**Introduction à la programmation**

Un programme informatique (également appelé application ou logiciel) est **une liste d'ordres indiquant à un ordinateur ce qu'il doit faire** (Wikipedia).

L'ensemble des fichiers contenant les instructions du programme constitue son **code source**. Programmer, c'est donc écrire le code source d'un programme, d'où l'emploi du terme synonyme de **coder**.

Pour être compris par un ordinateur, un programme doit respecter les règles du langage de programmation utilisé.

Un langage de programmation définit une manière de donner des ordres à un ordinateur. Un peu comme une langue vivante, tout langage a son vocabulaire (un ensemble de mots- clés, chacun jouant un rôle spécifique) et sa grammaire (un ensemble de règles définissant la manière d'écrire des programmes dans ce langage).

Le seul langage de programmation directement compréhensible par un ordinateur est le langage machine, également appelé **assembleur** (Wikipedia).

**L'exécution d'un programme**

On nomme exécution le fait de demander à un ordinateur de réaliser les ordres contenus dans un programme. Quel que soit le langage avec lequel il est écrit, un programme doit être traduit en assembleur pour pouvoir être exécuté. Ce processus de traduction dépend du langage choisi.

Avec certains langages, les lignes du code source sont traduites en assembleur puis exécutées ligne après ligne par un programme spécifique appelé interpréteur. On dit alors que le langage est interprété. Python et PHP sont des exemples de langages interprétés.

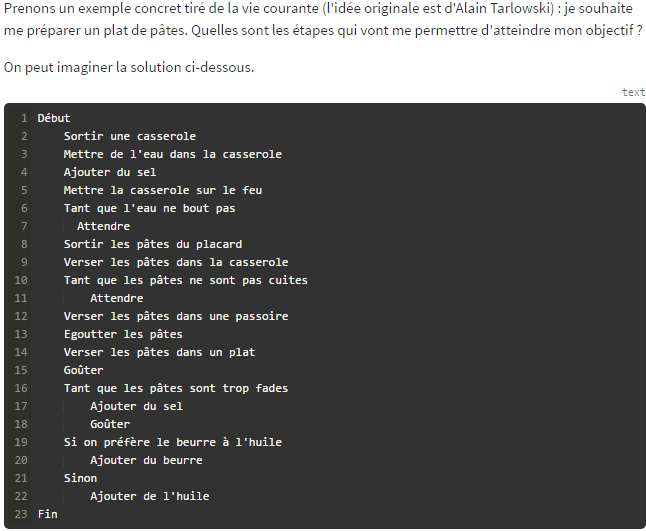
Une autre possibilité consiste à créer à partir de l'ensemble du code source un fichier directement exécutable (sous Windows, il portera l'extension .exe) en utilisant un programme intermédiaire appelé compilateur. On parle alors de langage compilé. Les langages C et C++ sont des exemples de langages compilés.

Enfin, une troisième option consiste à utiliser un pseudo-compilateur pour générer à partir du code  
source un ensemble de fichiers pouvant être exécutés sur n'importe quelle plate-forme supportant

l'environnement. C'est le cas du langage Java et des langages de la plate-forme Microsoft .NET  
(VB.NET, C#, etc).

Introduction aux algorithmes

On peut définir un algorithme comme **une suite ordonnée d'opérations permettant de résoudre un problème donné**. Un algorithme décompose un problème complexe en une suite d'opérations simples.



**Le rôle du programmeur**

Ecrire des programmes qui réalisent de manière fiable les tâches attendues est la première mission du programmeur. Un débutant arrivera vite à créer des programmes simples. La difficulté apparaît lorsque que le programme évolue et se complexifie. Il faut de l'expérience et beaucoup de pratique avant d'arriver à maîtriser cette complexité. C'est aussi ce qui fait de la programmation un art subtil et stimulant. Une fois les bases acquises, vos seules limites seront celles de votre imagination !

Quel langage pour débuter ?

