40-JavaScript de base

introduction

Ce tutoriel présente l'utilisation de **JavaScript** dans un patcheur Max. JavaScript est un langage de script orienté objet développé à l'origine pour faciliter l'utilisation de logiciels intégrés dans les sites Web. L'objet Max *js* vous permet d'utiliser JavaScript comme langage au sein de l'environnement Max, en écrivant du code de programme pour votre propre objet sans avoir besoin d'outils de développement externes (compilateurs, débogueurs, etc.). Dans ce tutoriel, nous allons montrer un exemple simple d'utilisation de l'objet *js* pour créer un objet en JavaScript afin de répondre à une entrée numérique simple de Max et de renvoyer un nombre au patcheur.

L'objet *js* donne aux utilisateurs de Max la possibilité puissante d'écrire un objet à l'aide d'un langage de programmation intégré directement dans Max. En un mot, cela nous permet de :
• Concevoir et programmer des opérations procédurales qui peuvent être difficiles voire impossibles à mettre en œuvre en utilisant les objets Max par eux-mêmes. Il peut s'agir d'opérations nécessitant une récurrence ou qui répondent à des messages avec un nombre inconnu d'arguments, pour donner deux exemples.
• Créez des objets qui répondent à des messages personnalisables et s'appuient sur leurs propres structures de données internes.
• Planifiez des événements programmés en réponse à des messages.
• Gérez les variables globales à utiliser entre plusieurs objets *js* ou entre des objets *js* et le patcheur Max.
• Interface avec la puissante architecture de script de Max.
• Accédez au système de fichiers de votre ordinateur pour rechercher des fichiers par nom et par type.

Afin de développer un objet Max en utilisant un langage de programmation autre que Max lui-même (c'est-à-dire des subpatchers), vous avez plusieurs options. Vous pouvez développer votre propre objet externe en C à l'aide du kit de développement logiciel Max. Vous pouvez également développer des objets en Java à l'aide de l'objet *mxj*. Ces deux solutions requièrent que le code que vous écrivez soit compilé dans un format exécutable par Max (soit directement en chargeant une bibliothèque partagée, soit en exécutant le code Java via la machine virtuelle Java). Avec l'objet *js* (et son cousin graphique, *jsui*), le code est évalué sous forme de un script directement par l'ordinateur lorsque vous exécutez votre patch, ce qui permet un retour d'information plus immédiat sur le comportement du mini-programme écrit pour votre objet *js*. Cela ne veut pas dire que *js* ne vérifie pas d'abord les erreurs ; il le fait, et nous examinerons certaines des manières dont l'objet *js* peut vous aider à détecter les erreurs dans votre programme.

L'objet Max *js* utilise la version 1.8.5 du langage JavaScript, un surensemble spécifique à Mozilla d'ECMAScript 5. Bien que nous ne présumions pas d'une connaissance spécifique de JavaScript dans ces tutoriels, plus vous comprendrez le langage, plus il vous sera facile de développer du code. La référence définitive pour la version de JavaScript que nous utilisons dans Max se trouve à l'URL suivante :

https://developer.mozilla.org/en-US/docs/Web/JavaScript/New_in_JavaScript/1.8.5

Il est à noter qu'il existe de nombreuses variantes du langage JavaScript, ainsi que de nombreuses extensions du langage spécialement conçues pour son utilisation avec les navigateurs Web. L'objet Max *js* ne supporte que le langage JavaScript de base (comme indiqué dans l'URL ci-dessus), ainsi que certaines extensions ajoutées au langage pour prendre en charge l'interfaçage avec Max.

Pour ouvrir le patch du didacticiel, cliquez sur le bouton vert **Ouvrir didacticiel** dans le coin supérieur droit de la fenêtre de documentation.

JavaScript en action

Jetez un coup d'œil au patcheur du tutoriel. Si vous souhaitez écouter la lecture MIDI du patch, sélectionnez un périphérique de sortie MIDI valide en double-cliquant sur un objet *noteout* en bas à gauche du patcheur.

