

## Table des matières

I. Pourquoi le Javascript pour initier à la programmation.....	1
II. Un environnement de développement : ProgLab.fr.....	1
III. Efficacité pour exécuter un programme.....	2
IV. Une interface conçue pour apprendre.....	3
1) Comportement lors de la saisie d'un script.....	3
2) Aides rapides.....	4
3) Une aide plus fournie.....	5
4) Usages variés.....	5
V. Diffuser/Publier un programme.....	5
1) Pour un usage dans ProgLab.fr.....	6
2) Pour un usage documentaire.....	6
3) Pour un usage autonome en local ou sur un site internet.....	6
VI. Utiliser une zone de dessin.....	8
VII. Gérer le clavier.....	10
VIII. Gérer la souris.....	11
IX. Faire de la géométrie dynamique.....	12

## I. Pourquoi le Javascript pour initier à la programmation

Le langage [Javascript](http://wikipedia.fr) (wikipedia.fr) est un langage de programmation à l'origine développé pour réaliser des scripts dans des pages web interactives ou non. Il est omniprésent de manière transparente sur la plupart des sites visités. Ce langage est **normalisé** et **d'usage professionnel**. Il est **accessible à des élèves jeunes**.

Un programme Javascript est basé sur un **texte ou script**, généralement **présent en clair** dans une page web. Il est accessible (sous FireFox, clic droit, Voir le code source), il est lisible et compréhensible, même si on peut le déporter dans un fichier annexe afin de ne pas alourdir cette page. Il suffit d'un **éditeur de textes** pour lire ou écrire un programme.

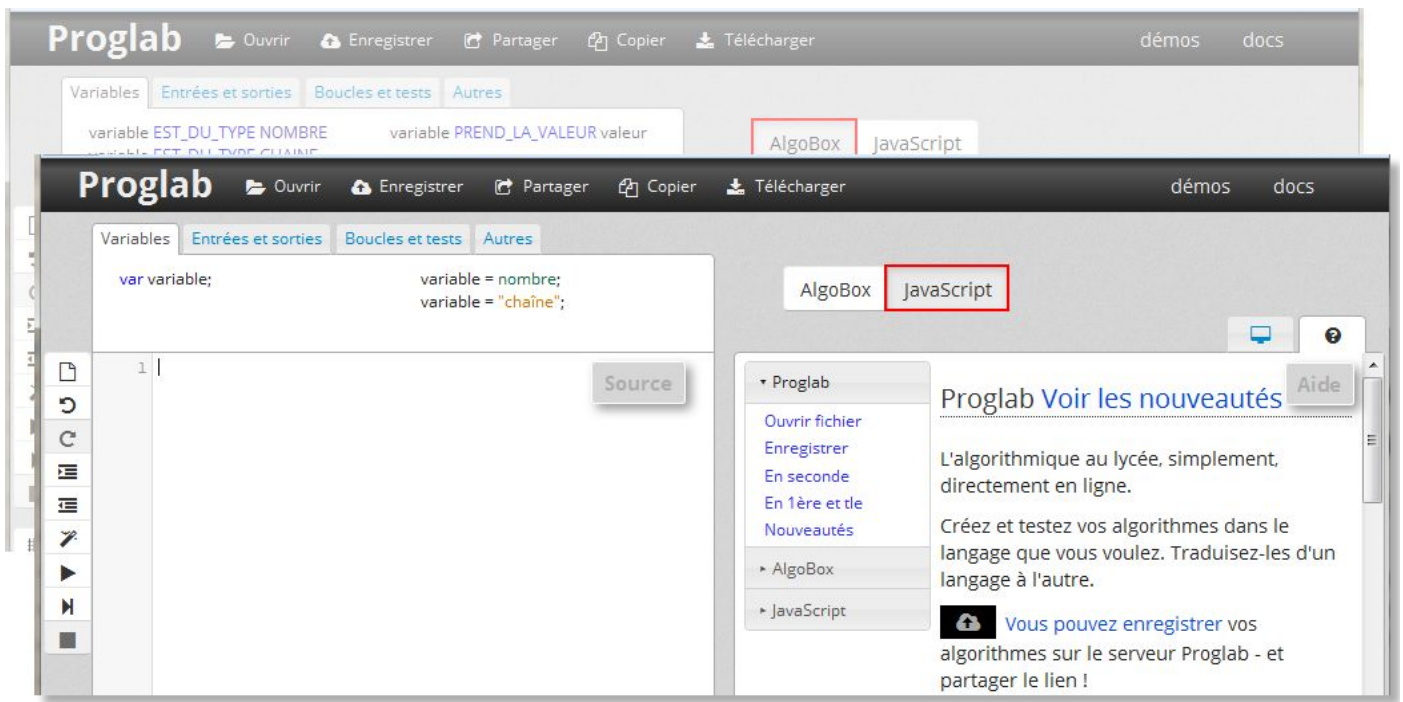
Pour faire fonctionner un programme en Javascript, **il suffit d'un navigateur internet**, même sans connexion internet, **aucune installation logicielle complémentaire** n'est nécessaire en général.

Un inconvénient qui n'en est pas un à l'usage : il utilise un langage basé sur des mots-clés **en anglais** mais dans un anglais assez **transparent et simple** que la plupart des langages utilisent : « **if then else for while function print write** ». On pourrait franciser en jumelant les mots-clés, mais c'est l'occasion de faire un peu d'anglais technique que côtoieront les élèves dans leurs études ou dans leur futur milieu professionnel, notamment si leur **orientation** comporte une partie « **outils numériques** ».

Javascript est l'**un des langages les plus utilisés au monde**, et le plus [recherché par les recruteurs actuellement](#) (developpez.com). C'est récent et ce n'est que le début : de plus en plus d'entreprises passent progressivement au "tout JS" y compris sur les serveurs et les applis mobiles. Alors [quitte à choisir](#) ...

## II. Un environnement de développement : ProgLab.fr

Dans ce document le site [ProgLab.fr](http://ProgLab.fr) sera utilisé pour écrire, tester et exécuter les programmes. En plus d'être en français, il est développé par un professeur de mathématiques François Pirsch **pour apprendre à programmer facilement en milieu scolaire**.