Javascript

JavaScript est avant tout le langage de programmation du Web. Il a été inventé en 1995 par Brendan Eich, qui travaillait à l'époque pour la société Netscape, créatrice du premier navigateur Web populaire (l'ancêtre de Firefox).

JavaScript est partout. Sa connaissance vous ouvrira les portes de la programmation côté navigateur Web (on parle de développement *front-end*), côté serveur (*back-end*) ou côté mobile. Plutôt pas mal pour un langage qui se veut malgré tout simple et facile d'accès.

**Coder avec JavaScript en ligne**

Une solution simple et très rapide pour coder avec JavaScript consiste à utiliser un "bac à sable" en ligne. Il s'agit de sites Web qui permettent d'écrire du code HTML/CSS/JavaScript et d'observer immédiatement le résultat. Les plus connus sont [JSFiddle](https://jsfiddle.net/), [JS Bin](http://jsbin.com/) et [CodePen](https://codepen.io/).

**Coder avec JavaScript sur sa machine**

Avec Firefox par exemple:

Comme tous les navigateurs Web, Firefox est capable d'exécuter du code JavaScript. Nous allons l'utiliser pour tester nos programmes écrits en JavaScript. Il dispose d'outils dédiés au développement Web dont nous allons faire un usage intensif dans ce cours. Affichez ces outils en sélectionnant **Développement Web** puis **Outils de développement** dans le menu **Outils** de Firefox.

**Choisir un éditeur de code**

Comme nous l'avons vu, programmer consiste essentiellement à écrire des lignes de code dans des fichiers. N'importe quel logiciel capable d'éditer du texte peut être utilisé pour créer des programmes. Cependant, il existe des logiciels spécialisés dans l'édition de code qui apportent au programmeur une aide précieuse. Certains d'entre eux offrent de nombreux services pour écrire, tester et publier ses programmes. On les appelle des environnement de développement intégrés, ou encore [IDE](https://fr.wikipedia.org/wiki/Environnement_de_d%C3%A9veloppement) en anglais. Parmi les IDE les plus connus, citons [Eclipse](https://eclipse.org/), [WebStorm](https://www.jetbrains.com/webstorm/) ou encore [Visual Studio](https://www.visualstudio.com/).

Pour ce cours, nous allons nous contenter d'un "simple" éditeur de code, plus facile à prendre en main. Parmi les éditeurs les plus populaires à l'heure actuelle, on trouve [Sublime Text](https://www.sublimetext.com/), [Atom](https://atom.io/) et [Brackets](http://brackets.io/). C'est ce dernier que nous allons utiliser.

HTML n'est pas un langage de programmation mais un langage de description de contenu. Habituellement, une page HTML contient de nombreuses balises qui permettent d'afficher des informations de manière structurée, avec des titres, des images, des liens vers d'autres pages, etc.

**Valeurs et Types**

Une **valeur** est un morceau d'information utilisé dans un programme informatique Le **type** d'une valeur détermine son rôle et les opérations qui lui sont applicables.

**Le type nombre**

Une valeur de type nombre (number) représente une valeur numérique

Création d’un objet en Javascript

Il y a 2 manières :

1° De manière littérale

A) Sans méthode

var personnage = {

nom :"Aurora",

sante : 150,

force : 25

};

B) Avec ajout d’une méthode:

var personnage = {

nom :"Qeyss",

sante : 200,

force : 50,

identification: function() {

return this.nom + " a " + this.sante + " points de santé."

}

};

console.log(personnage.identification());

C) Création d'un objet sans aucune propriété, puis ajout des propriétés et d’une méthode :

var perso = {};

perso.nom = "Aurora";

perso.sante = 150;

perso.force = 25;

// Renvoie la description du personnage

perso.decrire = function () {

var description = this.nom + " a " + this.sante + " points de vie et " +

this.force + " en force";

return description;

};

2° Via un constructeur

function Personnage (nom,sante,force) {

this.nom = nom;

this.sante = sante;

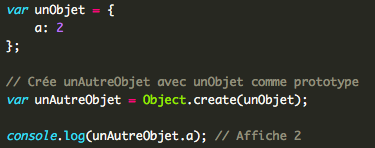
this.force = force;

};

var perso = new Personnage("Bilal", 150, 25);

**Objets et prototypes en JavaScript**

Pour créer des modèles d'objet en JavaScript, on utilise les prototypes, en Java on parle de classes.

