Les noms de variables et fonctions sont réduites au maximum (exemple : mavariable devient m) et les retours à la ligne supprimés. Faire appel à des retours à la ligne pour enchaîner des instructions est donc une mauvaise méthode, même si acceptée par le JavaScript, car elle rendra alors le code compressé incorrecte. *Les exemples précédents sont donc à éviter !* Nous écrivons donc de la manière suivante :

```
instructionA ;
instructionB ;
```

Ou, de manière "compressée" :

```
instructionA ;instructionB ;
```

### 3.2.2. Indentation



#### *Définition*

---

« Retrait des lignes d'un bloc, pour les distinguer de celles qui ne font pas partie du bloc » - Wikitionnaire

#### *Utilité*

---

Comme la plupart des langages (hormis, notamment, Python), *javascript n'est pas sensible aux espaces*. Cependant, une bonne indentation est indispensable pour la *lisibilité* du code.



### Exemple : Mauvaise indentation

```

1      function startTime()
2      {
3          var today=new Date();
4      var h=today.getHours();
5      var m=today.getMinutes();
6      var s=today.getSeconds();
7          // add a zero in front of numbers<10
8      m=checkTime(m);
9      s=checkTime(s);
10     document.getElementById('txt').innerHTML=h+":"+m+":"+s;
11     t=setTimeout(function(){startTime()},500);
12 }
13
14 function checkTime(i)
15 {
16     if (i<10){
17         i="0" + i;
18     }
19     return i;
20 }

```

Mauvaise Indentation Javascript



### Exemple : Bonne indentation

```

1      function startTime()
2      {
3          var today=new Date();
4          var h=today.getHours();
5          var m=today.getMinutes();
6          var s=today.getSeconds();
7
8          // add a zero in front of numbers<10
9          m=checkTime(m);
10         s=checkTime(s);
11
12         document.getElementById('txt').innerHTML=h+":"+m+":"+s;
13         t=setTimeout(function(){startTime()},500);
14     }
15
16 function checkTime(i)
17 {
18     if (i<10){
19         i="0" + i;
20     }
21     return i;
22 }

```

Bonne indentation JS

### 3.3. Commentaires



#### *Définition*

---

Portion de code non interprétée. Celle ci peut servir à simplifier la lecture par un agent humain, à documenter, ou à désactiver temporairement une ou plusieurs instructions.



#### *Attention*

---

Tout comme une écriture respectueuse des conventions, il est extrêmement important de *documenter* votre code, afin qu'il soit simple compréhensible par vous ou toute autre personne dans le futur. En effet, même si l'intérêt de votre fonction `fct(x)` est évidente pour vous, elle ne sera pas pour les autres, voir même pour vous dans un futur proche. Notez au passage qu'il est tout aussi important de *donner des noms clairs* à vos fonctions et variables.

#### *Commentaire en fin de ligne*

---

Permet de documenter de manière simple une instruction ou un groupe d'instructions.

```
instruction ; //commentaire
//autre commentaire
```

#### *Commentaire multiligne*

---

Permet le retour à la ligne, afin de réaliser un commentaire plus complet, ou de commenter temporairement plusieurs instructions.

```
/* Ceci est un commentaire multiligne.
   Je peu donc faire des retours
   a la ligne !
*/
```

# Fondamentaux



Variables	26
Opérateurs	28
Structures de contrôle	30
Fonctions	32

## 1. Variables

### 1.1. Déclaration de variables

Déclarer une variable, dans un langage à typage dynamique comme le JavaScript, ne consiste qu'à réserver un espace en mémoire pour celle-ci, c'est à dire la "créer" et permettre son utilisation. Pour ce faire, on utilise le mot clé "var", suivi du nom de la variable à déclarer.



#### Attention : Casse et caractères acceptés

JavaScript est *sensible à la casse*. Cela signifie que les variables `mavar` et `MaVar` sont deux variables différentes. De plus, le nom d'une variable (ainsi que d'une fonction), ne peut être composé que de *caractères alphanumériques, underscore "\_" et dollars "\$"*.



#### Exemple : Déclaration et affectation

```
1  var mavar;
2  mavar = 2;
```

*Déclaration et affectation de variable JS*

Ou, de manière plus simple :

```
1  var mavar = 2;
```

*Instruction unique et déclaration et affectation JS*



#### Exemple : Déclaration de plusieurs variables

```
1  var mavar = 1 , mavar2 = 2 , mavar3;
```

*Déclaration de plusieurs variables JS*

### 1.2. Types de données

Comme nous l'avons déjà évoqué, JavaScript est un langage à *typage dynamique "faible"* et *orienté objet*. Cela signifie donc que, d'une part, le type d'une variable sera directement déduit de la dernière valeur lui ayant été affectée, et que, d'autre part, il vous est possible de créer n'importe quel type. Voyons cependant ici quelques types pré-existants, que nous désignerons sous le terme de "primitifs".



### 1.2.1. Les différents types "primitifs"

#### *Number*

---

Le type *number* désigne toute *valeur numérique*, qu'elle soit *entière* (ex : 5) ou à *virgule flottante* (ex : 5.5 ou encore 3e-5, égal à 0.00003). Il est également possible d'écrire ces valeurs sur une *autre base qu'en décimal*, comme par exemple en hexadécimal (ex : 0xa310)

#### *String*

---

Le type *string* désigne toute *chaîne de caractère*, c'est à dire n'importe quel texte, composé d'une suite de caractères. Afin de le distinguer du reste du code, et donc éviter des mauvaises interprétations (espaces, mots clés, chevrons, etc ...), une chaîne de caractère est *encadré par des simples ou doubles quotes*. Par exemple : "Ceci est une chaîne de caractères !"

#### *Boolean*

---

Le type *boolean*, désigne, en français, un élément booléen, c'est à dire *binnaire*, et ne pouvant donc prendre que deux valeurs : *true* ou *false*.

### 1.2.2. Astuce

#### *Tester le type et l'existence d'une variable*

---

Pour cela, nous utilisons le mot clé *typeof* suivi de l'identifiant de la variable à tester, qui renverra une chaîne de caractère contenant le type.