Cliquez sur le *toggle* en haut du patch pour démarrer l'objet *metro*. L'objet *float* envoie une valeur dans l'objet *js* à chaque tick du *metro*. Incrémentez lentement la boîte de *nombres* à virgule flottante attachée à l'entrée droite de l'objet *float*. La sortie de l'objet *js* changera en réponse. Le nombre de sortie est affiché à l'aide de l'objet *multislider* du patch et génère des notes MIDI.

Au fur et à mesure que vous augmentez la valeur entrant dans l'objet *js* au-dessus de **3.0,** la valeur générée par notre objet *js* commencera à ***osciller*** entre une valeur élevée et une valeur faible. À des valeurs plus élevées (par exemple au-dessus de **3,5**), la sortie de deviendra chaotique.

L'objet *js* de ce patch simule une fonction chaotique simple appelée ***équation logistique de population***. La formule de base pour la fonction est la suivante :

$f(x) = rx(1-x)$
où **r** est la valeur d'entrée actuelle et **x**, la valeur de sortie précédente.

Double-cliquez sur l'objet *js* dans le patch du didacticiel. Un éditeur de texte apparaîtra, contenant le code source de l'objet *js* dans le patcheur. Ce code est enregistré dans un fichier appelé «popu.js» situé dans le même dossier que le patch du didacticiel.

L'objet Max *js* vous permet de modifier le code JavaScript directement dans Max via un éditeur de texte de base (le même éditeur de texte que vous utilisez lors de la modification du contenu d'un objet *coll* ou d'un objet *text*). Le code source que l'objet *js* charge est déterminé par le premier argument de l'objet, qui spécifie un fichier texte dans le chemin de recherche de Max. Si vous ne donnez pas d'argument à *js*, vous pouvez toujours écrire un programme JavaScript à partir de zéro dans l'éditeur, mais vous devrez enregistrer le fichier pour pouvoir l'utiliser.

Le code global

Lorsque vous ouvrez l'éditeur de l'objet *js*, vous voyez du code JavaScript. Le code commence par un bloc de commentaires qui nous indique le nom du fichier, ce qu'il fait et ce qui l'a écrit. L'objet *js* ignore ces lignes car elles sont précédées d'une double barre oblique ('//'), couramment utilisée en C ++ (et dans d'autres langages de programmation) pour définir un commentaire. Les commentaires de style C ("/ *" au début et "* /" à la fin) sont également autorisés en JavaScript.

La programmation qui suit le bloc de commentaires initial définit un code global pour l'objet *js*. Il s'agit du code qui nous nous permettra de définir des variables et d'exécuter toute partie du programme dont nous avons besoin avant que quoi que ce soit n'arrive à l'objet dans l'environnement Max. Dans notre exemple, nous utilisons le code global pour indiquer à Max le nombre d'entrées et de sorties dont nous avons besoin pour notre objet *js* et pour définir et initialiser une variable (appelée **x**):

// entrées et sorties
entrées = 1;
sorties = 1;
// variables globales
var **x** = *0,66*;

Les mots clés ***entrées*** et ***sorties*** indiquent à Max le nombre d'entrées et de sorties que doit avoir notre objet *js*. La boîte de l'objet sera mise à jour lorsque ces variables seront modifiées.

Le mot-clé ***var*** indique à JavaScript que l'étiquette qui le suit doit être déclarée comme une variable, à laquelle nous attribuerons ensuite une valeur quelconque, soit dans la déclaration elle-même (comme nous le faisons ici, en disant que **x** est égal à **0,66**) soit plus tard dans le code. Notre nouvelle variable, **x**, a une portée ***globale***; nous verrons plus loin ce que cela signifie exactement.

Afin de rendre permanentes toutes les modifications apportées à notre code JavaScript, nous devons enregistrer le code dans l'éditeur de texte. Lorsque vous enregistrez les modifications (en sélectionnant **Enregistrer** dans le menu **Fichier** avec l'éditeur de texte à l'avant), Max vous indique qu'il a mis à jour l'objet *js* et signale tout problème qu'il a pu rencontrer avec votre code.

