Débuter en programmation

Thomas Rey

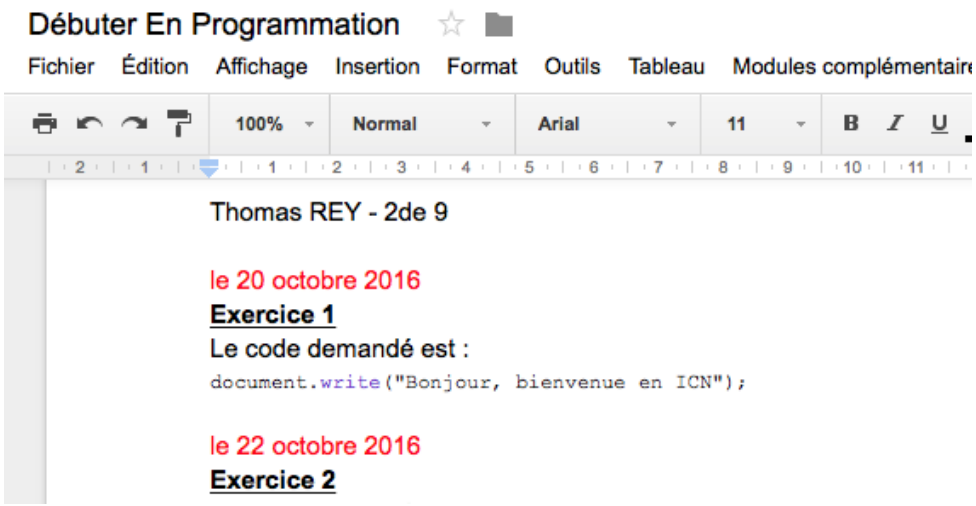
Lycée Marlioz
27 octobre 2016

Table des matières

Pour mener à bien le projet informatique que vont nous confier les élèves de SL (Sciences de Laboratoire), nous allons devoir programmer un ordinateur. La programmation consiste à écrire des instructions compréhensibles par l'ordinateur pour qu'il réalise les tâches qu'on souhaite lui confier. Pour cela, nous allons lui écrire dans un *langage* qui est une collection d'instructions de base qu'il peut exécuter.
Il existe beaucoup de langages informatiques (Java, Perl, Python, html, C++, ...). Nous allons apprendre cette année le JavaScript. Ce langage est exécutable dans les navigateurs internet (Firefox, IE, Chrome, ...) et donc utilisable sur toutes les machines supportant ces navigateurs (ordinateurs, tablettes, téléphones, ...).
Pour simplifier l'écriture de nos programmes, nous allons rapidement utiliser une *bibliothèque* d'instructions appelée P5.js (`http://p5js.org`) qui nous permettra de simplifier notre code (notamment pour dessiner à l'écran).

Dans un premier temps, nous utiliserons une plateforme de développement « en ligne » : https://jsfiddle.net
Vous allez réaliser les exercices de ce document en autonomie et recopier le code de chaque programme demandé dans un document du dossier ICN de votre drive (utilisez le « copier-coller » !). Pensez à le présenter correctement (comme un devoir) avec obligatoirement les éléments suivants (voir exemple ci-dessous) :

- vos Nom(s) - Prénom(s) - Classe en entête ;
- date de réalisation de chaque exercice ;
- numérotation des exercices ;
- code mis en forme avec la police `Courier New` ;



Enfin, lorsque vous êtes « bloqués », pensez à utiliser internet pour chercher de l'aide avant d'appeler « au secours » le professeur s'il est déjà occupé ! Un exemple de site où sont référencés la plupart des instructions JavaScript : http://www.w3schools.com/jsref/default.asp

1 Pour débuter

1.1 Affichage

Le premier programme que nous allons écrire consiste à écrire un texte à l'écran :

```
1 // Ceci est un commentaire (indispensable au correcteur mais non lu par l'ordinateur)
2 document.write("Bonjour, bienvenue en ICN");
```

Exercice 1.
Modifier ce programme pour afficher votre nom et votre classe.

Le langage JavaScript sert à exécuter des programme dans des pages web (en langage html). Dans l'affichage des textes on peut donc utiliser des *balises* html (une balise est un code permettant d'améliorer la mise en page, créer des liens, afficher des images, ...).

On donne dans le tableau suivant quelques balises :

Balise	Effet
`<strong>texte</strong>`	met `texte` en caractères gras
`<em>texte</em>`	met `texte` en caractères italiques
`<u>texte</u>`	souligne `texte`
` `	passe à la ligne (pas besoin de « fermante »)
`<code>texte</code>`	pour mettre en forme du code informatique

Exercice 2.
Créer un affichage qui utilise une ou deux balises au choix.