#### *Exemple*

---

```

1  var maVar = 1;
2  var msg = 'maVar is a '
3  document.write(msg + typeof maVar + '<br/>'); //number
4  maVar = '1';
5  document.write(msg + typeof maVar + '<br/>'); //string
6  maVar = true;
7  document.write(msg + typeof maVar + '<br/>'); //boolean
8  document.write('rien is ' + typeof rien + '<br/>'); //undefined

```

*typeof JS*

## 1.3. Exercice : Manipulation de variables et types primitifs

Voyons pour commencer un petit exercice pour mettre en pratique nos connaissances en manipulation simple de variable

## Question

[Solution p 47]

1. Pour commencer, ouvrir votre éditeur de texte ou votre IDE, et créer un premier fichier nommé variables.html
2. Créer la structure de votre page HTML5.
3. Dans une balise script, créer une variable « *maVariable* » et affectez lui "valeur 1"
4. afficher son type
5. demander à l'utilisateur de rentrer un nombre
6. affichez "vous avez entré : " suivi du nombre entré

Indice :

Les méthodes suivantes vous seront utiles :

- *alert('texte')* : pour afficher une boîte de dialogue contenant le texte en argument
- *parseInt('texte')* : pour extraire une valeur numérique de la chaîne de caractère 'texte'
- *prompt('texte')* : pour afficher une boîte de dialogue demandant à un utilisateur d'entrer un texte. Cette fonction prend pour argument le message à afficher (String) et renvoi une chaîne de caractère.

## 2. Opérateurs

### 2.1. Opérateurs JavaScript



#### *Définition : Opérateur*

Symbole (ou chaîne de caractère) permettant de réaliser une opération en deux opérandes.

Par exemple :  $2 + 3$  où  $+$  est l'opérateur arithmétique d'addition, définissant l'opération d'addition des deux opérandes numériques 2 et 3.

#### 2.1.1. Opérateurs logiques

Un opérateur logique permet de réaliser un test entre deux valeurs booléennes (true ou false) afin de renvoyer une autre valeur booléenne. Il s'agit du OU logique " $||$ ", du ET logique " $&&$ ", et de la négation logique " $!$ ". Bien entendu, n'importe quelle fonction renvoyant une valeur booléenne, ou n'importe quelle variable de type booléen peu représenter un opérateur.

#### 2.1.2. Opérateurs de comparaison

Les opérations de comparaison renvoient, tout comme les opérations logique, une valeur booléenne. En revanche, les opérandes peuvent s'appliquer sur tout les types de base (chaîne de caractère, numériques, booléens, etc ...).

<	strictement inférieur
>	strictement supérieur
<=	inférieur ou égal
==	égal
===	identique (égal et de même type)
!=	différent
!==	non identique

### 2.1.3. Opérateurs "mathématiques"

Les opérations dites "mathématiques" (abusivement, puisque les autres opérations appartiennent aussi aux mathématiques, comme par exemple les opérations logiques, relevant de l'algèbre booléenne), comprennent les quatre opérations arithmétiques usuelles (l'addition, la soustraction, la multiplication et la division) et le reste de division entière (modulo). Elles s'appliquent sur des opérandes numériques, et renvoient une valeur numérique.

+	addition
-	soustraction
*	multiplication
/	division
%	reste de division entière (modulo)

### 2.1.4. Opérateur d'affectation

L'affectation consiste à fixer une variable à une valeur donnée. Ses opérandes sont donc une variable et une valeur de même type que cette variable, étant donné que Python est un langage à typage "fort". Le caractère associé est « = », à ne pas confondre avec l'opérateur d'égalité ==.



#### *Complément : Opérateurs d'affectations mathématiques*

Il est également possible d'utiliser des opérateurs qui associent affectation et opération mathématique.

Par exemple : `mavar = mavar / 2` est identique à `mavar /= 2`

## 2.2. Exercice : Opérateurs JavaScript : mise en pratique

Une fois vus les opérateurs et types de données JavaScript, voyons, par la pratique, comment ceux-ci se comportent dans différentes situations.

## Question

[Solution p 47]

Créez une page HTML avec une balise script. Dans cette balise script, déclarez deux variables, en leur affectant deux valeurs de votre choix, puis créer une ou plusieurs alert() afin d'afficher le résultat et le type d'une ou plusieurs opération(s) entre ces valeurs. Modifiez les valeurs, et leur type, ainsi que les opérateurs utilisés, autant de fois que nécessaire afin de bien comprendre le résultat des différentes conversions implicites de types, ainsi que l'utilité de chaque opérateur. Afin d'en profiter pour vérifier vos autres connaissances, évitez autant que possible de consulter l'indice.

Indice :

Exemple de document HTML5 à réaliser, permettant de voir le résultat de "l'addition" entre un entier et une chaîne de caractère.

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4    <title>Test JavaScript</title>
5  </head>
6
7  <body>
8    <script>
9      var v1 = 1;
10     var v2 = '2';
11     var rslt = v1 + v2 ;
12     alert(rslt);
13     alert(typeof rslt)
14   </script>
15 </body>
16 </html>

```

Tests d'opérateurs JS

### 3. Structures de contrôle

#### 3.1. Alternatives en JavaScript

Test Si ...

---

```

1  if (booleen) {
2    instruction;
3  }

```

Test si JS

Test Si ... Sinon ....

---

```

1  if ( booleen ) {
2    // instruction
3  } else {
4    // instruction
5  }

```

Test si ... sinon JS

*Test Si ... Sinon-Si ...*

---

```

1  if (booleen) {
2      // instruction
3  } else if (booleen) {
4      // instruction
5  } else {
6      // instruction
7  }

```

*Test Si ... Sinon ... Si JS**Remarque : Imbrication*

---

Il est bien entendu possible d'imbriquer des tests si. Les opérations logiques et le else if nous permettent cependant de l'éviter la plupart du temps.

## 3.2. Boucles JavaScript

### 3.2.1. Blocs

*Tant que ...*

---

```

1  while (condition) {
2      instruction_1;
3      instruction_2;
4      instruction_3;
5  }

```

*Tant Que JS**Pour ... dans ...*

---

```

1  for (initialisation; condition; incrémentation) {
2      // instruction
3  }

```

*Boucle Pour JS*

### 3.2.2. Contrôle de boucle

*Sortie de boucle*

---

