Firefox 学生マーケティングチーム主催

学生 Add-on 勉強会

2013 年 3 月 30 日（土）

@Mozilla Japan オフィス

目 次

#gakumoz

はじめに

言語として必要な知識は…

- HTML
- CSS
- JavaScript

これらがメインとなりますが、全く知らなったとしても、
調べながら作っていけば大丈夫です。

必要な環境

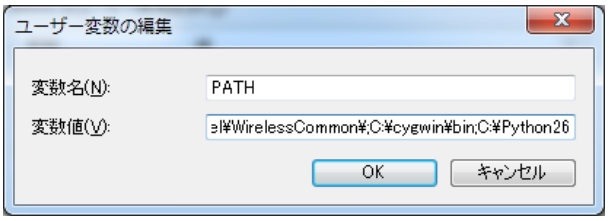
- Mozilla Firefox(最新版にしましょう)
- Python(2.5 または 2.6)
- Add-on SDK

Python のインストール方法

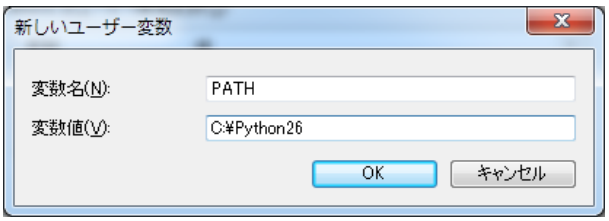
http://www.python.org/getit/releases/2.6/
(http://bit.ly/python26)
ここより各 OS のインストーラーをダウンロードしインストール
Windows の方はユーザー環境変数 PATH に Python のディレクトリを追加しましょう。

●設定方法

スタート＞コントロールパネル＞表示方法:アイコン＞システム＞システムの詳細設定＞詳細設定タブ＞環境変数(N)



◎ 既に PATH が設定されてる場合
ユーザー環境変数の一覧から PATH を選択し編集(E)をクリックし、既に入力されている値の後ろに ;C:¥Python26 と追加します。



◎ PATH が存在しない場合
ユーザー環境変数の一覧の下の新規(N)をクリックし、変数名に PATH
変数値に C:¥Python26 と入力します。

ターミナルやコマンドプロンプトで python と入力し、Python が起動できるか確かめましょう。

```
C:¥Users¥mozilla>python
Python 2.6 (r26:66721, Oct  2 2008, 11:06:43)
Type "help", "copyright", "credits" or "license" for more
information.
>>> quit()      /* python を抜ける */

C:¥Users¥mozilla>
```

Add-on SDK のインストール方法

https://dev.mozilla.jp/addon-sdk-docs/dev-guide/tutorials/installation.html (http://bit.ly/addon-install)
ここより各 OS のインストーラーをダウンロード

MacOSX や Linux の場合

適切な場所で展開

```
tar -xf addon-sdk.tar.gz
cd addon-sdk
```

アクティベート

```
cd addon-sdk-1.10
source bin/activate
```

Windows の場合

適切な場所で展開
(コマンドラインでは標準対応していないのでダブルクリック)
アクティベート

```
cd addon-sdk-1.10
bin¥activate
```

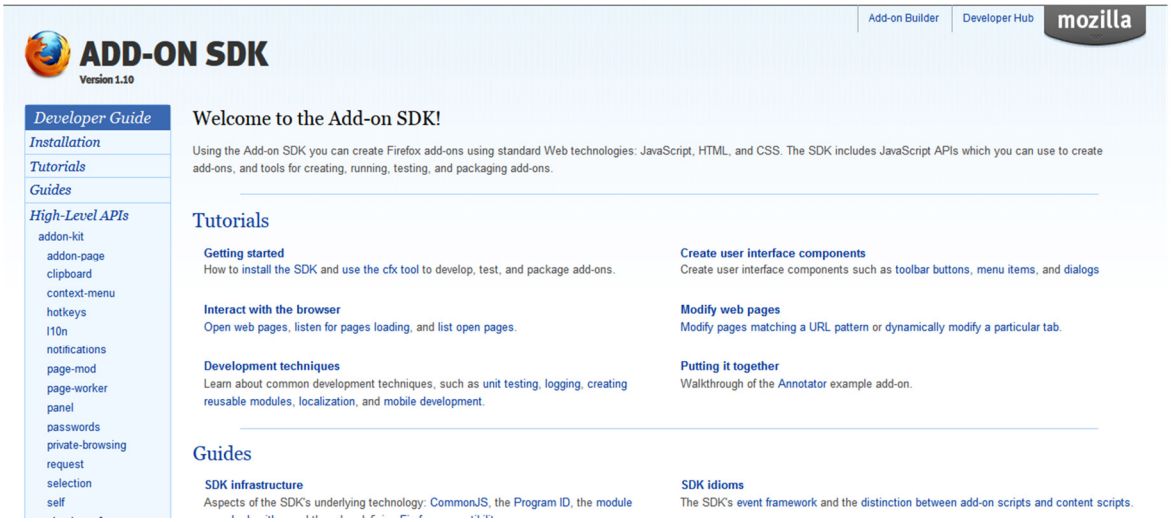
インストールした結果ちゃんと動いているかをチェックするため、

```
cfx
```