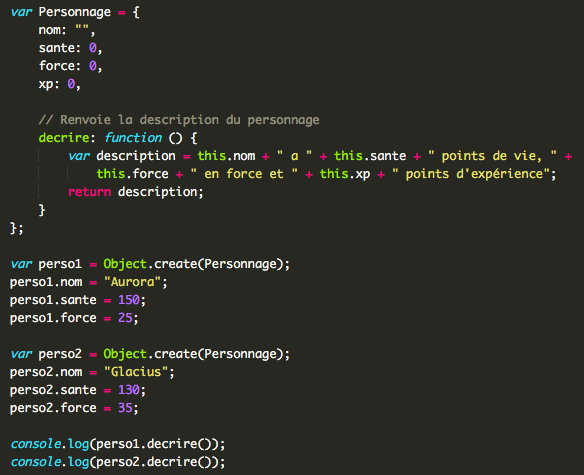
Lorsqu'on essaie d'accéder à une propriété qui n'existe pas dans un objet, JavaScript essaie de trouver cette propriété dans le prototype de cet objet.

Dans cet exemple, l'instruction JavaScript Object.create() est utilisée pour créer l’objet unAutreObjet en lui donnant comme prototype l’objet unObjet. Lors de l'appel à unAutreObjet.a, c'est la propriété a de unObjet qui est utilisée puisque la propriété a n'existe pas dans unAutreObjet.

Si la propriété n'a été trouvée dans aucun objet, son accès renvoie la valeur undefined

Ce mode de relation entre les objets JavaScript est appelé **délégation** : un objet délègue une partie de son fonctionnement à son prototype.

On parle d’héritage dans d’autres langages.

**Un prototype pour nos personnages**

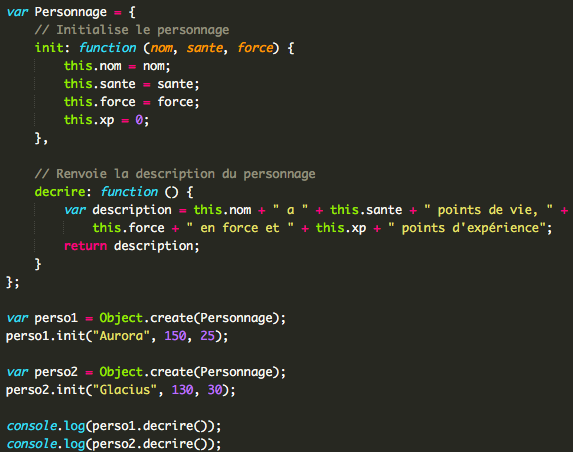
**Résultat :**

****

Nous avons créé un objet nommé Personnage, qui rassemble les propriétés communes à tous les personnages. Les objets perso1 et perso2 sont créés avec l’objet Personnage comme prototype, et lui délèguent les fonctionnalités communes.

Par convention, le nom d'un objet jouant le rôle de modèle (ici l’objet Personnage) commence souvent par une majuscule, mais ce n'est pas une obligation.

**Initialisation des personnages**

Pour chaque personnage créé il faut lui donner des valeurs à ses propriétés, le mieux dont est de créer une fonction d’initialisation dans le prototype Personnage :

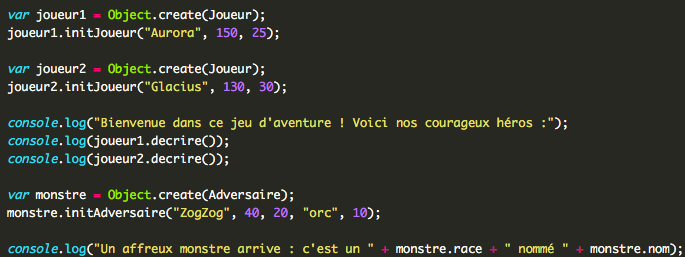
**Des adversaires pour nos héros**

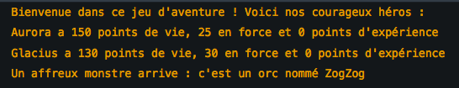
On va donc créer des joueurs et des adversaires qui seront tous deux des personnages, avec des points communs et des spécificités qui les distinguent.