1.2 Compléments

On peut passer ce paragraphe dans une première lecture.
En écrivant `document.write`, JSFiddle nous prévient d'une erreur : « document.write is disallowed in JSFiddle environment and might break your fiddle. »
Pour les modestes programmes que nous écrirons, ce n'est pas grave mais pour votre culture, il faut savoir qu'il est préférable d'écrire dans la zone « HTML » le code suivant :

```
1 <div id="texte"></div>
```

Et dans la zone « JAVASCRIPT », on remplace `document.write(...);` par :

```
1 document.getElementById('texte').innerHTML = "Bonjour,
    bienvenue en ICN";
```

Inconvénients : il faut créer un « `<div>...` » par élément à afficher ;

Avantages : ceci permet de modifier les textes affichés en cours d'exécution du programme et surtout ceci permet de mettre les balises de mise en forme du texte à l'extérieur du code JavaScript : c'est plus cohérent.

1.3 Variables

Une *variable* est un espace mémoire dans lequel le programme peut stocker de l'information qui peut être numérique, booléenne[1], ...), alphanumérique, ... On peut « modéliser » une variable comme étant une boîte qui contient *une* information qui peut varier (d'où son nom !) au cours de l'exécution du programme.
En JavaScript, pour définir une variable, on commence par lui choisir un nom (qui commence par une lettre) par exemple `compteur` puis on utilise le mot clé `var` suivi du nom choisi et enfin on peut l'initialiser :

```
1 var compteur;
2 compteur = 1;
```

ou plus simplement :

1. Une variable booléenne ne peut contenir que deux valeurs : TRUE ou FALSE.

```
1 var compteur = 1;
```

Exercice 3.
On donne le code suivant :

```
1 var salaire = 10000;
2 document.write("salaire");
```

1. Que va afficher ce code ? Vérifier dans http://jsfiddle.net.
2. Corriger le code pour qu'il affiche le contenu de la variable `salaire`.

Pour modifier le contenu d'une variable on écrit `nomVariable =` suivi d'une expression qui sera son nouveau contenu.

Exercice 4.
On donne le code suivant :

```
1 var compteur = 1;
2 document.write(compteur + "<br/>");
3 compteur = 4;
4 document.write(compteur+ "<br/>");
5 compteur = compteur + 3;
6 document.write(compteur + "<br/>" );
7 document.write("fin");
```

1. Qu'affiche-t-il ? Vérifier dans http://jsfiddle.net.
2. Expliquer la ligne 5.

1.4 Saisie au clavier

Dans un programme, il peut être utile de demander une information à l'utilisateur du programme. Pour cela on utilise la fonction [2] `prompt` :

```
1 var nom, prenom, age;
2 nom = prompt("Quel est votre nom ?");
3 prenom = prompt("Quel est votre prénom ?");
4 age = prompt("Quel est votre age ?");
5 document.write("Bonjour " + prenom + " " + nom + ".<br/>");
6 document.write("Tu as " + age + "ans");
```

2. Nous verrons plus tard ce qu'est une fonction et comment écrire nos propres fonctions (partie 3).

1.5 Complément sur les variables : les types

En JavaScript, toutes les variables sont déclarées grâce au mot clé `var`. Pourtant elles ne contiennent pas nécessairement le même *type* de données (numériques, textes, booléennes. Il est parfois utile de connaître le type des données contenues dans une variable et surtout de faire des conversions entre les types.

On donne le code ci-dessous (à tester !) :

```
1 var a="bonjour ", b="Thomas";
2 var c = a+b;
3 document.write(c);
```

Dans cet exemple, les variables `a` et `b` contiennent des textes, dans ce cas, l'opérateur « `+` » sert à *concaténer* les chaînes de caractères : il les met « bout à bout ».
Ceci est important pour comprendre l'exercice ci-dessous.

Exercice 5.
On donne le code suivant :

```
1 var a = prompt("Saisir un nombre :");
2 var b = prompt("Saisir un nombre :");
3 var somme = a + b;
4 document.write("La somme vaut : " + somme);
```

1. L'utilisateur saisit les nombres 4 et 5. Qu'affiche le programme ?
2. Vérifier avec http://jsfiddle.net. Donner une explication.