と打って、大量の情報が出力されれば OK。

```
cfx docs
```

と打つと、以下と同じ英文のドキュメントがブラウザに表示されます。



実際に動かしてみよう

すでにアクティベートしているので、適当な作業ディレクトリを作り、移動します

```
cd ../
mkdir ディレクトリ名
cd ディレクトリ名
```

まずは、アドオンの最初の手順です。

```
cfx init
```

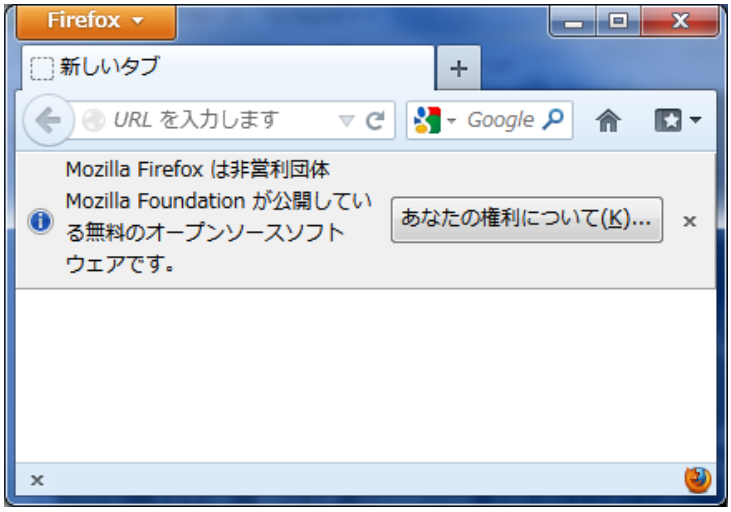
このコマンドで基本的なアドオンのファイル構造が生成されます。
このままでは何もしないアドオンなので、サンプルコードを lib/main.js に書き込みます。

```
const widgets = require("widget");
const tabs = require("tabs");

var widget = widgets.Widget({
  id: "mozilla-link",
  label: "Mozilla website",
  contentURL: "http://www.mozilla.org/favicon.ico",
  onClick: function() {
    tabs.open("http://www.mozilla.org/");
  }
});

console.log("The add-on is running.");
```

次にアドオンとして動かしてみましょう。

```
cfx run #初めて run をする場合は ID が割り当てられるので再び run
cfx run
```

すると…
右のようなアドオンバーの右下に Firefox のロゴが表示されました。
クリックしてみましょう。



もし、cfx run 実行時にターミナルがおかしくなったら、Ctrl＋C を押してください。

解説すると

```
const widgets = require("widget"); /* widget の API を使用 */
const tabs = require("tabs"); /* tabs の API を使用 */

var widget = widgets.Widget({
  id: "mozilla-link", /* 唯一の ID を割り当てる */
  label: "Mozilla website", /* マウスオーバーしたときのラベル名 */
  contentURL: "http://www.mozilla.org/favicon.ico", /* 画像の URL */
  onClick: function() {
    tabs.open("http://www.mozilla.org/"); /* クリックしたときの挙動 */
  }
});

console.log("The add-on is running."); /* デバッグ用コンソールへの表示 */
```

画像は自分の用意したアドオン組み込んだファイルも使用可能(後述)

それではアドオンとして配布可能な形にしましょう。

```
cfx xpi
```

作業ディレクトリに**ディレクトリ名**.xpi が生成されました。
これを Firefox にドラッグ・アンド・ドロップすると、自分の作ったアドオンをインストールできます。

API ってなに?