```

1  var mot;
2  while (true) {
3      mot = prompt("entrez un mot : ");
4      if (mot == "") {
5          alert('c'est fini');
6          break;
7      } else {
8          document.write(mot + '<br/>');
9      }
10 }

```

*Exemple de break JS*

### 3.3. Exercice : Calculs mathématiques simples

Histoire de mettre en pratique les fondamentaux du langage Python, réalisons quelques exercices classiques de calcul mathématique.

Question

[Solution p 47]

Factorielle : sachant que la factorielle d'un nombre  $n$  est le produit des nombres entiers strictement positifs inférieurs ou égaux à  $n$ , réaliser un programme `fact.js` calculant la factorielle d'un nombre entier rentré par l'utilisateur.

Question

[Solution p 47]

Créer un programme demandant à un utilisateur de rentrer un nombre, puis un nombre plus petit. Ce programme devra redemander à l'utilisateur de rentrer à nouveau le second nombre tant qu'il est supérieur ou égal au premier.

## 4. Fonctions

### 4.1. Fonctions

*Déclaration*

```
1 function maFonction(arg1, arg2) {
2     // instructions
3 }
```

*Déclaration de fonction JS*

*Appel*

```
5 var retour = maFonction('1', abc);
```

*Appel de fonction JS*

*Valeur de retour*

```
1 function maFonction(arg1, arg2) {
2     // instructions
3     return valeur;
4 }
```

*Valeur de retour de fonction JS*



## Complément : Les fonctions anonymes

Nous verrons plus loin que le concept de "fonction anonymes" est particulièrement utilisé en JavaScript, particulièrement quand il s'agit de POO par prototype. Alors, en quoi cela consiste-t-il ? Notez tout d'abord qu'il n'est en rien obligatoire de donner un nom à une fonctions. Et oui, il est parfaitement possible d'en déclarer comme ceci :

```
1 function (arg) {
2     alert('vous avez fait appel à une fonction anonyme avec ' + arg + ' comme argument');
3 }
```

### Exemple de déclaration de fonction anonyme JS

Oui, mais dans ce cas, comment y faire appel ? Il vous suffit pour cela de stocker cette fonction dans une variable, comme ceci (notez d'ailleurs que la syntaxe précédente n'était pas correcte sans cette affectation) :

```
1 var maFonction = function (arg) {
2     alert('vous avez fait appel à une fonction anonyme avec ' + arg + ' comme argument');
3 };
4
5 maFonction();
```

### Exemple de fonction anonyme avec affectation JS

Notez au passage la présence du point virgule après la fermeture de notre fonction, indispensable pour conclure l'affectation.

Il existe enfin un type particulier de fonction anonyme, que nous appelons les IIFE, pour « *Immediately-Invoked Function Expression* » ou encore « *Expression de fonction immédiatement invoquée* », qui, comme son nom l'indique, permet de créer une fonction anonyme et de l'invoquer directement, sans avoir à l'affecter à une variable (ce qui nous empêche cependant d'y faire appel ultérieurement). Celles-ci nous permettrons d'isoler le code contenu du reste de notre application, et s'écrivent comme ceci :

```
1 (function (arg) {
2     alert('vous avez fait appel à une fonction anonyme avec ' + arg + ' comme argument');
3 })('test');
```

### Exemple d'IIEF JS

## 4.2. Exercice : Fonctions JavaScript : tester si un nombre est premier

Nous allons ici mettre en place un petit calcul simple, afin de mettre en pratique vos quelques premières connaissances en JavaScript.

### Question

[Solution p 48]

Sachant qu'un nombre premier n'est divisible que par 1 et lui même, réalisez une fonction prenant un argument, et testant si cet argument est un nombre, et si ce nombre est premier. Pour la tester, faites tout d'abord appel à votre fonction directement dans votre code, avec le nombre de votre choix, puis demandez à l'utilisateur de rentrer un nombre, pour afficher si celui-ci est premier ou non.

### Indice :

Les fonctions suivantes pourront vous être utiles :

- `isNaN(number)` : permet de tester si un argument est égal à NaN (Not A Number)
- `parseInt(arg)` : revoit un l'entier contenu dans l'argument (NaN si l'argument ne contient pas un nombre)
- `parseFloat(arg)` : idem, mais renvoi le nombre décimal

Pensez également aux opérateurs booléens, ainsi qu'au reste de la division entière (modulo, %).



# Programmation Orientée Objet



Introduction à la programmation objet	34
Programmation orientée prototype	35
La programmation orientée objet en JavaScript	35
Exercice : TODO List	37