Pour connaître les trois principaux types de variables, écrivons ce programme :

```
1 var a = "2", b = 2;
2 var c = (2 < 3);
3 document.write("a = " + a + " est du type " + typeof(a) + "<
    br/>");
4 document.write("b = " + b + " est du type " + typeof(b) + "<
    br/>");
5 document.write("c = " + c + " est du type " + typeof(c) + "<
    br/>");
```

Heureusement, il existe des fonctions pour convertir les types de données. Ces fonctions sont résumées dans le programme ci-dessous :

```
1 var a = "2";
2 var b = 5;
3 var c = b + parseInt(a);// a est converti en nombre
4 var d = a + b.toString();
5 document.write(c + "<br/>");
6 document.write(d);
```

Exercice 6.
Corriger l'exercice 5 pour qu'il fasse ce qu'on attend de lui !

Autres méthodes utiles :
Dans chaque cas `a`, `b` sont des variables « numériques » et `str1`, `str2`, ... des variables « chaînes de caractères ».

Méthode	Description
`a.toFixed(3)`	ne conserve que 3 chiffres après la virgule pour `a`
`b.isInteger()`	renvoie `true` si c'est un entier et `false` sinon
`str1.length`	donne la longueur de la chaîne
`str1.toLowerCase()`	chaîne `str1` écrite en minuscules
`str1.toUpperCase()`	chaîne `str1` écrite en majuscules

Beaucoup plus de propriétés et de méthodes ici pour les numériques : http://www.w3schools.com/jsref/jsref_obj_number.asp et là pour les chaînes : http://www.w3schools.com/jsref/jsref_obj_string.asp

2 Tests et boucles

Dans un programme, il est parfois utile de pouvoir exécuter ou non une partie de programme selon qu'une condition est réalisée ou non (un exemple « concret » : *SI* il pleut *ALORS* je prends mon parapluie). Il est aussi parfois intéressant de répéter un certain nombre de fois une même série d'instructions, dans ce cas on écrit une *boucle*.
Dans cette partie, nous allons apprendre à utiliser ces deux types de structures.

2.1 Tests

Le code pour écrire un *branchement conditionnel* simple est :

```
1 if(<condition>) {
2   <instructions si condition vraie>
3 } else {
4   <instructions sinon>
5 }
```

Quelques précisions :
- `<condition>` est à remplacer par le test qu'on veut effectuer (`i < 3` ou `i == 5` ou `i >= j`, ...) ;
- le bloc `<instructions si vrai>` peut contenir plusieurs instructions (séparées par des « ; ») ;
- la partie `else { ... }` est facultative.

Exercice 7.
On donne le code suivant :

```
1 var d = Math.floor(Math.random()*6 + 1);
2 document.write("Résultat : " + d + '<br/>');
```

```
3 if(d == 6) {
4     document.write("Bravo ! Tu as fait un 6.");
5 } else {
6     document.write("Essaye encore.");
7 }
```

On précise que `Math.floor` arrondit à l'entier inférieur et `Math.random()` donne un nombre aléatoire entre 0 et 1.

1. Expliquer la ligne 1 de ce programme.
2. Que fait ce programme ?

2.2 Boucles

Vous avez déja utilisé la notion de boucle depuis le début d'année :
- dans le concours Castor-informatique : *répéter* six fois la séquence d'instructions suivantes...
- dans la rédaction du jeu des grenouilles : *tant que* c'est possible, déplacer une grenouille rouge, ...

On voit dans ces deux exemples qu'il existe (au moins) deux types de boucles :
- les boucles qui réalisent un nombre de fois *défini* une série d'instructions ;
- les boucles qui réalisent une série d'instructions *tant qu*'une condition est remplie.

Exercice 8.
On donne le code suivant :

```
1 var table = 4;
2 var i;
3 for(i=0; i<=10; i=i+1) {
4   document.write(table +" x " + i + " = " + table*i + "<br/>"
      );
5 }
```

1. Peut-on déterminer le nombre de fois que sera exécutée cette boucle ?
2. Que fait ce programme ?
3. Modifier ce programme pour qu'on demande à l'utilisateur quelle table il souhaite afficher.
4. Modifier ce programme en ajoutant une variable `fin` dont le contenu est demandé à l'utilisateur et qui donne le dernier facteur à multiplier. Modifier la ligne 3 pour tenir compte de cette contrainte supplémentaire.

Exercice 9.

1. Écrire un programme qui affiche 10 nombres aléatoires entre 1 et 6.
2. Modifier le programme précédent pour qu'il compte en plus le nombre de fois où le « 6 » est sorti.