Application Programming Interface の略。つまり、関数を使用するのに必要なライブラリです。

この Add-on SDK にはさまざまな API が用意されていて、先ほど使用した widget や tabs も API です。https://dev.mozilla.jp/addon-sdk-docs/dev-guide/tutorials/ (http://bit.ly/gaku-addon) に載っている左のサイドバーが API の一覧です。

ディレクトリ構造について

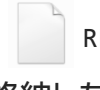

data ここに画像や css などを格納


doc


lib ここに main.js でメインのスクリプト


test


package.json 作者情報やバージョンなどを記述


README.md マークダウンファイル Github に上げる時など

※ data に格納したファイルを読み込むときは self API を使用してください。(後述)

Ex.) package.json の使用法

```
{
    "name": "gakumoz", /* 出力されるファイル名。小文字のみ */
    "license": "MPL 2.0", /* ライセンスの明記 */
    "author": "Firefox Student Marketing Team", /* 作者名 */
    "version": "1.0.1", /* バージョン */
    "fullName": "Gakusei Add-on Benkyokai", /* アドオン名称 */
    "id": "jid1-moy0RDGTKNWZww", /* ここは触らない */
    "description": "Gakusei Add-on Benkyokai sample",/* 説明 */
    "homepage": "http://atnd.org/events/34545" /* Web */
}
```

右図のようにインストールしたときに表示されます。

アドオンを作る上での注意点

- 他人の作品を侵害しない(スクリプトや画像など)
- 公序良俗を守る
- プライベートブラウジング機能時に履歴やキャッシュを収集しない

これらが挙げられます。ルールを守って楽しく開発しましょう。

Tips

他の人に作ったアドオンを使ってもらいたい

是非、`https://addons.mozilla.org/` (AMO)にユーザー登録して、申請しましょう。
世界中に公開できるチャンス。

ほかに参考になるページは?

modest (Mozilla Developer Street)の `https://dev.mozilla.jp/project/addons/` にアドオン開発者ガイドがあります。

技術を向上させたい

`https://developer.mozilla.org/ja/` にドキュメントや学習コーナーなどが設けられています。

Supplement

Add-on SDK 超最低限コマンドリスト!

○ Windows

コマンド	使い方	意　味
cd	> cd [ディレクトリ名]	[ディレクトリ名]のディレクトリの中に移動する
dir	> dir	ディレクトリの中身を見る
mkdir	> mkdir [ディレクトリ名]	[ディレクトリ名]のディレクトリを作成する
bin/activate.bat	> bin/activate.bat	Add-on SDK を起動する。 (addonsdk ディレクトリ内)

○ OSX / Linux

コマンド	使い方	意　味
cd	$cd [ディレクトリ名]	[ディレクトリ名]のディレクトリの中に移動する
ls	$ls	ディレクトリの中身を見る
mkdir	$mkdir [ディレクトリ名]	[ディレクトリ名]のディレクトリを作成する
source	$source bin/activate	Add-on SDK を起動する。 (addonsdk ディレクトリ内)

○ Add-on SDK

コマンド	使い方	意　味
cfx run	cfx run	作ったアドオンを実行してみる
"cfx" --version	"cfx" --version "cfx" -version (Windows)	Add-on SDK のバージョンを見る

例題① Self API の説明

ディレクトリ構造についてはもう説明しましたが、その data ディレクトリに格納したファイルを呼び出すときに使う API を説明します。

先ほど動かしたアドオンの画像をロゴマークから自分の用意した画像を表示させたい場合
（今回はこちら:favicon.png）

3 行目に

```
  var self = require("self");
```

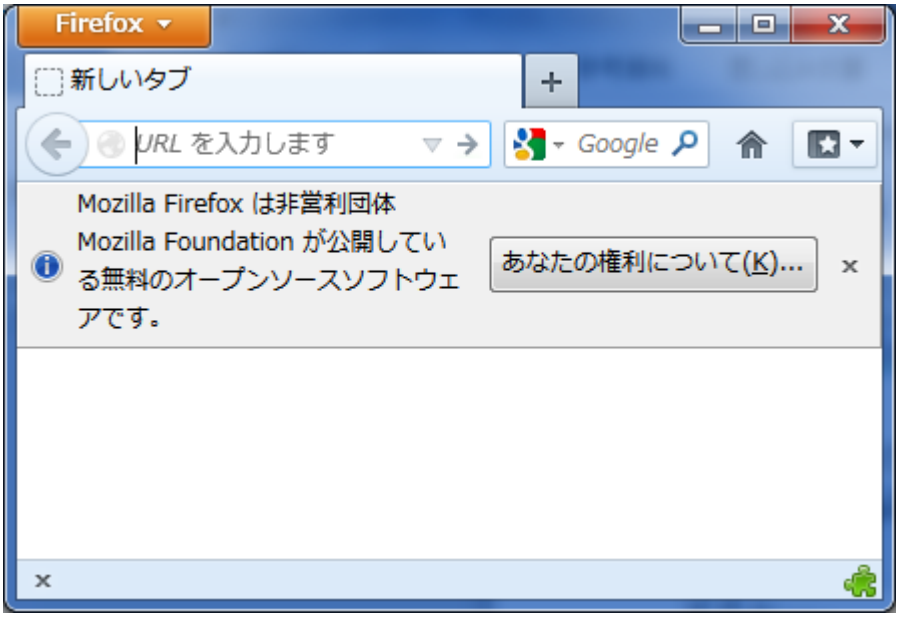
を追加して、この行

```
  contentURL: "http://www.mozilla.org/favicon.ico",
```