ProgLab.fr a été réalisé en Javascript (!) pour faire de l'algorithmique au lycée avec le langage [AlgoBox](#), ou pas, car le langage Javascript plus complet est supporté. L'interface permet de basculer de l'un à l'autre directement quand le code le permet.

Comme dit précédemment, un simple bloc-notes texte et un navigateur internet pourraient suffire à développer un programme en Javascript, notamment en l'absence de connexion internet. Mais un environnement de développement comme [ProgLab.fr](#) est utile, même aux programmeurs aguerris, pour **localiser rapidement les erreurs** de syntaxe, les fautes de frappe sur le vocabulaire classique, pour **s'aider de la coloration syntaxique et de la mise en page** par blocs afin d'**organiser et commenter son code**.


### III. Efficacité pour exécuter un programme

Voici un programme en langage Javascript écrit sous ProgLab et à tester dans [ProgLab.fr](#) : il demande 2 nombres puis les affiche avec leur somme et leur produit.

```

1 //Entrée
2 var n1 = proglab.inputNumber("Donnez un 1er nombre");
3 var n2 = proglab.inputNumber("Donnez un 2nd nombre");
4 //Traitement
5 var somme = n1 + n2;
6 var produit = n1 * n2;
7 //Sortie
8 proglab.print("Vous avez donné 2 nombres : ");
9 proglab.println(n1 + " et " + n2 + ".");
10 proglab.println("Leur somme est " + somme + ".");
11 proglab.println("Leur produit est " + produit + ".");

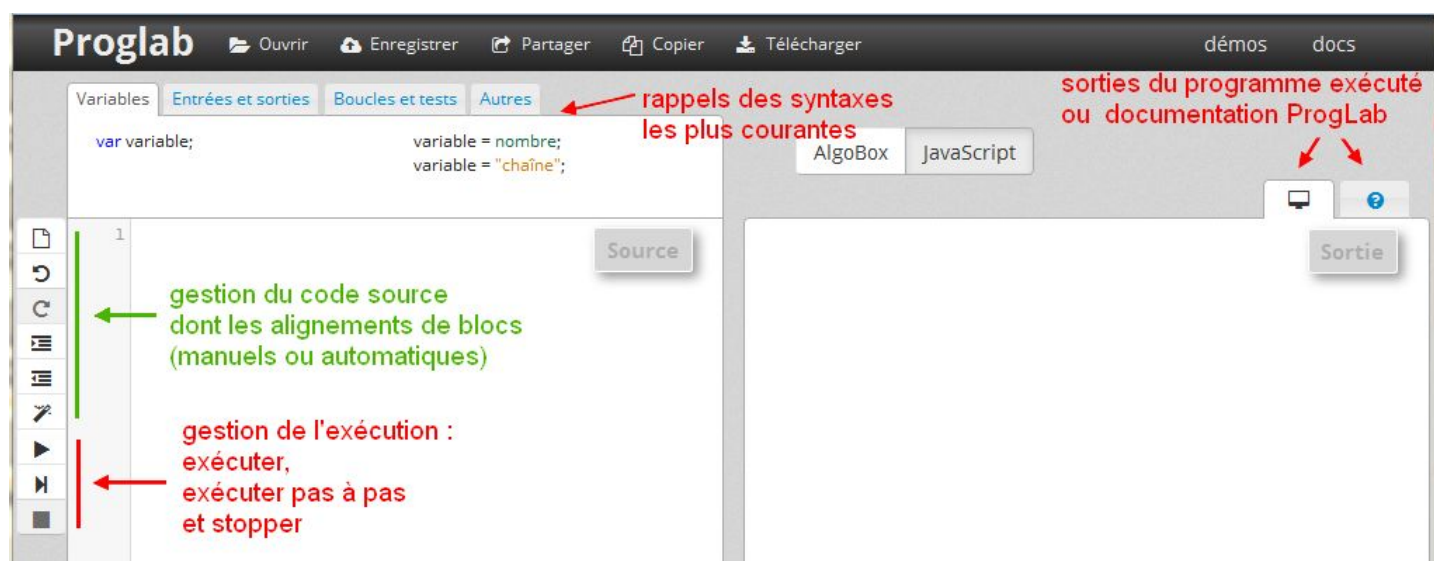
```

- Le copier et le coller dans la partie « Source » en mode Javascript de [ProgLab.fr](#) : *ce lien ouvre directement ProgLab en mode Javascript à l'adresse <http://www.proglab.fr/?lang=js>.*
- Puis l'exécuter en cliquant tout à gauche sur le bouton  **Lancer le programme**.
- Et voilà !

La patte du professeur de mathématiques qui a réalisé ce logiciel en ligne se fait tout de suite sentir à l'usage du programme : la saisie des nombres permet de s'aventurer au-delà des entiers et décimaux avec, par exemple,  $1/2$ ,  $\sqrt{2}$ ,  $\cos(3,1416/4)$  ...

Mais elle est encore plus forte dans la conception de l'interface dont la pratique du copier-coller occulte un des intérêts de l'environnement [ProgLab.fr](http://ProgLab.fr).

## IV. Une interface conçue pour apprendre



Avec autant de place pour le code (cadre Source) que pour les sorties (cadre Sortie), faire plus simple et plus clair semble difficile.