Exercice 10.
En utilisant une boucle et un branchement conditionnel, écrire un programme qui affiche tous les diviseurs d'un nombre demandé à l'utilisateur dans une variable appelée `n`.
Aides :
- un nombre d est un diviseur de n si le reste de la division de n par d vaut 0 ;
- en JavaScript, on obtient le reste de la division euclidienne grâce à l'opérateur modulo : `%`. Exemple après l'instruction `var r = 13 % 3;` la variable `r` contient 1 car $13 = 4 \times 3 + 1$.

Exercice 11.
On donne le code suivant :

```
1 var de = Math.floor(Math.random()*6 + 1);
2 document.write("dé sorti : " + de + "<br/>");
3 while (de < 6) {
4   de = Math.floor(Math.random()*6 + 1);
5   document.write("dé sorti : " + de + "<br/>");
6 }
```

1. Peut-on prévoir le nombre de fois que le bloc d'instructions (lignes 4 et 5) va s'exécuter ?
2. Que fait ce programme ?
3. Modifier le code pour qu'il compte le nombre de lancers effectués.

Exercice 12.
Écrire un programme qui pose des questions sur les tables de multiplication (« combien font 3×7 ? ») avec les contraintes suivantes :
- les nombres sont tirés au hasard entre 2 et 10 ;
- premier cas : on demande 10 calculs et on affiche le score à la fin ;
- deuxième cas : on joue tant que les réponses sont justes et on affiche le nombre de réponses justes à la première erreur.

3 Fonctions

Vous avez déjà utilisé la notion de fonction dans le jeu LightBot : une fonction permettait d'exécuter plusieurs fois la même séquence d'instructions et donc « d'économiser » des cases d'instructions dans le programme principal.
Observons le code suivant (exemple mathématique avec les fonctions affines) :

```
1 var i;
2 // Définition de la fonction f affine f:x->2x-3
3 function f(x) {
4   var y = 2*x - 3 ;
5   return y;
6 }
7
```

```
8  // Utilisation de f dans un programme "principal" :
9  for (i=-3; i<=3; ++i) {
10   document.write("l'image de " + i + " est " + f(i) + "<br/>"
       );
11 }
```

Dans la définition de la fonction, `x` est un *argument*. Sa valeur n'est connue qu'à l'intérieur de la fonction (si on cherche à afficher la valeur de `x` ailleurs, il ne se passera rien).
Une fonction peut prendre plusieurs *arguments* séparés par des virgules ou n'en prendre aucun (dans ce cas on la définit ainsi : `function nomDeMaFonction(){}`).
À la ligne 10, le programme fait appel à la fonction `f` et remplace `f(i)` par la valeur de l'expression qui est située après l'instruction `return`.

Exercice 13.
Ajouter l'instruction suivante à la fin du programme précédent :

```
1 document.write("y = " + y);
```

Affiche-t-il une valeur ? Pourquoi ne connait-il pas la variable `y` ?

En informatique, une fonction ne permet pas seulement de faire des calculs et elle ne renvoie pas nécessairement un résultat. On peut aussi la définir à n'importe quel endroit du programme (ce n'est pas obligatoire qu'elle soit au début).

Exercice 14.
On donne le code suivant :

```
1  var directeur = "Claude";
2  courier("Thomas");
3  courier("Alexandre");
4  courier(directeur);
5
6
7  function courier(nom) {
8    document.write("Bonjour " + nom + "<br/>");
9    document.write("Merci de bien vouloir passer dans mon
       bureau pour...");
10   document.write("<br/>");
11   document.write("À bientôt " + nom + "<br/><br/>");
12 }
```

Que fait ce programme ?

Exercice 15.
Écrire une fonction qui prend un argument appelé `nombre` et qui renvoie le texte `+` ou `-` selon que `nombre` est positif ou négatif.
Tester cette fonction dans un programme.

Exercice 16.
Écrire une fonction qui prend deux arguments `texte` et `n` et qui écrit `n` fois le contenu de `texte` en passant à la ligne à chaque fois.

4 Variables : un peu plus loin...

Parfois, il peut-être utile de stocker un grand nombre de données de même type (numérique, texte, ...) dans une seule variable ; on appelle une telle variable un *tableau*. Il existe aussi dans de nombreux langages (les langages orientés objet) des sortes de « super variables » que l'on peut construire et qu'on appelle *objets* ; ces objets sont eux-même constitués de variables et de fonctions (ou méthodes).