Tapez la ligne suivante sous l'instruction 'outlets = 1;' dans le code:
post ("*Salut à tous* !!!");

Enregistrez le code dans l'éditeur de texte. La console Max devrait avoir imprimé ce qui suit:
js • Salut à tous !!!

Le message affiché dans la console Max montre la sortie de l'instruction post() que nous avons insérée. L'instruction post() imprime ses arguments dans la console Max, vous permettant de faire exactement ce que l'objet print fait dans js.

Ajoutez la ligne suivante juste en dessous, où nous initialisons **x**, pour qu'elle soit égale à **0.66**:
post(x);

Lorsque nous sauvegardons le code, notre console Max nous indique maintenant:
js • Salut !!! *0,66*

Si vous donnez à **post ()** une chaîne de texte (entre guillemets), il l'imprimera. Les autres lettres sont interprétées comme des noms de variables. Dans ce cas, nous avons demandé à *js* de nous indiquer la valeur de la variable **x**, que nous avions initialisée à *0.66* à la ligne précédente. Notez que **post ()** ne met pas de retour chariot à la fin de la ligne; pour ce faire, nous pouvons placer une instruction **post ()** sans argument.

Ajoutez cette ligne quelque part entre les deux instructions **post ()** que nous avons ajoutées:
post ( );

Notre console Max nous dit maintenant ceci quand nous sauvegardons le code:
js • Salut à tous !!!
0,66

Erreurs de manipulation

Supprimez les parenthèses de clôture (‘)’ de toutes les instructions post () que nous avons ajoutées jusqu’à présent. Sauvegardez le code JavaScript. La console Max devrait imprimer une erreur:

js • [**popu.js**] Javascript SyntaxError: erreur de syntaxe, line *15*
ligne source: *post* (;

Cela nous indique que nous avons commis un type d'erreur appelé ***erreur de syntaxe***. Cela signifie que nous avons écrit quelque chose d'illégal du point de vue de la syntaxe JavaScript; dans ce cas, nous avons enfreint la règle qui stipule que les parenthèses doivent être équilibrées. Des parenthèses, des crochets et des accolades mal assortis sont des causes corantes d'erreur de syntaxe. Heureusement, *js* tente d’isoler la ligne sur laquelle l’erreur s’est produite (la ligne qui vous est citée peut être légèrement différente, selon l’emplacement où vous avez placé vos déclarations **post ()**).

Allez à la ligne incriminée dans votre code JavaScript et fermez les parenthèses correctement. Lorsque vous enregistrez le code, tout devrait aller bien à nouveau.

Les fautes d'orthographe sont une autre cause fréquente d'erreurs en JavaScript. Réécrivez votre déclaration **post ()** de manière à ce que le mot "post" soit mal orthographié (n'hésitez pas à faire preuve de créativité ici). Enregistrez votre script. Quelque chose comme ça va apparaître:

js • Salut à tous !!!
js • [**popu.js**] Javascript ReferenceError: pst non défini, line *15*
js • erreur de *fonction* d'axlling **bang** [popu.js]

Une erreur de référence signifie que nous avons demandé à JavaScript d’exécuter une commande qu’il ne comprenait tout simplement pas. Bien que **post ()** soit dans son vocabulaire de commandes légitimes, **pst ()** ne l’est pas.

Notez que JavaScript a arrêté d’exécuter notre code global au point où l’erreur s’est produite. Dans le cas susmentionnée, les mots «Hi There !!!» ont été imprimés dans la console Max, mais pas la valeur de **x** (**0,66**). Cela nous donne un indice important sur l'endroit où se situe l’erreur (c’est-à-dire quelque part entre les deux). L'utilisation fréquente des instructions **post ()** dans la phase de développement de votre code *js* est tout aussi importante que l'utilisation des objets *print* et des points de contrôle pour le débogage dans Max. Vous pouvez toujours les retirer plus tard.

