Introduction

I.1. Motivation

Deux raisons principales sont à l'origine de la rédaction de cet ouvrage :

– l'intérêt pour le langage JavaScript : le plus utilisé sur la planète. Il est exécuté des centaines de millions de fois par jour : quelle page web ne l'utilise-t-elle pas ?

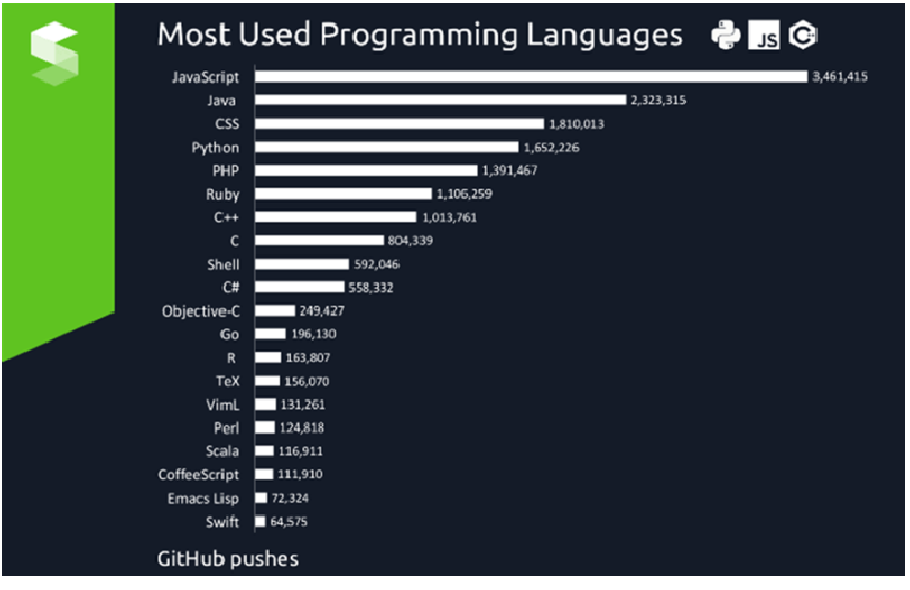

Figure I.1. *Contributions de code à différents langages sur GitHub*

Cet usage massif[1] lui assure maintenance et évolution permanente depuis plus de 25 ans. Dans les années récentes, plusieurs innovations de la norme ont convaincu la totalité des fournisseurs de navigateur. Le schéma suivant, publié le 17 octobre 2017, montre les contributions des internautes au développement de code en JavaScript ;

– l'accès libre aux données de l'Internet : tout citoyen a droit à des outils de lecture. Au-delà des données « propriétaires » à accès restreint, les données publiques ouvertes (Open Data par exemple : Nations unies, INSEE, US Census Bureau, etc.), ou en accès gratuit *via* des services privés (par exemple : Google drive partagé, Wikipedia), sont un énorme réservoir d'informations universelles, patrimoine commun de l'humanité.

Ces raisons nous ont incitées à proposer cet ouvrage sous forme d'un manuel de programmation de JavaScript, orienté vers l'accès aux données ouvertes, au croisement de données multiples pour leur traitement et leur affichage sur le navigateur de chacun. Depuis 2015, la large adoption des normes du langage encourage l'usage des outils les plus récents présentés dans ce manuel.

Tout citoyen qui a de la curiosité vis-à-vis des données et du goût pour l'informatique peut devenir un « data scientist » amateur et éclairé, capable d'étudier directement les informations et chiffres de ses domaines d'intérêt. Un lecteur professionnel, un étudiant en études politiques ou journalisme[2], un responsable d'association, etc. trouvera dans ce manuel des outils appropriés.

I.2. Plan de l'ouvrage

Un préambule complète cette introduction : historique du langage, démystification des préjugés, liste des prérequis et outils annexes, liste des fonctionnalités principales de JavaScript pour introduire ce que les parties suivantes vont détailler.

La partie 1 présente les bases du langage : variables, instructions, tests, objets natifs et objets de l'application, le traitement des chaînes de caractères, des tableaux, des objets et des fonctions. Nous verrons les aspects spécifiques de JavaScript, son originalité dans le monde des langages à objet et sa capacité à répondre à la plupart des tâches de traitement des données que nous mettons en avant dans cet ouvrage. Nous terminons par quelques exemples de programmation par « patterns ».

1. Figure I.1 issue de https://stackify.com/trendiest-programming-languages-hottest-sought-programming-languages-2017.
2. Le NICAR, *National Institute for Computer-Assisted Reporting* maintient des bases de données fédérales US et forme les journalistes aux pratiques d'accès et analyse des données ouvertes. http://ire.org/nicar/ et « IRE Journal », le magazine de *Investigative Reporters and Editors, Inc*.