4.1 Tableaux

On peut définir un tableau comme une liste ordonnée de valeurs rangées dans une variable. Par exemple :

```
1 var eleves = {"Laura", "Enzo", "Lucas", Artur", "Luca"};
```

définit le tableau `eleves` qui est constitué de 5 *entrées* numérotées de 0 à 4 (on commence toujours la numérotation à zéro).
On peut lire, écrire, modifier le contenu de chacune de ses entrées :

```
1 var eleves = ["Laura", "Enzo", "Lucas", "Artur", "Luca"];
2 var i;
3 document.write(eleves[0] + " est la première de la liste.<br
    />");
4 eleves[3] = "Arthur";// c'est mieux non ?
5 eleves[4] = eleves[4] + "s";// ajouter un 's'
6 for (i=0; i<5; ++i) {
7   document.write(eleves[i] + "<br/>");
8 }
```

Exercice 17.
On peut aussi déclarer un tableau et le remplir ensuite :

```
1 var notes = [];
2 var i, nbEleves = 3, somme = 0;
3 for (i=0; i<nbEleves; ++i) {
4   notes[i] = prompt("Quelle note pour l'élève " + i + " ?");
5   somme = somme + notes[i];
6 }
7 var moy = somme / nbEleves;
8 document.write("La moyenne de classe est " + moy);
```

1. Tester ce programme dans https://jsfiddle.net. Fonctionne-t-il ?

2. Ajouter une ligne `document.write(somme + "<br/>");` avant l'affichage de la moyenne. Que constate-t-on pour la somme ?
3. Convertir le résultat de `prompt(...)` en nombre avant de le stocker dans le tableau `notes`. Tester à nouveau.

4.2 Objets

Cette partie est plus difficile et réservée à ceux qui ont bien compris le reste de ce document...
Si on souhaite programmer un jeu de societé, il faudra « gérer » plusieurs joueurs qui chacun auront un nom, une position, un nombre de points, un pion, ... Dans ce cas, il est judicieux de créer un *objet* `Joueur` qui aura tous ces *attributs* et aura même des *méthodes* permettant de modifier ces attributs, de les afficher, ...
Observons le code suivant :

```
1  // On crée le type d'objets "Joueur"
2  Joueur = function() {
3    var nom;
4    var position;
5    var pion;
6    var points;
7    // Méthode d'initialisation
8    this.init = function(nom, pion) {
9      this.nom = nom;
10     this.pion = pion
11     this.position = 0;
12     this.points = 100;
13   }
14   //Méthode pour faire avancer le pion
15   this.avance = function(n) {
16     this.position = this.position + n;
17   }
18   // Méthode pour augmenter/diminuer le nombre de points
19   this.gagne = function(n) {
20     this.points = this.points + n;
21   }
22   // Méthode pour afficher le bilan du joueur
23   this.affiche = function() {
24     document.write(this.nom + " a choisi le pion " + this.
         pion + "; il a " + this.points + " pts et il est sur
         la case " + this.position);
25     document.write("<br/>");
26   }
27
28 }
29
```

```
30 // Programme principal : on crée les joueur et on les
       initialise
31 j1 = new Joueur();
32 j2 = new Joueur();
33 j1.init("Laura", "voiture");
34 j2.init("Enzo", "avion");
35
36 j1.affiche();
37 j2.affiche();
38
39 j1.gagne(20);
40 j1.avance(2);
41 j2.gagne(-10);
42
43 j1.affiche();
44 j2.affiche();
```

À vous d'inventer de nouveaux objets dans le cadre de vos projets...

5 Pour finir...

La plateforme https://jsfiddle.net nous a été très utile pour apprendre les rudiments du JavaScript, désormais nous allons créer de « vraies » pages html en y intégrant nos scripts.
Sur votre clé usb (ou à défaut dans votre espace personnel) créer un dossier `ICN` et y placer le dossier « dezippé » que vous pouvez télécharger en cliquant ici.
Désormais, tout ce qu'on écrivait dans la fenêtre HTML de JSFiddle, s'écrira dans le fichier `index.html`, tout ce qu'on codait dans la fenêtre JAVASCRIPT de JSFiddle sera à copier dans le fichier `script.js`.
Pour ceux qui connaissent le `css`, vous pouvez le modifier également.
Le résultat sera visible en ouvrant le fichier `index.html` dans votre navigateur préféré et les modifications seront à faire en ouvrant les fichier du dossier dans un éditeur de texte quelconque (nous utiliserons NotePad++ au lycée – à télécharger chez vous gratuitement depuis ce lien).