## 1. Introduction à la programmation objet

La programmation orientée objet, ou POO, est un paradigme de programmation extrêmement puissant, que tout bon développeur se doit de maîtriser. Car même si vous pouvez programmer dans certains langages sans "faire de l'objet", vous arriverez très rapidement aux limites d'une telle démarche, risquant d'avoir un code rapidement brouillon, et mal organisé. Ce n'est pas sans raison que les langages les plus utilisés aujourd'hui (à l'exception de C, mais c'est une autre question), sont eux même orientés objet. La POO vous permet de structurer votre pensée, et donc votre développement, en des termes simples (oui oui, simples, ne croyez pas les mauvaises langues) et précis. Bien entendu, penser votre programme en terme d'objets demande un peu d'entraînement, mais vous verrez que vous vous y ferez très vite.

### 1.1. Qu'est ce qu'un objet ?

Impossible de parler de POO sans bien comprendre tout d'abord ce qu'est un objet. Pour commencer, faisons simple : un objet est une *représentation d'une entité du monde réel ou d'un concept*. On est donc pas très loin du mot "objet" de la vie courante : il s'agit d'un élément que vous pouvez manipuler, selon ses caractéristiques propres.

D'un point de vue plus "informatique", un objet est avant tout une *structure de donnée*, c'est à dire une entité destinée à agir sur des données et les classer / stocker. Mais en plus des autres structures de données, un objet définit ces données (ses *attributs*), ainsi que les actions possibles sur elles (ses *méthodes*). On définit une *classe*, élément de code (souvent dans un seul fichier) qui définit toutes ces caractéristiques, et on "*instanciera cette classe*" afin de créer un objet particulier (une *instance* de cette classe). Au sein de cette classe, les attributs sont définis par des variables, et les méthodes par des fonctions (c'est un synonymes mais comme ça au moins c'est clair)



#### Fondamental

Une classe est donc l'élément abstrait qui définit un type d'objet (les attributs d'objet n'ont pas de valeur définie), tandis que l'objet est un élément concret, en mémoire. Ses attributs ont donc des valeurs, et le tout est stocké en mémoire (et référencé grâce au nom de l'objet).

Pour résumer, une classe est donc le type d'une variable, et l'objet est son contenu.



### Exemple : Titine, ma Voiture

Nous définissons la classe Voiture. Celle-ci est définie par des attributs (marque, modèle, niveau du réservoir, propriétaire, etc ...) et par des méthodes (allumer le moteur, remplir le réservoir, avancer, etc ...).

Ensuite, nous pouvons créer une variable « *titine* », de type Voiture, tout en définissant la valeur des ses attributs (Porsche, Cayenne, 100%, Pierre Paul, etc ...). On dit qu'on instancie la classe Voiture, c'est à dire qu'on a créé l'objet concret « *titine* », qui est une voiture (une instance de la classe Voiture). A présent, on peu appeller les méthodes de cet objet, comme par exemple `titine.allumerMoteur()`.

## 2. Programmation orientée prototype

### 2.1. Introduction

Aujourd'hui, le modèle le plus courant de programmation orientée objet est la POO par classe, celles-ci définissant nos objets de manière abstraite (et fixe dans la plupart des cas), chacun des objets étant l'instance d'une classe. Il existe cependant un autre modèle pour la programmation orienté objet, que l'on retrouve, par exemple, en JavaScript (le plus connu), en Lua, ou encore en Self (le tout premier langage à intégrer ce principe), et que l'on nommera "programmation orientée prototype". La POO par classe étant bien plus connue et répandue, nous nous permettrons de reprendre ici ses termes pour plus de simplicité.



### Rappel : Programmation par classe

Une classe définie des objets de manière statique.

Tout objet est une instance d'une classe.

Par conséquent, tout caractéristique d'un objet (héritage, méthodes, attributs ...) est définie statiquement.

### 2.2. Principe



### Définition : Prototype

« *Premier exemplaire, ébauche fonctionnelle d'un objet, ou d'une idée développée par un concepteur.* » - Wikitionnaire

Comme l'indique la définition générale du terme, un "prototype" est un objet, permettant de définir, par la suite, un ensemble d'objets sur son modèle. Il en va de même en programmation. Alors qu'en programmation par classe, un objet est définie de manière statique, un prototype, étant lui même un objet, est donc dynamique, et modifiable.

## 3. La programmation orientée objet en JavaScript

### 3.1. Constructeur

Le "prototype", en JavaScript, est réalisé via un "constructeur", de manière un peu similaire à ceux de la programmation par classe. Ce constructeur est donc une *fonction* (dont le nom, par convention, commencera par une *majuscule*, afin de le distinguer de ses "instances", comme dans la plupart des langage à programmation par classe).



### Exemple : Un constructeur en JavaScript