La partie 2 présente le JavaScript dans « l'écosystème de la page web » composé du protocole HTTP, du code HTML, des règles CSS et du langage de script JavaScript. L'interface avec les éléments et événements du HTML DOM (« document object model ») permet l'enrichissement dynamique de la page « côté client ». La technologie Ajax permet l'ajout d'informations extraites du « grand nuage » des données de l'Internet. Nous aborderons les questions de traitement asynchrone des données.

La partie 3 est dédiée au déploiement d'applications :

– accès aux données ouvertes, libres ou gratuites, combinaison des informations issues de sources de données multiples et leur « jointure » asynchrone ;

– affichage de données numériques en tracés graphiques, animation de données vectorielles, représentation cartographique.

Nous présenterons quelques prétraitements utiles pour la création de fichiers JSON issus de logiciels de tableurs, sur l'usage des Google *spreadsheets*, sur JSONP, autant d'outils permettant, à défaut de données directement accessibles *via* des « API », de convertir de nombreuses données de l'Internet en données que vous pourrez utiliser dans vos applications.

I.3. Préambule : historique du langage

Cette note historique sur la naissance et la carrière de JavaScript est une autre raison d'intérêt pour ce langage : à l'échelle de l'informatique, c'est un langage « vieux » : plus de 25 ans. Comment expliquer cette longévité ? Quels atouts significatifs ont permis sa pérennité ?

– 1993 : apparition du navigateur web Mosaic (qui a rendu le World Wide Web populaire), distribué par un centre de recherche américain le NCSA (National Center for Supercomputing Applications).

– 1995 : Mosaic devient Netscape (alors 90 % du marché), qui charge Brendan Eich de créer un langage de script sur le modèle de Java distribué quelques mois avant par Sun, pour son Navigator. En deux semaines, le travail est fait, inspiré du langage Self, créé à Xerox PARC, basé sur les « prototypes » plutôt que sur les « classes » de Java (question de délai).

– 1996 : Netscape soumet JavaScript à l'organisme de standardisation ECMA. Microsoft réagit en développant JScript pour Internet Explorer (version 3).

– 2006 : spécification de l'objet XMLHttpRequest par W3C pour « normer » le déploiement de la technologie Ajax à partir du JavaScript.

– 2008 : première version de V8, moteur JavaScript de Chrome.

– 2009 : ECMAScript Edition 5 (ES5), première version à être intégrée dans tous les navigateurs.

– 2009 : Node.js (Ryan Dahl) : JavaScript s'installe vraiment côté serveur.

– 2010 : version V8 avec compilation optimisée (*crankshaft*) : course à l'optimisation de JavaScript entre tous les navigateurs.

– 2015 : ECMAScript 2015, *alias* ES6, apporte des nouveautés importantes : les déclarations `let et const`, la méthode `Object.assign`, etc. Cette version est supportée par tous les navigateurs récents.

Année	Nom /alias	Description
1997	ECMAScript 1	First Edition
1998	ECMAScript 2	Editorial changes only
1999	ECMAScript 3 / ES3	Added Regular Expressions Added try/catch
	ECMAScript 4	*(never released)*
2009	ECMAScript 5 / ES5	Added JSON support
2015	ECMAScript 6 / ES6 / ECMAScript2015	Added let, const, Object.assign Added classes and modules
2016	ECMAScript 7	Added exponential operator (**) Added Array.prototype.includes

Tableau I.1. *Historique des versions de JavaScript*

I.3.1. *Analyse de la biographie de JavaScript*

On note des périodes très distinctes :

– succès initial : qui dénote l'intérêt perçu pour l'enrichissement interactif et dynamique de la page web (on parlait de DHTML à l'époque) : période 1997-1999 ;

– décade de stagnation : des versions différentes se développent sur les navigateurs, les tentatives de JavaScript côté serveur connaissent un échec. Seul le développement de jQuery, en offrant une couche unique d'accès à JavaScript, permet la survie du langage ;

– l'apparition de la technologie Ajax, puis de la compilation rapide V8 et la version « hors web » de Node.js réveillent la normalisation : introduction de l'objet JSON en 2009 ;

– JavaScript devient un langage généraliste avec ES5, largement adopté par tous les navigateurs (Microsoft « dépasse » la non compatibilité d'Internet Explorer en créant Edge). L'optimisation des interprètes en fait un langage efficace. La normalisation ES6 permet de surmonter la plupart des obstacles liés au design initial.