Tout comme un joueur,  un adversaire simulé par l'ordinateur aura un nom, des points de vie et de force. En revanche il ne gagnera pas de points d'expérience, mais possèdera deux caractéristiques particulières : sa race et le nombre de points d'expérience gagnés lorsqu'il sera tué par un joueur.

Nous créons d'abord un objet Personnage qui est le modèle commun à tous les personnages. Il possède les propriétés communes à tous les personnages (nom, santé, force) ainsi qu'une méthode pour les initialiser.

Les objets Joueur et Adversaire sont tous deux créés avec Personnage comme prototype. Ils disposent chacun d'une fonction d'initialisation particulière, qui fait appel par délégation à la méthode initPerso()  de l’objet Personnage. Enfin, l’objet Joueur possède une fonction de description.

Une fois ces objets modèles définis, nous pouvons les utiliser pour créer nos personnages : les joueurs Aurora et Glacius (avec Joueur comme prototype), ainsi que le vilain monstre ZogZog (avec Adversaire comme prototype).



**3, 2, 1... Fight !**

Ajoutons à notre mini-jeu la possibilité pour les joueurs et les adversaires de s’affronter.

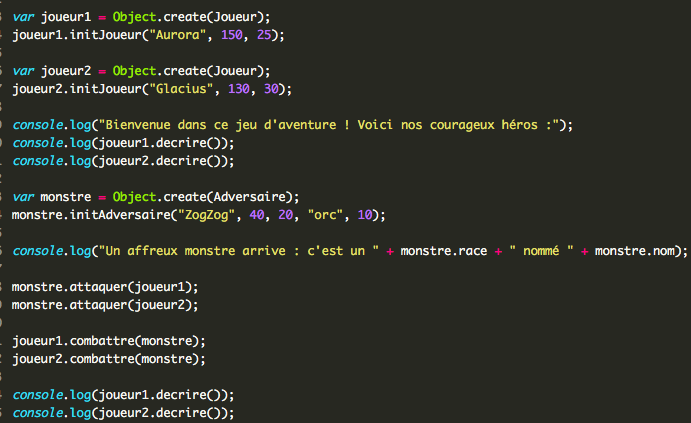
Règles du combat :

Un joueur peut attaquer un adversaire, mais l'inverse est aussi vrai. Un personnage attaqué voit ses points de vie diminués de la valeur de force de l'attaquant. Si ce nombre de points de vie tombe à zéro, alors le personnage meurt. Si son vainqueur est un joueur, il reçoit un nombre de points d'expérience égal à la valeur de l'adversaire tué.

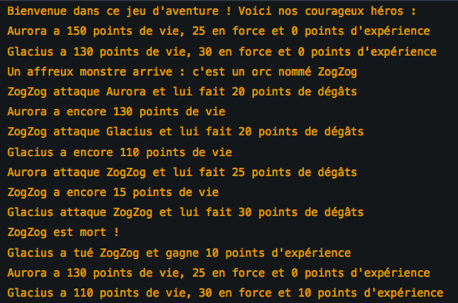


On ajoute à l’objet Joueur une méthode combattre() qui fait appel par délégation à la méthode attaquer() de Personnage et gère le gain d'expérience si l'adversaire meurt durant l’attaque.

L’objet Adversaire n'est pas modifié, mais peut malgré tout attaquer un jouer grâce à la méthode attaquer() de Personnage, dont il bénéficie par délégation( héritage).