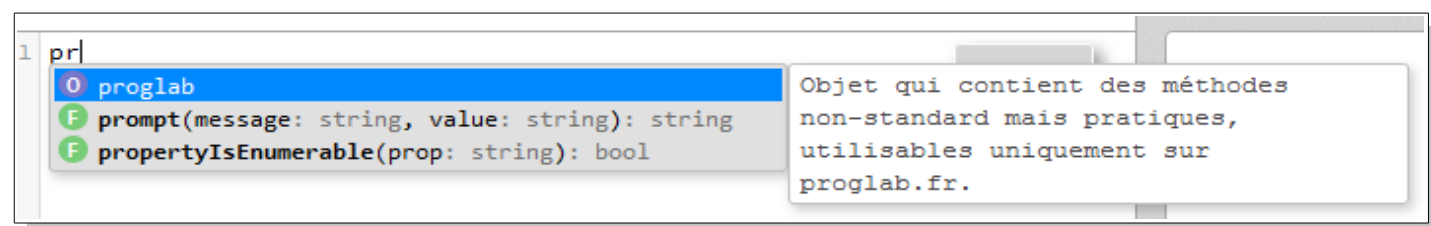
### 1) Comportement lors de la saisie d'un script

Voici un exemple simple pour afficher un texte :

```
proglab.println("Voici un premier test");
```

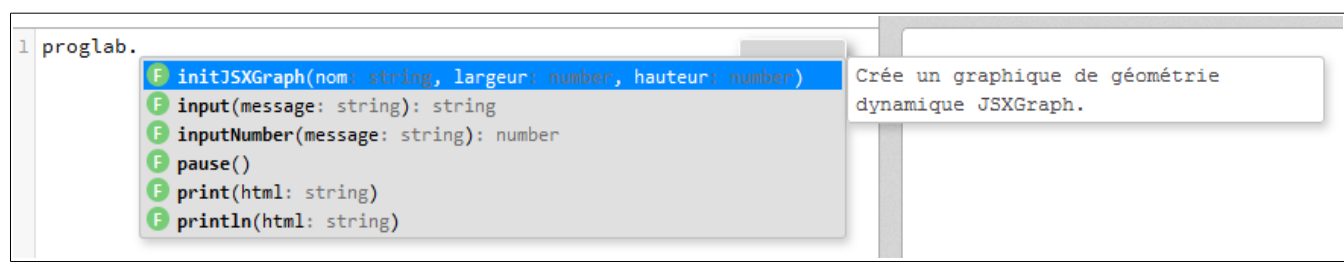
Il faut **le taper au clavier** et non le copier-coller afin de voir le comportement de l'interface ! Elle est conçue pour aider au maximum l'utilisateur non expert dans le langage, l'aider à rédiger comme à comprendre l'usage des mots clés (fonctions, paramètres et syntaxes).

Dès la frappe des premières lettre du mot `proglab`, l'interface propose des solutions pour compléter le mot, et donne des explications sur l'usage ou la syntaxe :

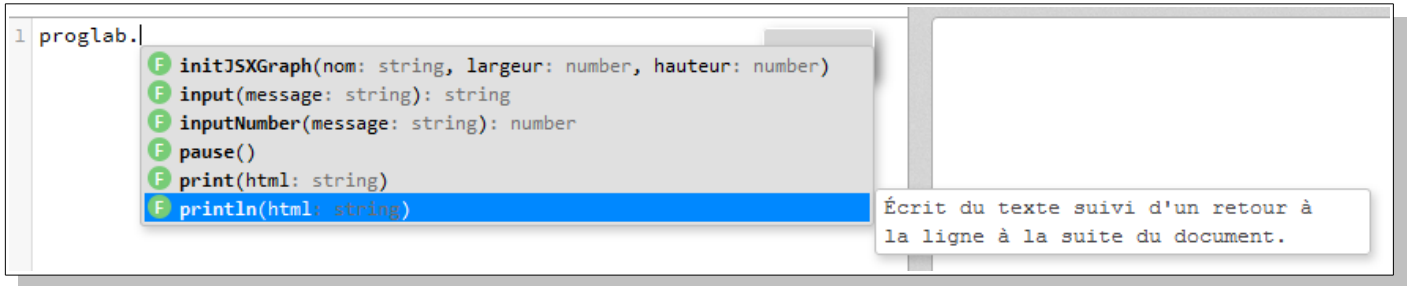


Vous pouvez continuer de taper le mot, ou valider pour conserver le 1er de la liste ou sélectionner une alternative avec les touches fléchées haut et bas puis la touche Entrée pour valider.

Arrivé au `.` l'interface propose de choisir l'une des fonctions disponibles pour l'objet `proglab` :



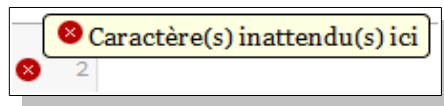
Avec les touches fléchées pour parcourir cette liste, les 2 syntaxes sont détaillées :



On voit 2 fonctions similaires : `print` et `println`.

La comparaison des informations dispensées montre que `print` écrit un texte dans la zone de sortie courante, que `println` fait de même mais avec un retour à la ligne final. Cela permet de déduire que `print` permet de compléter un texte d'une même ligne par des appels successifs.

Dans la marge le rond rouge signale un problème possible, en l'occurrence à ignorer car il faut terminer la saisie d'abord. Voilà ce qui est proposé si la souris s'attarde sur l'image rouge :



Enfin les paramètres de la fonction sont déjà proposés : ici un seul objet de type *string* est attendu, c'est-à-dire une chaîne de caractères. Ce typage est informatif, pour savoir quel type de donnée est attendue et traitée car, à la différence d'autres langages, le type est a priori : c'est lors de l'affectation que le « bon » type est associé par le moteur logiciel du navigateur qui interprète le code Javascript.

Quand il est saisi directement au clavier, un texte doit être entre guillemets (doubles) droits : "".

Le nom présenté pour le paramètre `html` n'est pas anodin :

il sous-entend que la chaîne peut comporter des codes HTML, utiles pour mettre en forme rapidement un texte :

```
proglab.println("Voici un <b>premier</b> test");
```

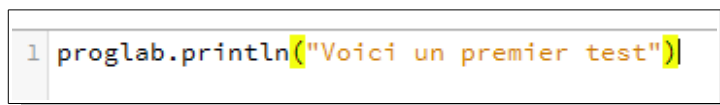
affichera

```
Voici un premier test
```

avec le mot `premier` en caractères gras grâce aux balises `<b> ... </b>`, `b` pour bold, gras en anglais.

La coloration syntaxique est parlante : les textes sont en orange pour se distinguer du code interprété.