I.4. Préambule : programmer sans « var », sans « for », sans « new »

JavaScript est un langage bien vivant qui s'est adapté aux évolutions de l'Internet et de ses nouveaux usages (vidéo, réseaux sociaux, publicité ciblée, etc.). De plus en plus d'applications web utilisent massivement JavaScript côté client, ou côté serveur depuis l'apparition de Node.js et du moteur V8.

I.4.1. *Analyse*

Les fondements du langage restent les mêmes, mais deux évolutions majeures modifient grandement la manière de coder en JavaScript aujourd'hui :

– les usages majoritaires ne sont plus les mêmes : le traitement interactif de la page web est devenu anecdotique, le traitement des données issues de requêtes Ajax est très important ;

– les introductions récentes (ES5 et ES6) dans le « CoreECMAScript » permettent de mieux tirer avantage de l'approche prototypale et de la nature fonctionnelle du langage.

I.4.2. *Parti pris*

Par conséquent, on n'écrit plus du JavaScript en 2018 comme on le codait avant 2015. La norme ES6 offre les outils qui permettent d'exprimer les qualités de JavaScript plutôt que ses défauts :

– langage à base de prototypes : utilisez les prototypes, programmez sans opérateur `new` ;

– les fonctions sont des objets de première classe : programmez fonctionnellement, programmez sans boucle `for` ;

– utilisez des déclarations mieux « contrôlées », n'utilisez pas `var`.

Le code produit avec ces principes est à la fois plus concis, mieux lisible et plus facile à déboguer.

I.4.3. *Prérequis pour la lecture*

Le *Big Data* ne doit pas être la chasse gardée des « big actors » : un ordinateur, un navigateur, une liaison Internet, un peu de travail personnel permettent d'ouvrir la porte du Big Data. Cet ouvrage suppose toutefois de posséder quelques notions de base :

– sur l'Internet et le World Wide Web (WWW) ;

– sur le HyperText Markup Language (HTML) et un minimum sur les Cascading Style Sheets (CSS) ;

– sur les outils « développeur » de votre navigateur.

I.4.4. *Outils d'aide à la programmation en JavaScript*

Le navigateur sait interpréter ce langage, ce sera donc votre outil d'expérimentation, même sans connexion Internet :

– écriture directe d'une page web, qui affiche les résultats de l'exécution d'un code JavaScript (voir la partie 2) : utilisez la « Web Console » ou la « Browser Console » de votre navigateur ;

– le « ScratchPad » de votre navigateur, testé avec Firefox, permet de tester rapidement une ligne (ou quelques lignes) de code avec un affichage direct des valeurs en commentaire.

De plus, il existe de nombreux outils « en ligne » pour vous aider dans votre apprentissage :

– W3Schools ; permet de tester rapidement du code HTML + JavaScript, un tutoriel est présent sur le même site ;

– JSBin et JSFiddle, populaires parmi les développeurs, proposent un contexte semblable. On peut les utiliser pour archiver ;

– Thimble de Mozilla : environnement qui permet de tester des projets de taille plus importante et de travailler de manière collaborative ;

– JSLint réalise une analyse lexicale du JavaScript en ligne et suggère des corrections.

I.5. Préambule : fonctionnement et fonctionnalités

I.5.1. *JavaScript : un langage de script à l'intérieur d'un écosystème*

JavaScript est un langage de script. Son interprétation et son exécution sont réalisées par un moteur de script qui a besoin d'un environnement hôte dont il utilise les objets, événements et ressources.

Il faut distinguer le « Core JavaScript », norme commune du langage et le JavaScript hébergé (« embedded ») qui intègre les objets spécifiques à l'environnement (par exemple : « Client-Side JavaScript »).

Dans l'écosystème de la page web, cet environnement est le navigateur (l'objet 'window'). Il y a aussi les « Workers » fournis par le navigateur, mais indépendants de la page web, et l'écosystème des modules Node.js qui fonctionne hors du navigateur (par exemple dans un serveur).