```

1  function Personne(nom, prenom, age) {
2      // attributs
3      this.nom = nom;
4      this.prenom = prenom;
5      this.age = age;
6      // methodes
7      this.anniversaire = function() {
8          this.age++;
9      };
10 }

```

Exemple de constructeur JS

Ici, nous définissons le constructeur "Personne", qui, comme avec une classe, nous permettra de définir des objets.

## 3.2. Création d'un objet

Pour "instancier" un objet à partir de son constructeur, cela se réalise donc exactement de la même manière qu'en programmation par classe : nous faisons appel à la fonction, avec le mot clé *new*.



### Exemple : Création d'une "Personne" en JavaScript

```

12  var jean = new Personne('Jean', 'Pierre', 42);

```

Création d'un objet JS

## 3.3. Prototype

Tout constructeur, en JavaScript, a un attribut *prototype*, qui est la définition générale du type d'objet. Il est donc possible de modifier un prototype dynamiquement.



### Exemple : Définition d'une méthode par prototype en JavaScript

```

12  var jean = new Personne('Jean', 'Pierre', 42);
13  var paul = new Personne('Paul', 'Jacques', 83);
14
15  Personne.prototype.toString = function(){
16      return this.nom + ' ' + this.prenom + ', âgé de ' + this.age + ' ans.';
17  };
18
19  alert(jean.toString());
20  alert(paul.toString());

```

Exemple de définition par prototype JS



### Remarque : Intérêt

L'ajout de méthode via le prototype permet d'ajouter celle-ci "à la volée". Ainsi, inutile de modifier le constructeur pour que tous les objets construits à partir de ce type aient accès à cette méthode. Cela sera particulièrement utile si le constructeur a été défini dans un autre fichier, voir, si vous n'êtes pas l'auteur de ce constructeur (comme par exemple avec String ou Array).

## 4. Exercice : TODO List

Afin de mettre en pratique quelques principes de la POO, sans autre connaissance nécessaire, nous allons ici créer un petit exercice simple et rapide, illustrant l'intérêt de l'approche objet.

Question

[Solution p 48]

Grâce à l'HTML5, au CSS et au JavaScript, réalisez une page similaire à la suivante :

Nom :

Date :

**TODO List**

Nom	Date
Acheter du pain	14/06/14
Trier mails	22/09/14

*Todo List JS : Résultat*

Cette page consiste en un simple formulaire, permettant de rajouter des "tâches" (objets Todo), définis par un nom (name) et une date (date), à un tableau HTML.

Indice :

*Pour ajouter l'élément au tableau, votre formulaire devra :*

- avoir un bouton appelant une fonction javascript "afficher" lors du click, cette fonction prenant le formulaire en argument  
`<input type="button" value="Ajouter" onClick="ajouter(this.form);"/>`
- pour accéder aux champs de ce formulaire dans cette fonction "afficher", il vous suffira de faire appel à l'attribut "value" de ses attributs portant le nom de l'id du champ  
*exemple : form.name.value si le champ est <input ... id="name" ... />*
- enfin, il vous faudra bien entendu un tableau HTML, avec `id="todoList"`, ce qui vous permettra, en JavaScript, d'y ajouter des informations à son contenu, accessible via :  
`document.getElementById('todoList').innerHTML`

# Manipulation de pages web



Document Object Model	38
l'objet document	39
Elements DOM	41
Événements	43

## 1. Document Object Model

### 1.1. Qu'est ce que c'est ?

#### *Problématique*

---

Bien entendu, tout langage destiné à dynamiser une page web, par essence statique, puisqu'en HTML, a besoin d'*accéder aux différents éléments de cette page*, afin d'en connaître les caractéristiques et de les manipuler. De fait, il paraîtrait stupidement long et fastidieux de devoir, pour chaque développeur, créer ses propres fonctions d'analyse et de manipulation de ce code. Quoi de plus logique, donc, de créer une interface standard pour tous, quelque soit le langage et le contexte ? Le premier obstacle à cela, pour le web, fut d'abord la divergence sur ce point des différents navigateur. Car même si, dès le départ, une certaine convergence a dû se faire autour du JavaScript, chaque navigateur disposait de son propre DOM, et continuait à intégrer quelques spécificités de manière différente, rendant le code créé pour l'un incompatible avec l'autre.

#### *Standardisation*

---

C'est donc là qu'entre en œuvre le *W3C*, afin de créer un *standard unique*, normalement intégré indifféremment par chaque navigateur. C'est cela qu'on appelle aujourd'hui le DOM : une *API standard, indépendante de tout langage et de toute plateforme*, permettant d'accéder et de modifier le contenu, la structure ou le style de documents XML et HTML. Le développement du DOM est aujourd'hui arrêté, la dernière (et définitive) version (DOM 3) ayant été publiée en 2004.

#### *DOM et JavaScript*

---

DOM permet donc, en JavaScript, d'accéder de manière standard à notre page HTML, grâce à différents objets, dont nous allons voir ici les noms, attributs et méthodes.

## 1.2. Principaux objets

### *window*

---

L'objet *window* représente la *fenêtre de navigation*, premier objet à utilisé, parent de nombreux autre. C'est un objet dit "*implicite*", c'est à dire n'ayant pas besoin d'être spécifié. Par exemple, la fonction *alert()*, permettant d'afficher une boîte de dialogue, n'est pas une fonction globale (comme *isNaN()* par exemple, au final très peu nombreuses), mais une méthode de l'objet *window* (*window.alert()* ni nous voulions l'écrire explicitement). Notez que, dans votre page, toute variable JavaScript n'ayant pas été déclarée sera par conséquent ajoutée comme attribut de *window*, et aura donc une portée globale.

### *document*

---

Sans doute l'objet que vous utiliserez le plus en JavaScript, puisqu'il s'agit de celui représentant la *page web* (plus précisément, la balise *<html>* ), et qui aura donc tout les attributs et fonctions permettant de manipuler chaque éléments de cette page. Par exemple, la méthode *document.getElementById(id)* permet de récupérer n'importe quel élément (balise) de votre page grâce à son id.

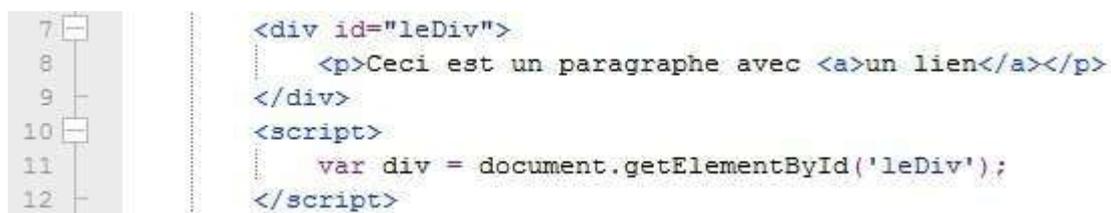
## 2. l'objet document

L'objet *document* représente donc notre page web. Voyons ici les différentes méthodes et attributs utiles.

### 2.1. Récupération d'éléments

#### *Par ID*

---



*DOM JS : récupération d'un élément par son ID*

L'objet *div* sera dès lors de type *HTMLDivElement*, et contiendra toutes les informations nécessaires pour la manipulation de notre élément *div*.

## Par nom de balise

```

7 | 
8 |
9 |
10| 
11|
12|
13| 
14|
15| 
16|
17|
18|

```

```

<div id="leDiv">
  <p>Ceci est un paragraphe avec <a>un lien</a></p>
</div>
<div id="autreDiv">
  Et un div avec du texte dedans.
</div>
<script>
  var divs = document.getElementsByTagName('div');
  for (var i = 0, c = divs.length ; i < c ; i++) {
    alert('Element numéro : ' + (i + 1) + ' : ' + divs[i]);
  }
</script>

```

DOM : Récupération d'élément par nom de balise

La méthode `document.getElementsByTagName(tag)` (remarquez au passage le *s* à *Elements*), renvoi un tableau, contenant tout les éléments correspondants au nom de balise donné. Ici, par exemple, ce tableau est constitué de deux éléments de type `HTMLDivElement`.

## Par nom