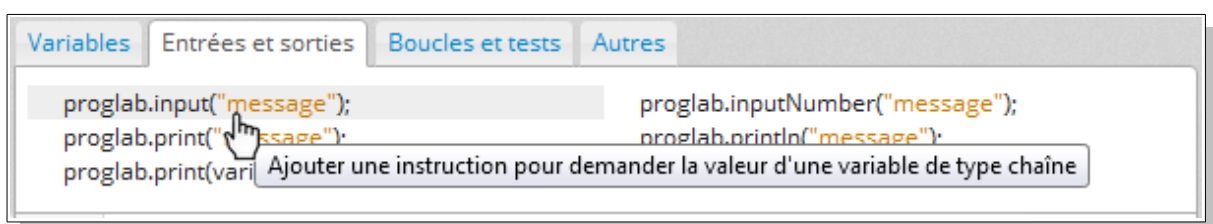
Ensuite, quand on saisit la dernière parenthèse, la parenthèse fermante, ProgLab met en évidence la parenthèse ouvrante afin de contrôler rapidement qu'il n'y a pas d'erreur sur les niveaux de parenthèses.



Enfin le point virgule ; final est optionnel mais conseillé. Il n'y aura pas d'erreur si on l'omet.

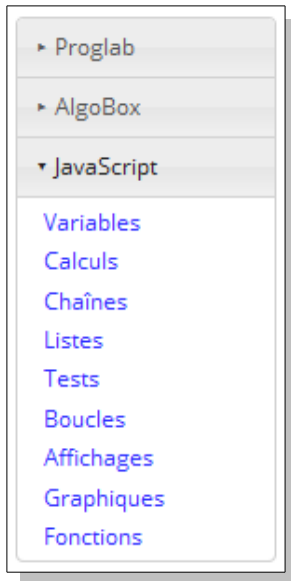
## 2) Aides rapides

Elles sont dispensées en haut de page où sont regroupés par usages les mots-clés courants et leurs syntaxes : en laissant la souris dessus, une information est donnée sur leur usage réel, ce qui permet de trouver ou retrouver même si la traduction en français n'est pas évidente (et de finir par mémoriser à force de chercher).



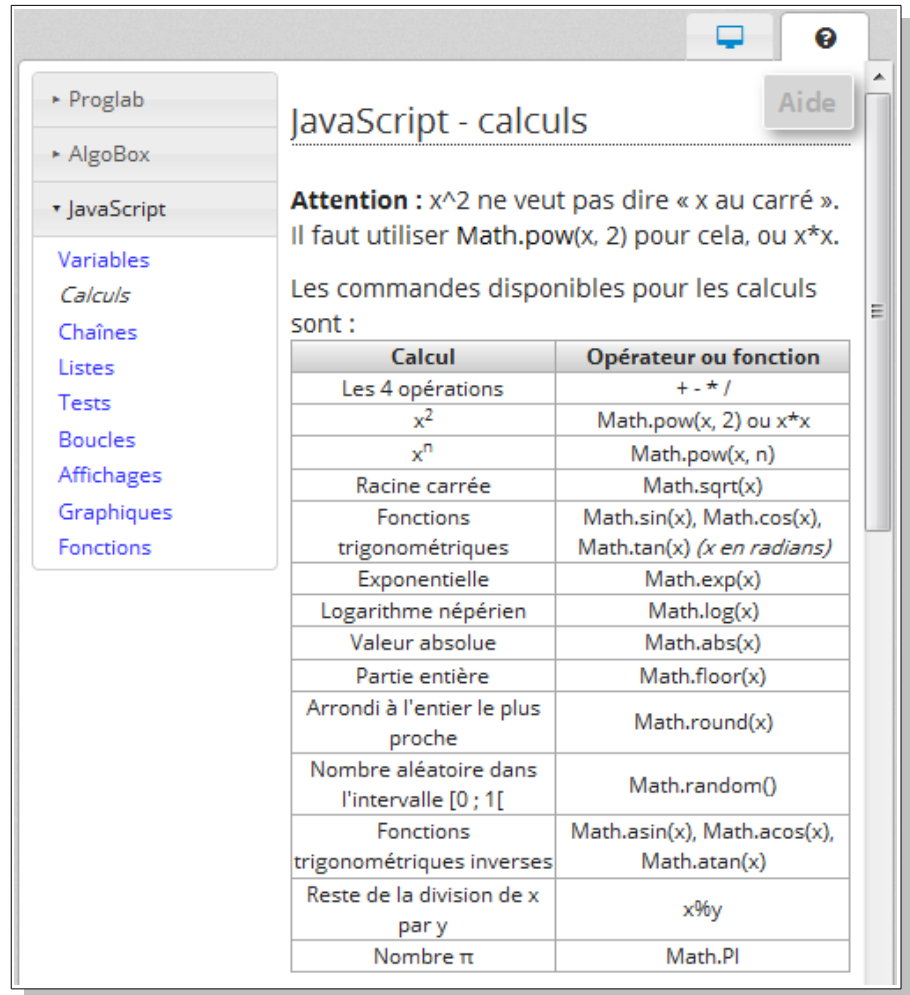
### 3) Une aide plus fournie

L'onglet  au-dessus de la zone des sorties affiche une aide locale plus détaillée, classée par thème.



Elle reprend par le menu l'essentiel de ce qu'il y a à savoir pour transcrire un algorithme en Javascript, avec quelques explications et surtout des exemples de codage.

Ainsi la liste des fonctions mathématiques disponibles en Javascript s'avère courte mais suffisante. Des fonctions AlgoBox sont disponibles en Javascript mais ne sont a priori pas utiles pour les élèves en collège : le principe est de programmer les fonctions qui manqueraient.