Corrigez votre orthographe et sauvegardez votre code. Passons au reste du script.

Définir des fonctions

Les objets Max s'interfacent entre eux grâce à l'utilisation de méthodes, qui sont des routines de code exécutées en réponse à des messages reçus aux entrées de l'objet. En JavaScript, ces méthodes sont définies comme des fonctions dans notre code, chacune d'entre elles répondant à un type différent de message Max entrant. Dans notre exemple d'objet *js*, deux fonctions sont définies. Ces fonctions (**msg_float ()** et **bang ()**) contiennent le code que *js* exécute lorsque notre objet reçoit dans son entrée un nombre à virgule flottante et un **bang**, respectivement.

Jetez un coup d'œil à la fonction **bang ()** en premier. Le code ressemble à ceci:

```
function bang ()
{
   post ("la population actuelle est");
   post (x);
   post ();
}
```

Cette fonction indique à *js* ce qu'il doit faire si nous lui envoyons un **bang** depuis notre patch Max. Elle imprimera la valeur actuelle de **x** dans la console Max avec une préface explicative. L'instruction **post ()** à la fin de la fonction indique à Max de placer un retour chariot dans la console Max afin que le prochain message imprimé commence sur une nouvelle ligne. Notez que notre fonction est entourée d'accolades. La suppression d'une de ces accolades entraînera une erreur de syntaxe.

Dans le patch du tutoriel, cliquez sur l'objet *button* relié à l'objet *js*. Notre fonction **bang ()** devrait fonctionner correctement. Regardez la fonction msg_float (). Voici le code:

```
function msg_float (r)
{
   x = r * x * (1.-x);
   outlet (0, x);
}
```

Notre fonction **msg_float ()** exécute la partie la plus importante de notre code JavaScript, qui consiste à exécuter une seule itération de notre formule. Notez que, contrairement à notre fonction **bang ()**, notre fonction **msg_float ()** a un argument (indiqué entre parenthèses par **r**). Cet argument indique à l'objet *js* d'affecter la valeur flottante entrant dans l'entrée de notre objet à la variable *r*. Nous pouvons ensuite utiliser cette valeur dans le reste de la fonction.

De manière générale, le nom de la fonction en JavaScript correspondra directement au nom du ***message*** auquel vous voulez l'appeler. Par exemple, nous répondons à un message **bang** avec la fonction **bang ()** de notre objet *js*. Une fonction appelée **beep ()** répondrait à un message Max commençant par le mot **beep**. Cependant, comme ***float*** et ***int*** sont des mots réservés en JavaScript, nous utilisons respectivement **msg_float ()** et **msg_int ()** pour définir les fonctions qui répondent aux floats et aux ints.

Le corps principal de notre fonction **msg_float ()** définit **x** comme résultat de la multiplication de **r** (la valeur entrant dans l'entrée), de l'ancienne valeur de **x** et de l'ancienne valeur de **x** soustraite de **1.0**. Cette instruction:

```
x = r * x * (1.-x);
```

est un exemple d'une caractéristique puissante de l'utilisation d'un langage de programmation intégré dans Max. Pour accomplir ceci avec l'objet *expr*, par exemple, nous devrions prendre la sortie de l'objet et le renvoyer à une entrée de manière à éviter une boucle de feedback dans Max, probablement à l'aide d'une boîte de *nombres*.

Double-cliquez sur le patcheur appelé **done_with_expr** pour voir comment vous vous y prendriez avec des objets Max normaux. Notez que parce que l'objet *expr* n'a aucune connaissance de sa valeur de sortie précédente, nous devons le stocker manuellement et le saisir à nouveau en utilisant une deuxième entrée de l'objet.

La deuxième ligne de code de notre fonction **msg_float ()** prend notre nouvelle valeur de **x** et l'envoie par la sortie de l'objet *j s* vers Max. La fonction outlet () prend comme arguments le numéro de la sortie à utiliser et l'information à envoyer. La numérotation des sorties commence à **0** pour la sortie la plus à gauche (et dans notre cas, la seule).