```

7 | 
8 |
9 |
10| 
11|
12|
13| 
14|
15| 
16|
17|
18|

```

```

<div id="leDiv">
  <p name="test">Ceci est un paragraphe avec <a>un lien</a></p>
</div>
<div id="autreDiv">
  Et un div avec du texte dedans.
</div>
<script>
  var elements = document.getElementsByName('test');
  for (var i = 0, c = elements.length ; i < c ; i++) {
    alert('Element numéro : ' + (i + 1) + ' : ' + elements[i]);
  }
</script>

```

DOM JS : Récupération d'éléments par nom

De la même manière que précédemment, nous pouvons récupérer tous les éléments (au pluriel encore une fois) dont l'attribut `name` est égal à l'argument donné. Ici, par exemple, nous récupérerons un tableau composé d'un seul élément de type `HTMLParagraphElement`.

## 2.2. Écriture dans le document

Même si rarement utilisées, nous disposons de deux méthodes pour écrire dans le document : `document.write('text')` et `document.writeln('text')`. Attention, cette méthode écrit dans le document directement à l'endroit où vous l'appellez, c'est à dire à l'endroit où a été placé votre balise `script`. `writeln`, en plus d'écrire le texte donné en argument, ajoutera un retour à la ligne `"\n"` après ce texte, mais attention, ce retour à la ligne ne permet pas d'afficher un retour à la ligne en HTML. Utilisez donc plutôt la balise `<br/>` avec `write`.

## 2.3. Propriétés

### *Quelques exemples notables*

---

Document étant un objet, celui-ci dispose bien entendu de nombreuses propriétés. En voici quelques exemples :

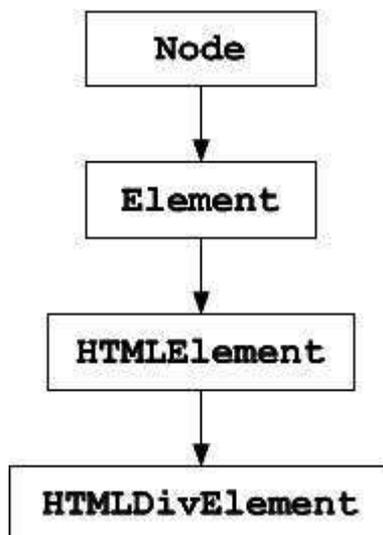
- alinkColor : couleur des liens
- bgColor : couleur de fond de page
- defaultCharset : jeu de caractère
- location (équivalent à URL) : url de la page
- referrer : adresse de la page d'origine

## 3. Elements DOM

### *Structure et héritage*

---

Grâce à la POO et l'héritage, DOM met en place une structure, vous permettant de retrouver certains attributs et méthodes sur tout un ensemble d'éléments, pourtant de type différent. Par exemple, la relation d'héritage partant de HTMLDivElement :



*DOM : parents de HTMLDivElement*

### 3.1. Node

Avec DOM, tout est un node :

- le document lui même
- les éléments
- les attributs
- le texte à l'intérieur de chaque élément
- les commentaires

Node (ou noeud en français) est donc l'objet générique duquel héritent tout les objets manipulables via le DOM. Vous retrouverez donc ses attributs et méthodes avec chacun des objets manipulés. Vous les retrouverez à l'adresse suivante :  
[http://www.w3schools.com/dom/dom\\_node.asp](http://www.w3schools.com/dom/dom_node.asp) et  
<https://developer.mozilla.org/en-US/docs/Web/API/Node>

## 3.2. Element

Le DOM HTML étant basé sur le DOM XML, un élément correspond à n'importe quelle balise de votre code. Par exemple, chaque balise `<a></a>` est un élément. Attention cependant, `<p>texte</p>`, par exemple, est bien un élément, mais son contenu est un tout autre objet (text). Gardez donc à l'esprit que lorsque vous récupérez un ou plusieurs éléments grâce aux méthodes de document, celui ci sera de type Element, avec donc des attributs et méthodes, commun quelque soit le type d'élément récupéré ... mais aussi d'un type "fil", comme par exemple HTMLDivElement, qui offrira d'autre attributs et méthodes spécifiques.

Pour la présentation complète de Element, voir :  
[http://www.w3schools.com/dom/dom\\_element.asp](http://www.w3schools.com/dom/dom_element.asp) et  
<https://developer.mozilla.org/en-US/docs/Web/API/Element>

### Récupérer la valeur d'un attribut

```
// attribute sera de type String, et contiendra la valeur de
cet attribut, ou null ou "" si l'attribut recherché n'est pas
présent
var attribute = element.getAttribute(attributeName);
```

## 3.3. HTMLÉlement

HTMLÉlement est l'élément spécifique du HTML, englobant tout les éléments, ce qui nous permet, encore une fois, d'ajouter des méthodes et attributs particulièrement adaptés à ce langage (car ne l'oubliez pas, DOM n'est pas fait que pour le HTML, mais également pour tout document XML). Bien entendu, il hérite de Element, précédemment vu.

### Éléments HTML spécifiques

A chaque balise HTML est associé une interface spécifique. Vous en retrouverez la liste à la rubrique [HTML Interfaces > HTML Element Interfaces](https://developer.mozilla.org/en-US/docs/DOM/DOM_Reference) :

Ceux-ci vous permettront généralement d'accéder directement à certains attributs spécifiques à cet élément, comme par exemple le href d'une balise a :

```
<a id="myLink" href="http://www.noelmace.com">Un lien</a>
```

```
var link = document.getElementById('myLink');
var href = link.href;
```



### Complément : Formulaires

Pour une liste complète des attributs et méthodes spécifiques aux éléments `<form>` et `<input>`, voir, respectivement :

- <https://developer.mozilla.org/en-US/docs/Web/API/HTMLFormElement>
- <https://developer.mozilla.org/en-US/docs/Web/API/HTMLInputElement>



### Complément : CSS

Chaque élément possède un attribut "style", possédant lui même les différents attributs du CSS (comme par exemple width). Vous pouvez donc utiliser cet attribut pour connaître et modifier les paramètres CSS d'un affichage. Pensez également qu'il vous est possible de modifier directement la classe d'un élément, grâce à son attribut className (ce qui sera bien plus pratique et fonctionnel dans la plupart des cas ... rappelez vous, toujours séparer la forme du font !).

## Contenu : *innerHTML*

---

L'attribut `innerHTML` de l'objet `HTMLInputElement` est particulièrement utile, car contenant tout le texte (sous forme de `String`) contenu au sein de la balise. Par exemple :

```
<p id="monPara">Du texte avec <a>un lien</a></p>
```

```
var para = document.getElementById('monPara');
alert(para.innerHTML); // affiche "Du texte avec <a>un
lien</a>"
```

## 4. Événements

### *Qu'est ce qu'un événement ?*

---

Le but principal du JavaScript est bien entendu de créer du contenu dynamique, donc particulièrement de réagir aux actions de l'utilisateur. Les événements vont donc ne permettre de réagir à ses actions, en appelant un code spécifique dès qu'un événement survient (par exemple, cliquer sur un élément)/

### 4.1. Les différents événements

#### *Liste (non exhaustive) des événements à connaître*

---

- `click` : Cliquer (appuyer puis relâcher) sur l'élément
- `dblclick` : Double-cliquer sur l'élément
- `mouseover` : Faire entrer le curseur sur l'élément
- `mouseout` : Faire sortir le curseur de l'élément
- `mousedown` : Appuyer (sans relâcher) sur le bouton gauche de la souris sur l'élément
- `mouseup` : Relâcher le bouton gauche de la souris sur l'élément
- `mousemove` : Faire déplacer le curseur sur l'élément
- `keydown` : Appuyer (sans relâcher) sur une touche de clavier sur l'élément
- `keyup` : Relâcher une touche de clavier sur l'élément
- `keypress` : Frapper (appuyer puis relâcher) une touche de clavier sur l'élément
- `focus` : « Cibler » l'élément
- `blur` : Annuler le « ciblage » de l'élément
- `change` : Changer la valeur d'un élément spécifique aux formulaires (`input`, `checkbox`, etc.)
- `select` : Sélectionner le contenu d'un champ de texte (`input`, `textarea`, etc.)

#### *Événements spécifiques à l'élément <form>*

---

- `submit` : Envoyer le formulaire
- `reset` : Réinitialiser le formulaire

### 4.2. Utiliser des événements sans DOM

#### *Limites*

---

Nous allons rapidement voir que l'usage des événements sans passer par le DOM est extrêmement limité. Cependant, cela sera utile pour quelques usages simples, ne nécessitant pas de développement poussé.

#### *Comment ?*

---

Pour ce faire, nous allons directement ajouter un attribut `"on"` + le nom de l'événement à gérer, à notre balise.





### Exemple : Evènements click et focus