**JavaScript - calculs**

**Attention :**  $x^2$  ne veut pas dire « x au carré ». Il faut utiliser `Math.pow(x, 2)` pour cela, ou `x*x`.

Les commandes disponibles pour les calculs sont :

Calcul	Opérateur ou fonction
Les 4 opérations	+ - * /
$x^2$	<code>Math.pow(x, 2)</code> ou <code>x*x</code>
$x^n$	<code>Math.pow(x, n)</code>
Racine carrée	<code>Math.sqrt(x)</code>
Fonctions trigonométriques	<code>Math.sin(x)</code> , <code>Math.cos(x)</code> , <code>Math.tan(x)</code> ( <i>x en radians</i> )
Exponentielle	<code>Math.exp(x)</code>
Logarithme népérien	<code>Math.log(x)</code>
Valeur absolue	<code>Math.abs(x)</code>
Partie entière	<code>Math.floor(x)</code>
Arrondi à l'entier le plus proche	<code>Math.round(x)</code>
Nombre aléatoire dans l'intervalle [0 ; 1[	<code>Math.random()</code>
Fonctions trigonométriques inverses	<code>Math.asin(x)</code> , <code>Math.acos(x)</code> , <code>Math.atan(x)</code>
Reste de la division de x par y	<code>x%y</code>
Nombre $\pi$	<code>Math.PI</code>

### 4) Usages variés

Outre des sorties sous forme de texte simple, les mises en forme HTML sont utilisables comme mettre en gras, construire et peupler un tableau ...

Les sorties peuvent aussi être graphiques voire géométriques : visitez [les démos](#) dont [Fractale du Dragon](#) ou [Loi binomiale](#) ou [Polygones réguliers à n côtés](#).

Enfin on peut gérer le clavier, la souris voire des images.

A partir de la [partie VI](#) de ce document des exemples simples seront proposés pour illustrer

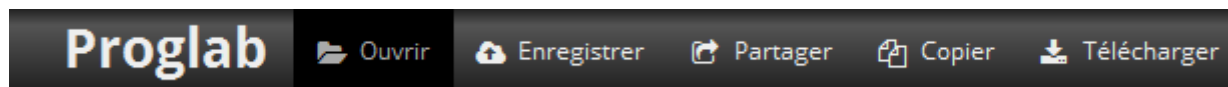
## V. Diffuser/Publier un programme

Comme déjà dit, l'intérêt de l'utilisation du Javascript est qu'il n'y a pas à s'inquiéter du système d'exploitation ou d'un logiciel tiers à installer pour faire fonctionner un programme : un navigateur internet suffit.


Un autre intérêt d'un programme écrit à l'aide d'un simple texte est qu'il est facilement échangé avec un fichier texte, ou directement dans un message ou un article en ligne !


### 1) Pour un usage dans ProgLab.fr

Outre le simple **coller du script**, [ProgLab.fr](http://ProgLab.fr) permet de charger ou sauvegarder ses scripts via la barre en haut de page :





En local :

le bouton  permet classiquement de charger un fichier texte présent sur son ordinateur,

le bouton  permet à l'inverse de récupérer directement le fichier texte sur son ordinateur.


En ligne :

le bouton  permet de stocker le programme en ligne sur le serveur de ProgLab.fr,

le bouton  donne alors un lien court pour ouvrir directement ProgLab.fr et le projet !

Voir les démos plus haut : la Fractale du Dragon est accessible par <http://proglab.fr/oth194> .

### 2) Pour un usage documentaire

Le bouton  permet de **récupérer le texte avec sa coloration syntaxique**, ce qui est pratique pour un professeur qui voudrait proposer des scripts dans une fiche de travail.

### 3) Pour un usage autonome en local ou sur un site internet

Il suffit « presque » de coller le texte du programme dans le corps du texte de la page internet entre 2 balises :

```
1 <script type="text/javascript">
2   // le texte de mon programme/script
3 </script>
```

Les couleurs sont ici utilisées pour montrer les différentes écritures et parties du code d'un page HTML, ce qui n'est pas possible dans un bloc-notes, sans être gênant pour le bon fonctionnement du code :

- magenta : balisage d'un script au sein du code HTML
- bleu : balisage du code HTML autour.

Mais c'est un petit peu plus compliqué car ProgLab.fr est un peu plus riche que le Javascript de base pour faciliter l'usage par les élèves de tout âge.

Ainsi les fonctions `proglab.xxxx` ne sont pas dans le standard, mais elles sont pratiques car elles évitent des appels plus lourds. Par exemple `proglab.println` ressemble à la fonction standard `document.writeln` qui a le même comportement sous ProgLab. Mais la seconde syntaxe est faite pour écrire directement dans le source HTML d'un page web avec un retour à la ligne dans ce source et non à l'affichage : le retour à l'affichage doit être géré par des balises HTML comme `<br/>` (break pour coupure) ou `<p> ... </p>` (pour paragraphe) à mettre dans le texte à afficher.

Pour éviter d'avoir à convertir votre code javascript pour le rendre totalement standard, il est plus simple de faire appel à un script séparé nommé `proglab-runtime.min.js` qui contient une version allégée du noyau des fonctions de ProgLab et permet donc de les utiliser de manière transparente. Ce fichier peut simplement être mis dans le même dossier que celui du fichier HTML produit.

Le code d'appel supplémentaire est alors :

```
1 <script type="text/javascript" src="proglab-runtime.min.js"></script>
```

Voici le code pour une page web HTML, minimale, fonctionnelle pour un programme Javascript de ProgLab :

```
1 <html>
2 <body>
3   <script type="text/javascript" src="proglab-runtime.min.js"></script>
4   <script type="text/javascript">
5     proglab.println("Voici un premier test");
6   </script>
7 </body>
8 </html>
```

Il suffit d'enregistrer le document texte, de le renommer avec un extension `.html` afin que le navigateur internet par défaut soit utilisé pour ouvrir le document, l'afficher et ainsi exécuter le programme qu'il contient.

- `<html>` et `</html>` balisent le début puis la fin du code en HTML de la page ;
- `<body>` et `</body>` balisent le corps (body en anglais) de la page HTML, ce qui est affiché à l'écran.