を、このように

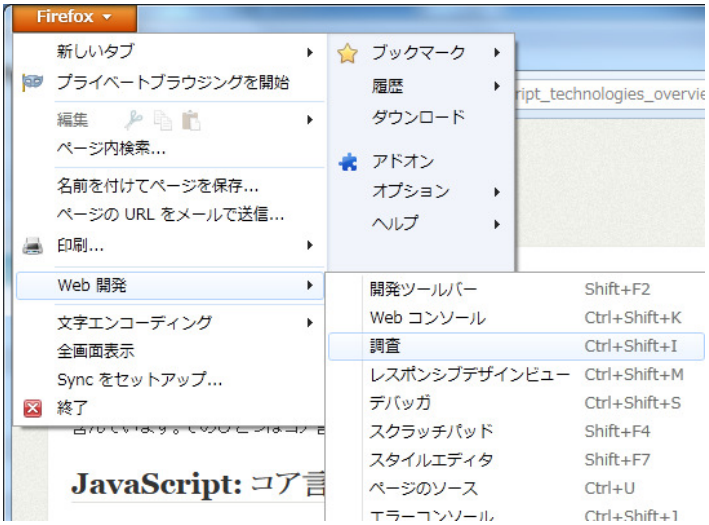
```
  contentURL: self.data.url("favicon.png"),
```

変更するだけです。
下図のように、機能はそのままでアイコンが変更されました。



例題②　2秒おきに画面が傾くアドオンを作ろう

CSS には表示を変形するプロパティがあります。
まず、それを試してみるためにメニューのツール＞Web 開発＞調査　の順に選択します。



Windows の場合

Ctrl ＋ ⇧Shift ＋ I

Mac の場合

Option ＋ ⌘Command ＋ I

「調査」の画面になったら、任意の要素をクリックします。

で囲った部分に以下の CSS を書き入れます。

```
transform: rotate(6deg);
```

この画面での入力時には「：や；」は入力せず、Enter キーを押してください。

すると、以下のように表示が回転します。

この挙動を自動で行うアドオンを作ります。

`main.js` に以下のコードを入力します。

```
var data = require("self").data;
var pageMod = require("page-mod");
pageMod.PageMod({
  include: "*",
  contentStyleFile: data.url("tilt.css"),
  contentScriptWhen: 'end',
  contentScriptFile: data.url("tilt.js"),
  contentScript: 'tiltA();'
});
```

`data` ディレクトリに `tilt.css` を作成します。

```
.tilt-A{
  transform: rotateZ(6deg);
}
.tilt-B{
  transform: rotateZ(-6deg);
}
```

data ディレクトリに tilt.js を作成します。

```
var tiltA = function() {
    document.body.classList.remove("tilt-B");
    document.body.classList.add("tilt-A");
    setTimeout(tiltB, 2000);
    }
var tiltB = function() {
    document.body.classList.remove("tilt-A");
    document.body.classList.add("tilt-B")
    setTimeout(tiltA, 2000);
    }
```

これを cfx run すると開いたすべてのページで表示が一定時間ごとに傾きます。

解説

page-mod API は特定のページでスクリプトを実行する API です。

```
var pageMod = require("page-mod");
```

この API の関数の引数は list で与えられていて、以下のように記述します。

```
pageMod.PageMod({
  include: "*", /* どのページで実行するか */
  contentStyleFile: data.url("tilt.css"),  /* CSS を指定する */
  contentScriptWhen: 'end', /* 実行するタイミング */
  contentScriptFile: data.url("tilt.js"), /* 実行するスクリプトファイル */
  contentScript: 'tiltA();' /* 実行するスクリプト */
});
```