```
<span onclick="alert('Hello !');">Coucou !</span>
<input id="untexte" type="text" value="Entrez du texte !"
onfocus="document.getElementById('unautre').value = 'Il faudra
aussi écrire quelque chose ici';"/>
<input id="unautre" type="text" value="" />
```

Quand l'utilisateur click sur le span, nous affichons une boîte de dialogue Hello !. Quand l'utilisateur "cible" le premier champ texte, c'est à dire qu'il a mis le focus sur cet élément (via un click ou la touche tab), à priori pour y écrire, nous modifions le contenu de l'input suivant.



### Complément : Bloquer l'action par défaut de l'élément

Il peut être utile, également, de bloquer l'action par défaut d'un élément. Par exemple, l'action par défaut d'un élément `<a></a>` pour un click est bien entendu de suivre le lien. Donc, un code comme le suivant fera deux actions : celle du onclick, puis l'appel du href :

```
<a href="http://www.noelmace.com" onclick="alert('Vous avez
cliqué !');">Cliquez-moi !</a>
```

Pour remédier à cela, on ajoute l'instruction "return false ;" à la fin de l'action associée à notre évènement, comme ceci :

```
<a href="http://www.noelmace.com" onclick="alert('Vous avez
cliqué !'); return false ;">Cliquez-moi !</a>
```

## 4.3. Gestion d'événements avec DOM2

### Le standard

La méthode suivante est la méthode standard, prescrite par le W3C, et parfaitement gérée par tout les bons navigateurs (c'est à dire tout le monde sauf IE8 et inférieur, nous le verrons plus loin). Il faut donc ici :

1. faire appel à la méthode `addEventListener` de votre élément
2. lui donner comme premier attribut le nom de l'évènement à gérer (click par exemple)
3. indiquer une fonction, qui sera déclenchée lors de l'évènement  
(vous pouvez parfaitement utiliser une fonction anonyme si vous ne la réutilisez pas ensuite)
4. indiquer un booléen, permettant de dire si nous souhaitons déclencher la phase de capture ou de bouillonnement. Nous expliquerons ce point plus loin. Pour l'instant, sachez que ce paramètre est généralement à false.

```
element.addEventListener('click', function() {
    alert("Vous m'avez cliqué !");
}, false);
```

Il est ainsi parfaitement possible de créer plusieurs évènements identiques pour un évènement (par exemple, deux évènements se déclenchant l'un après l'autre lors d'un click sur l'élément). Pour cela, il suffit de faire appel à cette méthode plusieurs fois de suite, avec des paramètres différents.

Enfin, vous pouvez supprimer la gestion de cet évènement en faisant appel à la méthode `removeEventListener` avec les même paramètres utilisés lors de l'ajout.

*Et ... internet explorer 8 et inférieur ...*

---

Et oui, IE8 (et inférieur) ne connaît pas ces deux méthodes ... mais les remplace par `attachEvent` et `detachEvent`, comme dans l'exemple suivant :

```
// On crée l'événement
element.attachEvent('onclick', function() {
    alert('Tadaaaam !');
});
// On supprime l'événement en lui repassant les mêmes
paramètres
element.detachEvent('onclick', function() {
    alert('Tadaaaam !');
});
```

Attention : remarquez que IE8 ne connaît pas "click", mais "onclick", comme quand nous n'utilisons pas DOM.



# Conclusion



Exercice : TP : formulaire interactif

46

## 1. Exercice : TP : formulaire interactif

Pour conclure cette introduction au JavaScript, quoi de mieux que de réaliser une petite page interactive complète, mêlant HTML, CSS et JavaScript.

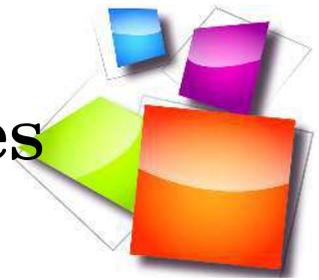
Question

[Solution p 51]

Pour ce faire, rendez vous sur *ce site* . Bien entendu, le but ici est d'apprendre (et on apprend souvent de ses erreurs). Evitez donc de consulter la solution.



# Solutions des exercices



## > Solution n° 1

Exercice p. 27

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4    <title>Hello World!</title>
5  </head>
6
7  <body>
8
9    <script>
10     var maVariable = 'val 1'
11     alert('maVariable is a ' + typeof maVariable);
12     maVariable = parseInt(prompt('Entrez un nombre :'));
13     alert('Vous avez entré : ' + maVariable);
14   </script>
15
16 </body>
17 </html>
18

```

*Solution Exercice Variables JS*

Exercice p. 29

## > Solution n° 3

Exercice p. 32

```

1  var nbr = parseInt(prompt('Entrez un nombre : '));
2  var fact = 1
3  for ( var i = 1 ; i <= nbr ; i++) {
4    fact *= i;
5  }
6  alert('la factorielle de ' + nbr + ' est : ' + fact);

```

*Calcul de factorielle JS*

Exercice p. 32



## &gt; Solution n° 4

```

1 var premier = parseInt(prompt('entrez un nombre'));
2 var second;
3 do {
4     second = parseInt(prompt('entrez un nombre plus petit :'));
5 } while (! (second < premier));
6 alert(second + ' est bien plus petit que ' + premier);