Essayez de relancer le patch en sachant comment le code fonctionne. Voyez si vous pouvez déterminer à quel moment l'équation devient chaotique et pourquoi.

Portée des variables

La clé du succès de notre code JavaScript réside dans l'utilisation de **x** en tant que variable globale. JavaScript, comme de nombreux langages de script, alloue dynamiquement les variables au fur et à mesure de leur utilisation, ce qui vous permettra d'utiliser de nouveaux noms de variables, sans devoir à les prédéfinir à l'avance. Ces variables dynamiques sont toutefois locales aux fonctions dans lesquelles elles sont utilisées. Par exemple, si nous avions une variable **i** dans notre fonction **msg_float ()**, notre fonction **bang ()** ne serait pas en mesure de l'utiliser. Par conséquent, nous pourrions utiliser **i** comme variable dans les deux fonctions indépendamment l'une de l'autre. Parce que nous avons explicitement défini **x** comme variable dans notre code global, **msg_float ()** (qui évalue **x**, lui donne une nouvelle valeur et l'envoie à notre patch) et **bang ()** (qui imprime la valeur actuelle de **x** dans la console Max) parlent de la même chose quand elles font référence à **x**.

Commentez la ligne:
var **x** = *0,66*;

en plaçant deux barres obliques («//») au début de celle-ci. Enregistrez votre script et recréez l'objet *js* ou rouvrez le patch du didacticiel. Essayez d'exécuter le patch.

Avec **x** indéfini, notre objet *js* signale des erreurs de référence lorsque vous lui envoyez un *float* ou un **bang**. Ceci est dû au fait qu'il essaie d'accéder à une variable qui n'a jamais été initialisée. Nous pourrions supprimer ces erreurs en définissant **x** à une certaine valeur dans chacune de nos fonctions, mais nous utiliserons alors **x** localement. Non seulement cela nous empêcherait de partager la valeur de **x** entre nos deux fonctions, mais cela réinitialiserait également **x** chaque fois que nous enverrions un flottant dans *js*, empêchant ainsi notre objet de maintenir **x** sur plusieurs itérations de la fonction.

Supprimez le commentaire de variable pour **x**. Tout devrait rentrer dans l'ordre lorsque vous enregistrez votre script.

Résumé

L'objet *js* est un outil puissant qui vous permet d'écrire du code en utilisant un langage de programmation standard dans Max. Vous définissez les entrées, les sorties et les variables globales dans la section du code global du script (qui réside en dehors de toute fonction spécifique). Les méthodes qui répondent à des messages particuliers (floats, **bang,** messages, etc.) sont écrites en tant que fonctions sous le code global et sont exécutées en réponse aux messages pertinents de l'environnement Max. Vous pouvez utiliser la fonction **post ()** pour imprimer des informations dans la console Max (de la même manière que vous utiliseriez l'objet *print* dans un patch), et vous pouvez utiliser la fonction **outlet ()** pour renvoyer des messages de votre objet *js* au patch Max le contenant. Vous pouvez écrire du code JavaScript directement dans l'éditeur de texte de l'objet *js*; lorsque vous enregistrez un script modifié, l'objet *js* analysera votre code à la recherche d'erreurs de programmation telles que des erreurs de syntaxe et de référence, ce qui vous permettra de programmer et de déboguer votre code depuis Max.

Liste de code

```
// popu.js
//
// simule l'équation logistique de la population:
// f (x) = rx (1-x)
//
// l'entrée est r. la sortie est actuelle x.
//
// rld, 5.04
//
// entrées et sorties
inlets = 1;
outlets = 1;
// variables globales
var x = 0,66;
// float - lance l'équation une fois
function msg_float (r)
{
    x = r * x * (1.-x);
   outlet (0, x);
}
// bang - affiche la population actuelle dans la fenêtre max
function bang ()
{
    post("la population actuelle est");
    post(x);
    post();
}
```