CSS は表示の変形や色を変えたり、さまざまな指定をすることができます。
id や class を割り当て、HTML 要素にスタイルを付加します。
書き方は以下の通りで、

```
●●{
  プロパティ 1: 値 A;
  プロパティ 2: 値 B;
  プロパティ 3: 値 C;
}
```

●●には id (例 #foxkeh) や class (例 .mozilla) や HTML タグ(例 div)が指定でき、
カンマ区切りで複数の要素に対して指定することができます。
タグの中のタグを指定することもできます。

```
div section a{ …略… }
/* div の中の section の中の a の要素について */
```

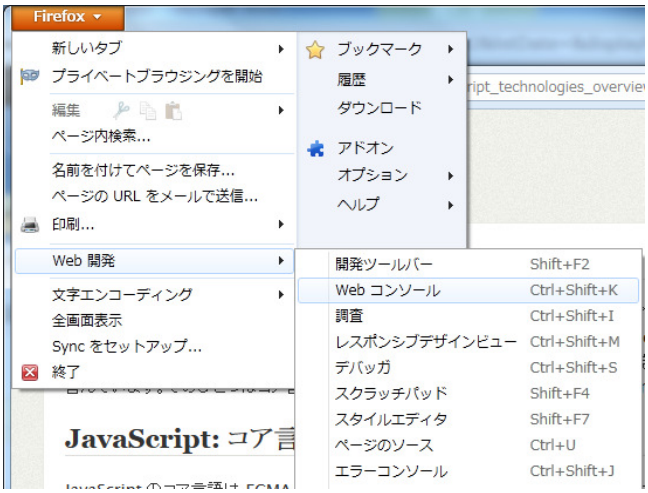
CSS プロパティは HTML タグにもよりますが、
例えばこんなものがあります。

display:none;	表示を(させない)	font-size:12px;	文字サイズ
color:red;	文字色	border-radius:4px;	角丸
width:600px;	幅	background:url('http://…');	背景

自作の関数を定義して、ページ表示時にその関数を実行する形になります。

```
/* body のクラスを削除する */
document.body.classList.remove("tilt-B");
/* body のクラスを付加する */
document.body.classList.add("tilt-A");
/* タイムアウトのミリ秒とその後の挙動を指定 */
setTimeout(tiltB, 2000);
```

JavaScript が動くかどうかをチェックするには、Web コンソールを使いましょう。
メニューのツール＞Web 開発＞調査 の順に選択します。



Windows の場合

Ctrl + ⇧Shift + K

Mac の場合

Option + ⌘Command + K

一番下の入力欄でスクリプトを実行することができます。

Web ページからアドオンをインストールする

```
<script type="application/x-javascript">
<!--
function install (aEvent)
{
  var params = {
    "Foo": { URL: aEvent.target.href,
             IconURL: aEvent.target.getAttribute("iconURL"),
             Hash: aEvent.target.getAttribute("hash"),
             toString: function () { return this.URL; }
    }
  };
  InstallTrigger.install(params);

  return false;
}
-->
</script>

<a href="http://www.example.com/foo.xpi"
  iconURL="http://www.example.com/foo.png"
  hash="sha1:28857e60d043447c5f4550853f2d40770b326a13"
  onclick="return install(event);">拡張機能をインストール!</a>
```

以上のようにWebページを書くとドラッグ・アンド・ドロップではなく、クリックするだけでインストールすることができます。

出典:Web ページから拡張機能とテーマをインストールする
https://developer.mozilla.org/ja/docs/Installing_Extensions_and_Themes_From_Web_Pages

[付録] JavaScript について

Q. 計算したい!

A. 計算用の記号が用意されています。

+	足し算	1 + 2	結果は 1 + 2 の結果である 3
-	引き算	1 - 2	結果は 1 - 2 の結果である -1
*	掛け算	1 * 2	結果は 1 * 2 の結果である 2
/	割り算	5 / 2	結果は 5 / 2 の結果である 2.5
%	剰　余	5 % 2	結果は 5 / 2 の余りである 1

また計算をするときに ＋と－より×と÷が先に計算するように
プログラムにも優先順位が存在します。

```
① 1 + 2 * 3 / (4 - 5)
② 1 + 2 * 3 / -1
③ 1 + 6 / -1
④ 1 + (-6)
⑤ -5
```

計算をしたとおり -5 になります。

+ - * / % などのように、各種の演算をあらわす記号・符牒を演算子と呼びます。
他にも演算子がありますので、少し凝った計算をしたい方はこちらをご覧ください。