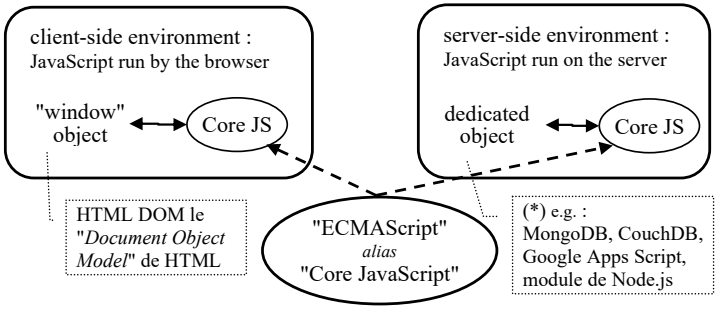

Figure I.2. *JavaScript a besoin d'un environnement*

I.5.2. *Fonctionnement du moteur JavaScript*

Le moteur de script analyse puis exécute le code JavaScript dans l'environnement hôte dont il utilise les objets, événements et ressources : ce qu'on nomme « l'objet global ». Dès que le moteur de script démarre dans cet environnement, il exécute deux phases successives :

– analyse lexicale (« Read-time » ou « Lexical-time ») ;

– suivie d'exécution directe ou compilation d'un code adapté à l'ordinateur hôte (« Run time »). Le moteur de script V8 exécute même deux étapes : compilation préalable (*full-codegen*), puis une compilation optimisée en temps-réel (*crankshaft*) pour améliorer l'efficacité du traitement.

Du point de vue du programmeur, cela signifie que :

– les déclarations lexicales sont effectuées en premier, avec une définition implicite à la valeur `undefined` et au type `undefined`. On peut s'en convaincre avec les lignes suivantes :

```
let x ; // declaration sans definition explicite
console.log( x );
console.log( typeof x );
```

– les assignations sont traitées ensuite : définition et typage des variables, expressions de fonctions, expressions régulières, etc. tout ce qui apparaît en partie droite d'un signe =.

I.5.3. *Variables et instructions : fonctionnalité des langages impératifs*

L'analyse lexicale détermine les variables et leur « portée ». L'exécution définit les variables (assigne une valeur définie) ce qui en détermine le type : celui de la valeur. On parle de « typage dynamique », terme plus approprié que « typage faible ».

Les valeurs peuvent être des valeurs primitives (nombres, chaînes de caractères), ou des expressions évaluées, ou des références à des objets, des tableaux, des fonctions, des expressions régulières, etc.

Les instructions peuvent être des assignations, des appels de fonctions, des structures de contrôle classiques comme des boucles ou des instructions conditionnelles.

I.5.4. *Langage à objets « basé prototype » (« prototype based object-oriented language »)*

Les objets peuvent être « natifs » (`Object`, `Function`, `Array`, `Math`, `JSON`, etc.), ou fournis par l'écosystème hôte comme les éléments du DOM (document HTML) ou construits par l'application.

JavaScript ne fait pas de distinction entre « classe » et « instance » : il n'y a que des objets et tout objet peut servir de « prototype » pour la création d'un autre objet. On peut résumer en disant qu'un objet c'est un ensemble de propriétés plus un prototype. Par défaut le prototype d'un objet est l'objet natif `Object.prototype`.

I.5.5. *Les fonctions comme « objets de première classe » : fonctionnalité des langages fonctionnels*

Les fonctions et les objets sont traités de la même manière. Seule une fonction peut être « appelée », mais toute opération faite sur un objet peut être faite sur une fonction, ce qui permet de construire des fonctions d'ordre supérieur (« higher order function »), c'est la nature fonctionnelle de JavaScript :

```
const mult = function(f,g) {
                        return function(n) {
                                return f(n) * g(n) ; } },
square = function(x) { return x * x ; };
mult(square, square)(3);        // -> 81
```

I.6. Conclusion

JavaScript est né pour le Web et le Web lui permet de vivre depuis bientôt 25 ans, c'est la raison qui nous pousse à l'étudier. Ce petit langage, créé en 15 jours par une seule personne est étonnamment flexible. Cette flexibilité est à la fois un avantage qui a permis à la norme d'évoluer sans toucher à ses bases et un inconvénient qui autorise trop d'erreurs de programmation difficiles à identifier et qui lui a valu le qualificatif de « langage le plus mal compris » de la part d'un des principaux « gourous », Doug Crockford. Trop grande « tolérance » autorisée, faible contrôle des types, existence d'objets sans « classe », syntaxe qui présente toutes les caractéristiques d'un langage impératif et procédural, sont des « pièges » déroutants. En revanche, le choix initial des fonctions comme « objets de première classe » lui confère des capacités fonctionnelles d'une rare puissance, souvent négligée.

Les efforts de normalisation récents permettent de s'affranchir des principaux pièges : on peut aujourd'hui programmer sans déclaration « var », sans boucle « for » et sans opérateur « new » pour écrire un code beaucoup plus stable et lisible.

L'étonnant ouvrage *Si Hemingway parlait JavaScript* démontre comment rendre méconnaissable de plusieurs manières différentes un code JavaScript qui pourtant réalise toujours la même opération.