On peut personnaliser ce code en ajoutant par exemple une entête dans la section `<head>` (head signifiant tête ou ici entête) qui précise




- le titre (title en anglais) affiché dans la barre titre du navigateur : balisage `<title>` et `</title>`,
- l'auteur de la page HTML : champ `author` d'une balise `<meta />`. (elle est seule, sans `</meta>` final mais se termine par un `/` avant le `>`)

```
1 <html>
2   <head>
3     <title>Page premier test</title>
4     <meta name="author" content="E.Ostenne" />
5   </head>
6   <body>
7     <script type="text/javascript" src="proglab_tools.js"></script>
8     <script language="javascript">
9       proglab.println("Voici un premier test");
10    </script>
11  </body>
12 </html>
```

Ce n'est plus de la programmation Javascript mais de la programmation de page web en HTML, tout aussi incontournable pour désacraliser ce qu'est essentiellement une page web : un texte structuré qui décrit comment afficher un contenu.

Des explications sont proposées dans [l'article sur les balises meta](#) (AlsaCreations.com) ou dans ce cours sur [les bases du HTML](#) (Developpez.com).

**Par souci d'efficacité et de simplicité**, l'archive `proglab_export.zip` contient le noyau allégé de ProgLab.fr un tel fichier `index.html` (commenté) dont il suffit de compléter dans la section `<script> ... </script>` en fin de document avec un script copié depuis Proglab.fr. L'archive contient :

Nom	Type	Taille
 index.html	Firefox HTML Doc...	2 Ko
 proglab.css	Document de feui...	1 Ko
 proglab-runtime.min.js	Fichier de script JS...	21 Ko

L'archive `proglab_export_jsx.zip` contient en plus le noyau hors ligne de JSXGraph pour faire de la

géométrie dynamique :

Nom	Type	Taille
index.html	Firefox HTML Doc...	2 Ko
jsxgraph.css	Document de feui...	2 Ko
jsxgraphcore.js	Fichier de script JS...	536 Ko
proglab.css	Document de feui...	1 Ko
proglab-runtime.min.js	Fichier de script JS...	21 Ko

## VI. Utiliser une zone de dessin

[ProgLab.fr](http://ProgLab.fr) rappelle les objets de programmation essentiels pour dessiner dans l'onglet supérieur Autres :

```
Variables Entrées et sorties Boucles et tests Autres
// Commentaire
Créer une zone de graphique JSXGraph
Créer une zone de graphique
Tracer un segment
Tracer un point
//Créer une zone de graphique
2 var canvas = document.body.appendChild( document.createElement(
3 var context = canvas.getContext( "2d" );
4 canvas.width = 400; // Largeur en pixels
5 canvas.height = 300; // hauteur en pixels
//Tracer un point
7 context.beginPath();
8 context.arc(•, •, 3, 0, 2*Math.PI);
9 context.fill();
10
```

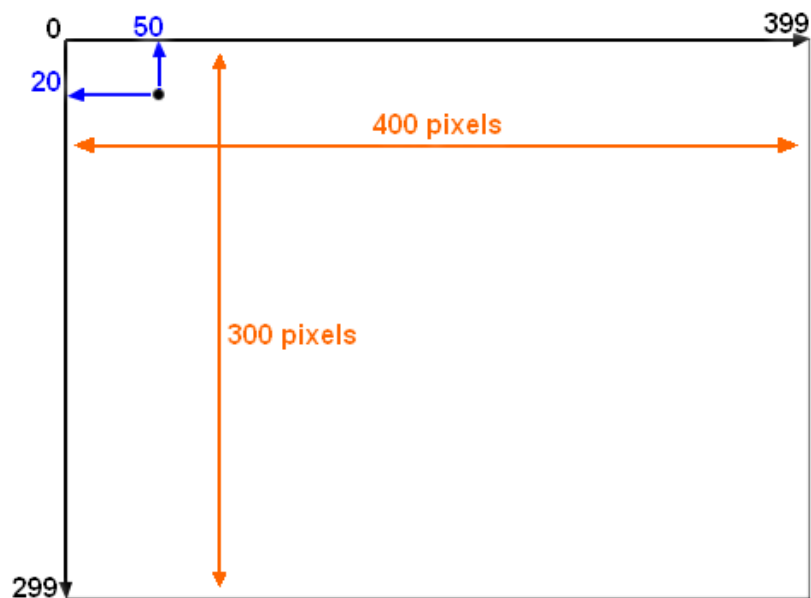
Ce script qui affichera un point « rond » a été obtenu par une série de clics :

1. sur `//commentaire`
2. sur `Créer une zone graphique`
3. sur `//commentaire`
4. et sur `Tracer un point.`

⊗ Le drapeau rouge en marge signale que le script n'est pas fonctionnelle en l'état : il faut remplacer les 2 points noirs par les coordonnées du point, dans le repère du canevas (`canvas`) faisant ici 400 par 300 pixels. Ce repère est celui conventionnellement utilisé sur les interfaces graphiques informatiques, sous-entendu non mathématiques, comme des moniteurs :

- l'origine est en haut à gauche,
- l'axe horizontal est orienté de gauche à droite,
- l'axe vertical est orienté de haut en bas,
- l'unité est ici le pixel (point d'un moniteur).





Voici le script pour un copier-coller éventuel, en ayant mis 50 et 20 comme coordonnées :

```

1 //Créer une zone de graphique
2 var canvas = document.body.appendChild( document.createElement("canvas" ) );
3 var context = canvas.getContext( "2d" );
4 canvas.width = 400; // Largeur en pixels
5 canvas.height = 300; // hauteur en pixels
6 //Tracer un point
7 context.beginPath();
8 context.arc(50, 20, 3, 0, 2*Math.PI);
9 context.fill();
10

```

Les lignes 2 à 5 règlent le canevas (`canvas`) sur lequel le dessin sera réalisé :

- Ligne 2 : création du canevas de dessin dans la page et récupération de l'objet correspondant.
- Ligne 3 : récupération du contexte de ce canevas, c'est-à-dire les fonctionnalités disponibles pour gérer le dessin sur le canevas. Le canevas assure simplement le rendu graphique, qui peut être sur un moniteur comme sur une imprimante par exemple.
- Ligne 4 et 5 : réglage des dimensions du canevas.