参考　とほほの JavaScript リファレンス － 演算子
http://www.tohoho-web.com/js/operator.htm
とほほの JavaScript リファレンス － 数学関数(Math)
http://www.tohoho-web.com/js/math.htm

Q. データを保存させたい!

A.『変数』を使いましょう。

変数とは、箱のようなものです。
この変数には、数値や文字などを入れることができます。
例

```
var a = 3;
```

var というのは、「これから変数を書く」という宣言になります。variable（変数）という意味です。
a は、変数の名前です。これは、宅配便のダンボール箱に貼る送り状のようなものです。
= は、この変数に、右辺の値を代入するという意味になります。
通常の数学の ＝（イコール）とは違う使い方なので、注意が必要です。
最後の ；（セミコロン）は、「この式はここで終わり」という意味になります。

プログラムでは、 = の右側の値を、左側の変数に代入します。
上記の式では、数字の 3 は、変数に入れる値になります。

引用

http://crocro.com/write/manga_javascript/wiki.cgi?p=%BF%F4%C3%CD%CA%D1%BF%F4%A4%C8%B7%D7%BB%BB

つまり、`var a = 3;` は `3` という数字のデータが入った箱(変数) `a` を作成したということになります。さらに例文をあげてみます。

```
var a = 5;
var b = a;
```

このプログラムの場合、`5` が入った `a` という箱を作成しました。

```
var b = a;
```

この場合 `a` の箱には `5` のデータが入っているので中身を取り出します。

```
var b = 5;
```

`a` に入っていた数字が `b` にも代入されました。
この時点で `a` と `b` の箱には `5` のデータが入っていることになります。さらに例文をあげてみます。

```
a = 2;
a = a + 3;
```

`+` の意味は算数でおなじみの足し算をするという意味です。
さて、この場合どうなるでしょうか?

```
a = a + 3;
```

この式は成り立たない!と思った方、残念ですが少し違います。
この場合は `a = a + 3;` は、左側の変数 `a` に、「元の変数 `a` に `3` を足した値を代入」という計算です。つまり、

```
a = 2 + 3;
```

となり、結果は `a = 5` となります。

備考

文字データを入れることも可能です。

```
var a = "このようにダブルクォーテーションを囲むことによっていれることができます";
```

注意事項

変数の名前には半角の英数字、および _ (アンダーバー)が利用できます。
ただし、最初の 1 文字目に数字を使用することはできません。

Q. プログラムの中にメモを残したい!

A. `//` や `/* ～ */` を使うことで残すことができます。

プログラムの文字だけでは分かりづらい、といったことは実際プログラムを作っているとよくあることです。

そこでプログラムの説明を補助するために、コメントアウトといった機能があります。

例えば

```
a = a + 3; // 左側の変数『 a 』に、「元の変数『 a 』に 3 を足した値」を入れる
```

このようにプログラムに対して説明文を入れることができます。

`//` を書くとその行はプログラムと判断されなくなります。

`/* ～ */` と書くと、間の～部分はプログラムと判断されなくなり、自由に記入することができます。

```
var a = 3; // 例えばこの日本語の文章はプログラムに反映されません。
/*
ここの部分はコメントです。
この日本語の文章はプログラムに反映されません。
*/
// コメント以外で日本語文字が入っているとエラーになります。
// プログラムの説明にはコメントを使いましょう。
a = 4; 4 に入れなおす
```

Q. 臨機応変にプログラムを動かしたい!

A. if を使いましょう

英語で「もし～」を表すときに使う `if` です。以下のように書きます。

```
if (/* 条件式 */) {
    // A の処理
} else {
    // B の処理
}
```

`if` に続く (～)内の式が正しければ A の処理 が、間違っていれば B の処理 が実行されます。この 正しい・間違っている ということを、プログラムの世界では 真・偽 と言います。

たとえば `10 < 1` の式は間違っています。

この場合、この式の結果は 偽 であると言います。

逆に `1 < 10` であれば正しいので、真 であると言います。

プログラムでは `if (a >= 10)` のように、変数を利用して条件式を書くことで、

「変数 a が 10 以上」(真)なら A の処理 を実行させ、

「変数 a が 10 未満」(偽)なら B の処理 を実行させます。

この条件式に使った　< や >= のような記号を、『比較演算子』と言います。

条件分岐を行うためには、if の丸括弧の中で、真偽 の判定を行わなければなりません。そのために利用できるのが 比較演算子 です。演算子の左右を比較して、その演算子の条件に合っていれば 真 を、合っていなければ 偽 を返します。
代表的なものを、以下に示します。