```

*Plus Petit JS*

## &gt; Solution n° 5

*Exercice p. 33*Exemple de correction

```

1 function premier(arg){
2     // conversion de nbr en nombre entier
3     nbr = parseInt(arg);
4     // on teste si nbr est bien un nombre entier
5     if (!isNaN(nbr) && (parseFloat(arg) == parseInt(arg))) {
6         // si nbr est négatif, on calculera si sa valeur absolue est première
7         if (nbr < 0) {
8             nbr = -nbr;
9         }
10        // tester le reste de la division de nbr par tout nombre entre 2 et nbr-1
11        for (var i = 2; i < nbr; i++) {
12            if (nbr%i == 0) {
13                // si nbr est divisible par un autre nombre que 1 ou nbr, c'est qu'il n'est pas premier
14                return false;
15            }
16        }
17        return true;
18    } else {
19        /* si nbr n'est pas un entier, nous renvoyons NaN
20         * c'est à dire une valeur de type number qui indique ne pas être un nombre correct
21         * et sera considérée comme "false" dans un test booléen
22         */
23        return NaN;
24    }
25 }
26
27 // premier test
28 alert(premier(234));
29
30 // test interactif
31 var nombre = prompt('Entrez un nombre : ');
32 var isPremier = premier(nombre);
33 if (isPremier) {
34     alert(nombre + ' est premier !');
35 } else if (isNaN(isPremier)){
36     alert(nombre + " n'est pas un nombre correct !");
37 } else {
38     alert(nombre + " n'est pas un entier !");
39 }

```

*Nombre premier JS**Exercice p. 37*

## &gt; Solution n° 6

*Code HTML et JavaScript*

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="utf-8" />
5 <title>TF : Todo List</title>
6 <link rel="stylesheet" type="text/css" href="todo.css">
7 </head>
8
9 <body>
10
11 <form id="todoForm">
12
13 <label class="form_col" for="name">Nom :</label>
14 <input name="name" id="name" type="text" />
15 <br /><br />
16
17 <label class="form_col" for="priority">Date :</label>
18 <input name="date" id="date" type="date" />
19 <br /><br />
20
21 <span class="form_col"></span>
22 <input type="button" value="Ajouter" onClick="ajouter(this.form);"/>
23
24 </form>
25 <br/><br/>
26
27 <table id="todoList">
28 <caption>TODO List</caption>
29 <tr><th>Nom</th><th>Date</th></tr>
30 </table>
31
32 <script src="todo.js"></script>
33 <script>
34
35 Todo.prototype.afficher = function() {
36     document.getElementById('todoList').innerHTML += "<tr><td>" + this.name + "</td><td>" + this.date + "</td></tr>";
37 }
38
39 function ajouter(form) {
40     var todo = new Todo(form.name.value, form.date.value);
41     todo.afficher();
42 }
43 </script>
44
45 </body>
46 </html>

```

*Todo List JS : HTML*

## Code CSS

```

1  body {
2    padding-top: 50px;
3  }
4
5  .form_col {
6    display: inline-block;
7    margin-right: 15px;
8    padding: 3px 0px;
9    width: 200px;
10   min-height: 1px;
11   text-align: right;
12 }
13
14 input {
15   padding: 2px;
16   border: 1px solid #CCC;
17   -moz-border-radius: 2px;
18   -webkit-border-radius: 2px;
19   border-radius: 2px;
20   outline: none;
21 }
22
23 table {
24   padding: 2px;
25   border-radius: 2px;
26   width: 800px;
27   border: 1px solid black;
28 }
29 caption {
30   text-align: center;
31   font-weight: bold;
32   color: red;
33 }
34 th {
35   background-color: gray;
36   color: white ;
37   text-align: center;
38   border-radius: 2px;
39   border: 1px solid black;
40 }
41 td {
42   text-align: center;
43 }
44
45 input:focus {
46   border-color: rgba(82, 168, 236, 0.75);
47   -moz-box-shadow: 0 0 8px rgba(82, 168, 236, 0.5);
48   -webkit-box-shadow: 0 0 8px rgba(82, 168, 236, 0.5);
49   box-shadow: 0 0 8px rgba(82, 168, 236, 0.5);
50 }

```

*TODO List JS : CSS*



## Constructeur des objets Todo en JavaScript

---

```
22 function Todo(name, date){  
23     this.name = name;  
24     this.date = date;  
25 }
```

*TODO List JS : définition du constructeur*

### > Solution n° 7

*Exercice p. 46*

Vous trouverez un exemple de correction *ici* .

# Glossaire



## portabilité

« Capacité d'un programme informatique à fonctionner sous différents environnements d'exécution, systèmes d'exploitation. » - [Wikitionnaire](#)

## taxinomie

A l'origine science de classement des organismes vivants (en tant que branche de la biologie), la taxinomie s'étend aujourd'hui à d'autres sciences, et notamment l'informatique. Elle consiste donc, de manière générale, à classifier et hiérarchiser un ensemble d'entités (taxons) selon des caractères communs, des plus généraux aux plus particuliers.



# Abréviations



CSP

**CSP** : Constraint Satisfaction Problem - Problème de satisfaction de contraintes

POO

**POO** : Programmation Orientée Objet

PSF

**PSF** : Python Software Foundation

VIM

**VIM** : Vi (Visual) IMproved

# Références



## *TIOBE*

Société fournissant un [index](#) de popularité des différents langages de programmation du marché, déterminé en fonction du nombre d'ingénieurs compétents, cours disponibles et revendeurs retrouvés grâce aux moteurs de recherche les plus populaires. Il s'agit à ce jour du meilleur indice existant de la popularité d'un langage.

## *VIM*

[Site officiel](#)  
[Répertoire des extensions](#)  
[Communauté francophone](#)



# Crédits des ressources



## **Architecture 3-Tiers**

*<http://creativecommons.org/licenses/by-sa/2.0/fr/>, *wikimedia*  
p. 12*

## **Brendan Eich**

*<http://creativecommons.org/licenses/by-sa/2.0/fr/>, *wikimedia*  
p. 14*

## **DOM : parents de HTMLDivElement**

*<http://creativecommons.org/licenses/by-nc-sa/2.0/fr/>, *openclassrooms.com*  
p. 41*