Fight :

Résultat :



**Conclusion**

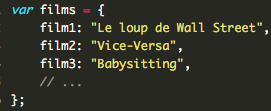
La programmation orientée objet consiste à écrire des programmes en utilisant des objets. La POO permet de rassembler des données et des comportements (les **méthodes**) dans des entités appelées des objets.

Le modèle objet de JavaScript se base sur des **prototypes** ( classes) pour créer des modèles et partager des propriétés entre plusieurs objets. Chaque objet a un prototype et une propriété absente d'un objet sera recherchée dans la chaîne de ses prototypes.

**Différence entre procédurale et POO**

Si je souhaite informatiser la liste de tous les films que j’ai vus cette année.

Une première solution serait de créer une variable par film, comme dans l'exemple suivant :

La liste risque de devenir très longue avec beaucoup de variables. En POO on pourrait tout mettre dans un objet :

Mais la meilleure solution est d’utiliser un tableau…

**Stockez vos données dans des tableaux**

**Introduction : le rôle des tableaux**

Un **tableau** est un type de donnée qui permet de stocker un ensemble d’éléments.

D’autres langages parlent de listes ou collections.

**Manipulation des tableaux en JavaScript**

En JavaScript, un tableau est un objet disposant de propriétés particulières.

**Créer un tableau**

var films = ["Le loup de Wall Street", "Vice-Versa", "Babysitting"];

On déclare un tableau à l'aide d'une paire de crochets[]. Tout ce qui se trouve entre les crochets correspond au contenu du tableau. Les différents éléments stockés sont séparés par des virgules.

Avec JavaScript, on peut stocker dans un tableau des éléments de différents types, comme dans l'exemple ci-dessous.

var tab = ["Bonjour", 7, true];

Obtenir la taille d’un tableau

La taille d'un tableau s'obtient en lui appliquant la propriété length.





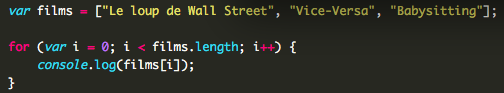
**Accéder à un élément d'un tableau**

var films = ["Le loup de Wall Street", "Vice-Versa", "Babysitting"];

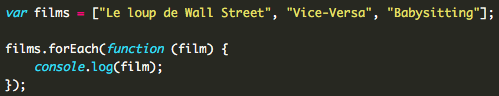
console.log(films[0]); // Affiche "Le loup de Wall Street"

console.log(films[1]); // Affiche "Vice-Versa"

console.log(films[2]); // Affiche "Babysitting"

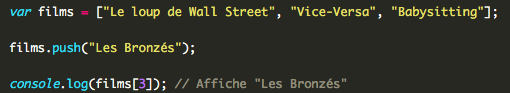
**Parcourir un tableau**

Avec la boucle for, on fait varier l'indice du tableau de 0 (indice du premier élément) à taille du tableau - 1 (indice du dernier) pour accéder aux éléments les uns après les autres.

Une autre solution consiste à utiliser la méthode forEach() sur le tableau. Celle-ci permet d'appliquer une fonction sur chaque élément du tableau. Voici l'exemple précédent réécrit avec forEach().

**Ajouter un élément dans un tableau**

L'ajout d'un nouvel élément dans un tableau se fait avec la méthode push(). Elle prend en paramètre l'élément à insérer, qui est ajouté à la fin du tableau.



**Tableaux d’objets**

Un tableau permet de stocker tout type d'élément, y compris des objets... Et même d'autres tableaux !

Imaginons que vous souhaitiez stocker également l'année de sortie de chaque film vu cette année. Une première solution est de stocker ces dates directement dans le tableau des films, juste après chaque titre.

Cependant, cela complique le parcours de la liste des films, puisqu'un indice du tableau sur deux correspond maintenant à un nombre.

Nous pouvons faire mieux en représentant chaque film sous la forme d'un objet.

A présent, on peut créer un tableau films contenant nos objets, puis l'utiliser pour afficher la description de chaque film.