演算子	説　明
左 == 右	左と右が同じなら真(『=』2 つなので注意)
左 != 右	左と右が違うなら真(『!』は否定記号)
左 < 右	左が右より小さいなら真
左 > 右	左が右より大きいなら真
左 <= 右	左が右より小さいか同じなら真
左 >= 右	左が右より大きいか同じなら真

論理演算子

if の丸括弧の中の条件判断ですが、時には複雑な判定を行いたい時もあります。
例えば、「点数が 60 点以上、70 点未満の場合」といった条件があったとします。
この条件は、if を 2 つ使い、以下のように書きます。

```
var tensuu = 65;
if (tensuu >= 60) {
    if (tensuu < 70) {
        alert("60 点以上、70 点未満");
    }
}
```

しかし、見た目がちょっと複雑です。
このプログラムは、論理演算子 を使えば、もう少しシンプルに書くことができます。
論理演算子 は、A かつ B といった条件や、A または B といった条件を表す式です。

演算子	説　明
左 && 右	左であり、かつ右であるなら真
左 || 右	左または右なら真

これは、以下のように使います。

```
var tensuu = 65;
if ((tensuu >= 60) && (tensuu < 70)) {
    // ( tensuu >= 60 ) かつ ( tensuu < 70 )である
    alert("60 点以上、70 点未満");
}
```

この 論理演算子 を使えば、複雑な条件を短く書くことができます。

Q. 同じようなプログラムがずっと続いているので短縮したい!

A. ループという機能があります。

ループ処理 では、多くの場合、 `for` 文を利用します。

典型的な `for` 文では、以下のような処理を行います。

1. 実行回数を表す変数を用意する。
2. `{ }` 内のプログラムを 1 回処理するごとに、その変数の値を 1 増やす。
3. 指定の回数処理を行ったか、変数の値を確認する。
4. 指定の回数処理を行ったら、処理を終了する。

具体的な例で説明しましょう。

たとえば、変数 `i` に `1` という値を設定して、

1 回の処理ごとに `2`, `3`, `4`, …… というように `1` ずつ増やしていき、

`1000` を越えたら終了する、といったようなプログラムを書くとします。

このような処理を、 `for` 文を使って、プログラムとして書くと、以下のようになります。

```
var sum = 0;
for (var i = 1; i <= 1000; i = i + 1) {
    console.log(i+ "回目"); // コンソールというところに出力します
}
```

for の中身は `;` で区切って記入します。

記入するのは以下3つです。

`var i = 1;`	一回目に実行する処理を書く
`i < 1000`	if 分と同じように継続方法を書く
`i = i + 1`	{} 内の処理を実行した後に実行する処理を書く

参考

http://crocro.com/write/manga_javascript/wiki.cgi?p=%A5%EB%A1%BC%A5%D7%BD%E8%CD%FD

Q.プログラムが長くなって来たし、同じような処理が多いので楽をしたい!
A.プログラムには 何度も行う処理を 一度だけ書いて済ませる 関数というものがあります。

書き方はこのようになります。

```
function 関数名(引数) {
    内部処理
    return 戻り値;
}
```

引数や戻り値また、内部に処理がなくても構いません

```
function 関数名(引数) {}
```

引数には数値や文字列、変数を入れることができます。
プログラムの 別の場所で 関数名が書かれた時に 中身が実行されます。
また その際 引数 を受け取ったり 戻り値 を戻したり できます。
以下その例です。

```
var kekka = han(16);
kekka = han(kekka);
function han(no) {
    var res = no / 2;
    return res;
}
```

引数の値を半分にする関数を作っています。

`han(16);`	引数 16 の値を han 関数を呼び出しています
`function han(no = 16)`	no は変数で引数に 16 の値が入れられた状態です。 function の中に書かれたプログラムを実行します。
`var res = no / 2;`	res には 8 が入っています。
`return res;`	return は値もしくは結果を返します。
`var kekka = han(16);`	han(16)の結果は 8 になります。
`var kekka = 8;`	
`kekka = han(kekka);`	kekka 変数には 8 が入っているので、
`function han(no = 8)`	no には 8 の値を入れた状態で han 関数のプログラムを実行します。
	han(8)の結果は 4 になります。

このようにして、何度も使うような処理をまとめることができます。

Q. 便利な機能を使いたい!プログラムを再利用をしたい!
A. クラスやオブジェクト型変数を使えるようになりましょう!

ウェブ上にあがっている便利な機能や、自分で作ったプログラムが
数値や文字などをいれる変数の他に、オブジェクト型変数といったものがあります。
オブジェクト型変数は色々なものを扱うことができます。