Les lignes 7 à 9 dessinent un point sous forme d'un cercle rempli :

- Ligne 7 : déclaration d'un chemin à tracer,
- Ligne 8 : un arc de cercle est tracé : centre de coordonnées (50 ; 20), de rayon 3, d'angle initial 0 rad et d'angle final  $2\pi$  rad pour fermer l'arc (les angles définis sur un cercle en tournant dans le sens horaire),
- Ligne 9 : la fonction `fill()` dessine – c'est à dire provoque le rendu graphique – le chemin proposé et le remplit. Pour ne pas remplir, il suffisait de la remplacer par la fonction `stroke()`.

La fonction `closePath()` aurait pu être invoquée entre la ligne 8 et la ligne 9. Elle est généralement omise car provoquée par `fill()` ou `stroke()` qui ferme le chemin ouvert par un précédent `beginPath()`.

### Astuce :


en précisant les coordonnées du centre de l'arc, l'éditeur affiche les paramètres disponibles pour la fonction `arc` :

```

context.arc(50, 20, 3, 0, 2*Math.PI);
context.fill(
  arc(x: number, y: number, radius: number,
    startAngle: number, endAngle: number, anticlockwise?: bool)
)

```

On voit d'ailleurs que le paramètre `anticlockwise` est marqué d'un ? car, comme dans l'exemple, il est généralement omis pour conserver le sens horaire par défaut.

ProgLab propose quelques informations sur le dessin : onglet , rubrique Javascript, Graphiques



**Référence** : [Objet Canvas](#) (anglais) pour avoir la liste des propriétés et des fonctions pour dessiner : couleurs, styles et ombres, rectangles, chemins et courbes, textes, transformations, images et gestion bitmap (point par point), transparences ..

## VII. Gérer le clavier

La gestion du clavier n'est pas prévue dans ProgLab.fr donc ce n'est pas expliqué dans son aide rapide.

En fait, il s'agit ici de détourner la gestion mise en place par le navigateur pour répondre aux événements du clavier quand la page a été affichée : une touche enfoncée (`keydown`), une touche relâchée (`keyup`) voire une touche pressée et relâchée (`keypress`), qui est utilisée moins fréquemment car elle ne permet pas de tester directement les touches mortes (`SHIFT`, `ALT`, ...)

Voici le code pour un « télécran » répondant à cette consigne : faire une ligne en déplaçant un point à l'écran grâce aux touches fléchées.

```
1 //Télécran
2 // Sortie: point de départ en haut à gauche
3 // Cliquer 1 fois dans la zone des sorties,
4 // puis utiliser les touches fléchées du clavier.
5 //
6 //Codes des touches fléchées
7 var FLECHE_GAUCHE = 37;
8 var FLECHE_HAUT = 38;
9 var FLECHE_DROITE = 39;
10 var FLECHE_BAS = 40;
11 //Zone de dessin
12 var canvas = document.body.appendChild(document.createElement("canvas"));
13 var context = canvas.getContext("2d");
14 canvas.width = 400; // Largeur en pixels
15 canvas.height = 300; // hauteur en pixels
16 //Couleur de remplissage : rouge
17 context.fillStyle = "red";
18 //Position de départ : en haut à gauche
```

```

19 var posX = 0;
20 var posY = 0;
21 //Surchage de l'événement touche enfoncée
22 document.onkeydown = function (evenement) {
23     //on récupère les informations utiles sur l'événement
24     //la gestion dépend du navigateur d'où ce code magique
25     //pour récupérer l'objet à utiliser :
26     //le paramètre local evenement ou l'objet global window.event
27     var winObj = evenement || window.event;
28     //dont le n° de la touche
29     var codeTouche = winObj.keyCode;
30     //surchage juste pour le code touche concerné
31     if (codeTouche >= FLECHE_GAUCHE && codeTouche <= FLECHE_BAS) {
32         //suivant le code on change de position courante
33         if (codeTouche == FLECHE_GAUCHE) posX--;
34         if (codeTouche == FLECHE_HAUT) posY--;
35         if (codeTouche == FLECHE_DROITE) posX++;
36         if (codeTouche == FLECHE_BAS) posY++;
37         //on dessine
38         context.fillRect(posX, posY, 2, 2);
39     }
40 };

```

**Attention à l'exécution** : une fois le code exécuté, **il faut cliquer une fois dans la zone de sortie** pour que l'éditeur de ProgLab perde sa priorité sur la gestion du clavier (gestion faite en Javascript !).

En dehors du code magique ligne 27 et 29 pour récupérer les codes des touches impliquées, le code source est court et clair : suivant la touche enfoncée, la position du point est déplacée (`posx` et `posy` augmentés ou diminués) puis le point dessiné (rectangle plein par `fillRect`).

Les codes numériques des touches sont déclarés au début pour plus de lisibilité. La liste correspond au codage ASCII : Entrée 13, ... A 65 ... les chiffres du clavier alphabétique vont de 48 (0) à 57 (9), les chiffres du pavé numérique de 96 (0) à 105 (9) ... A toutes fins utiles, voici un programme qui renvoie le code d'une touche :

```

1 //Codes Clavier : appuyer sur une touche
2 //
3 //affiche la combinaison CTRL ALT SHIFT si enfoncées
4 //suivi du code de la touche enfoncée
1 document.onkeydown = function (evenement) {
2     var winObj = evenement || window.event;
3     var codeTouche = winObj.keyCode;
4     //
5     var avecAlt = winObj.altKey;
6     var avecCtrl = winObj.ctrlKey;
7     var avecShift = winObj.shiftKey;
8     //
9     if(avecCtrl) proglab.print("Ctrl+");
10    if(avecAlt) proglab.print("Alt+");
11    if(avecShift) proglab.print("Shift+");
12    //
13    proglab.println(codeTouche);
14 };

```

## VIII. Gérer la souris

La gestion de la souris n'est pas prévue dans ProgLab.fr donc ce n'est pas expliqué dans son aide rapide.

Il s'agit de détourner la gestion mise en place par le navigateur pour répondre aux événements souris : souris qui

se déplace (mousemove), bouton enfoncé (mousedown) ou bouton relâché (mouseup).

Voici le code pour laisser la trace du passage de la souris dans une zone de dessin, avec une zone texte pour afficher les coordonnées renvoyées par la souris ou un message d'information lors du clic souris.

```
1 // Dessine un point au survol de la souris
2 // clic souris : change la couleur : rouge ou blanc
3 //
4 // Zone texte pour afficher des informations :
5 // les coordonnées de la souris, changement de couleur
6 var texte = document.body.appendChild(document.createElement("input"));
7 // Zone de dessin
8 var canvas = document.body.appendChild(document.createElement("canvas"));
9 var context = canvas.getContext("2d");
10 canvas.width = 400; // Largeur en pixels
11 canvas.height = 300; // hauteur en pixels
12 // Styles
13 // Encadré du canevas (code CSS)
14 canvas.style = "border:1px solid #000000;";
15 //bleu pour le trait, rouge pour le remplissage
16 context.strokeStyle = "#0000ff";
17 context.fillStyle = "#ff0000";
18 //Quand la souris se déplace sur le canevas
19 canvas.onmousemove = function (evenement) {
20     var rect = canvas.getBoundingClientRect();
21     var ax = evenement.clientX - rect.left;
22     var ay = evenement.clientY - rect.top;
23     texte.value = "(" + ax + ";" + ay + ")";
24     context.fillRect(ax, ay, 2, 2);
25 };
26 //Quand le bouton de la souris est relâché
27 canvas.onmouseup = function (evenement) {
28     if (context.fillStyle !== "#ff0000") {
29         context.fillStyle = "#ff0000";
30         texte.value = "couleur:rouge";
31     } else {
32         context.fillStyle = "#ffffff";
33         texte.value = "couleur:blanc";
34     }
35 };
36 //
37 texte.value = "Passez sur le dessin";
38 //texte.type="hidden";
```

Contrairement au clavier où le gestionnaire clavier du document a été détourné, ce n'est pas le gestionnaire souris du document qui est détourné mais le gestionnaire souris de la zone de dessin qui est surchargé. Ainsi l'événement ne provoque de réaction qu'au survol de la zone de dessin.

En réponse à un clic, la bascule de couleur se fait en testant la valeur de la couleur de remplissage en cours, ce qui se fait avec la valeur numérique RGB en hexadécimal `rrggbb`, Dans le script, le `#` précède un nombre pour pour indiquer à l'interpréteur Javascript qu'il est en hexadécimal (il faut pouvoir distinguer l'entier décimal 16 `#16` qui vaut en décimal 22 et non 16) . Ainsi rouge est codé en hexadécimal `FF0000` car il y a une composant rouge maximale (`FF`) et pas de vert (1<sup>er</sup> paquet `00`) ni de bleu (2<sup>nd</sup> paquet `00`).

#### Astuce :

Le dernier commentaire permet de cacher (`hidden`) la zone texte. Cette zone texte affiche des informations en continu lors des mises au point. C'est plus commode que d'utiliser des messages avec la fonction `alert()` qui

bloque le fonctionnement de l'interface.

On pourrait aussi utiliser l'objet `console` et notamment `console.log(message)` mais pour cela il faut ouvrir en plus la console de débogage Javascript du navigateur : ce n'est pas l'objet de ce document.

## IX. Faire de la géométrie dynamique

[ProgLab.fr](http://ProgLab.fr) utilise la bibliothèque [JSXGraph](#) réalisée par l'Université de Bayreuth, libre et gratuite à des fins éducatives. Cette fonctionnalité permet de construire et de récupérer des informations sur une construction.

```
1 // Initialisation du graphique.
2 // 500x300 pixels, pas d'axes ni de grille.
3 proglab.initJSXGraph('box', 500, 300);
4 // pas d'axes ni de grille, abscisses de -5 à 5, ordonnées de -3 à 3.
5 var board = JXG.JSXGraph.initBoard('box', { boundingbox: [-5, 3, 5, -3] });
6 // Deux points A et B, un cercle et un segment.
7 var pointA = board.createElement('point', [0,0], {name:'A',size: 4,
8   face: 'o'} );
9 var pointB = board.createElement('point', [2,-1], {name:'B',size: 4, face:
10  '+'});
11 var cercleAB = board.createElement('circle', ['A','B'], {
12  strokeColor:'green' });
13 var segmentAB = board.create('segment', ['A','B'], { dash:2 });
14 // Deux textes sur le graphiques
15 // l'un affiche la longueur AB obtenue via JSXGraph
16 // l'autre affiche la longueur AB calculée avec les coordonnées
17 var texte1 = board.create('text', [3,0,function() {return
18  "AB="+segmentAB.L()}}]);
19 var texte2 = board.create('text', [3,-0.5,function() {return
20  "AB="+Math.sqrt(Math.pow(pointB.X()-pointA.X(),2)+Math.pow(pointB.Y()-
21  pointA.Y(),2))}}]);
```

La 1ère ligne est spécifique à ProgLab : elle permet de créer une zone où JSXGraph pourra dessiner dans un espace de 500 pixels par 300 pixels.

Pour le reste il s'agit de Javascript et de fonctions/syntaxes propres à la bibliothèque JSXGraph.

**Références** : la [documentation ProgLab sur JSXGraph](#) et le [Wiki de JSXGraph](#).

Pour information, JSXGraph est le cœur de l'application [Sketchometry.org](http://Sketchometry.org) disponible sur tout terminal informatique, du PC au téléphone mobile, tactile essentiellement mais pas exclusivement.

Pour publier un tel script dans une page HTML, il faut ajouter des appels à la bibliothèque JSXGraph et à sa feuille de style (mise en forme de la zone graphique).

Voir le IV.3. et l'archive `proglab_export_jsx.zip` pour n'avoir qu'à coller le script depuis ProgLab dans la page `index.html`.

Attention toutefois de veiller à enregistrer le fichier au format texte encodé UTF-8 pour gérer les caractères spéciaux convenablement. Voici le réglage de l'encodage à l'enregistrement dans le Bloc-Notes de Windows :