例えば簡単に図などのグラフを扱えるものや HTML のタグ要素など色々あります。
既に用意されているもので良いのですが、自分で一から作成するためにはクラスという設計図がいります。
その設計図に書けることできる大きなことを簡単にまとめました。

- クラスを元に作成し、その設計図の通りに動かすことができます。
- フィールドと呼ばれるオブジェクト型変数ごとに変数を保持することができます。
- フィールドの操作に最適な関数、メソッドを持つことができます。

兄弟で、一つずつ別の貯金箱を持つことになりました。
弟は、貯金箱にお金を入れること(メソッド)を行いました。
貯金箱はお金を入れると残金を喋る貯金箱だったので、残金を喋りました。
弟の貯金箱の残金(フィールド)は変更され、
兄の貯金箱の中身は変わっていません。

以下そのプログラムを記します。

```
// オブジェクト型変数の生成
// new クラス名()
var ani = new Tyokinbako(100);
var otouto = new Tyokinbako(100);
// メソッドの使用
// オブジェクト型変数.メソッド名()
otouto.plusMoney(1000);
ani.nowMoneyAlert();
otouto.nowMoneyAlert();
// クラスの定義
// function クラス名(引数)
function Tyokinbako(startMoney) {
    // フィールドの定義(オブジェクト型変数が持つ変数)
    // this.フィールド名
    this.tyokin = startMoney;
    // メソッドの定義
    // this.メソッド名 = function()
```

```
    this.nowMoneyAlert = function() {
        alert('現在のお金は '+ this.tyokin + ' 円です。');
    }
    // 引数付きメソッドの定義
    this.plusMoney = function(money) {
        this.tyokin = this.tyokin + money;
        this.nowMoneyAlert();
    }
}
```

オブジェクト型変数は new クラス名()により生成されます。
そのオブジェクト型変数のメソッドを使うときは . でつないでメソッド名を記述します。
再利用がしやすいプログラムをどんどん作ってみてください!

Q. ページを書き換えたい!
A. getElementById を使おう!

まずその前に、HTML をご存知ですか?
HTML とはウェブ上のドキュメントを記述するための言語です。これを見ることによって現在参照しているサイトのページがどのように構成されているかを表示することができます。
HTML を見るためには、Firefox で見たいページで右クリックした後、「ソースを表示する」メニューを選択することによって、見ることができます。HTML は <> で囲まれたタグと呼ばれるワードを組み合わせて構成しています。
以下はその例です。

```
<!DOCTYPE HTML>
<html lang="ja">
<head>
    <meta charset="UTF-8">
    <title>要素の取得テスト</title>
</head>
<body>
    <div id="test1">書き換えるための文字列 (text)</div>
    <div id="test2">書き換えるための文字列 (html)</div>
    <div id="test3">id test3 の中の文字列</div>
    <script>
    <!--
    // 文字ベースで書き換えることができます。
```

```
    var test1object = document.getElementById("test1");
    test1object.textContent     = "id test1 の文字列を<br>書き換えたよ!";
    // html ベースで書き換えることができます。
    var test2object = document.getElementById("test2");
    test2object.innerHTML = "id test2 の文字列を<br>書き換えたよ!";
    // id test3 の中の文字列を取得して表示
    var test3object = document.getElementById("test3");
    alert(test3object.innerHTML);
    -->
    </script>
</body>
</html>
```

さて、上記の HTML を書き換える方法ですが、まず、オブジェクトを取得します。
大抵一つのタグが、一つのオブジェクトに対応していると考えてください。
オブジェクトを取得する関数です。

```
document.getElementById(引数)
```

これは、html のソースコードの中の id 属性を検索し、
引数の id 属性にあったものをオブジェクトとして返します。
例えば、

```
document.getElementById("test1")
```

この場合は、8 行目の id が test1 の div タグを返します。
取得したタグの属性や中身を変えることができます。例えば下記のプログラムでは、

```
test1object.textContent = "id test1 の文字列を<br>書き換えたよ!";
test2object.innerHTML = "id test2 の文字列を<br>書き換えたよ!";
```

前者はそのままテキストとして書き換え、後者は HTML も認識して変更できます。
innerHTML や innerText 属性をいじることによってページを書き換えることが可能になります。

書き替えたり加えたりすること以外にどんなことができるか
下記サイトにまとめられています。

要素の取得方法色々
http://crocro.com/write/manga_javascript/wiki.cgi?p=Web%A5%DA%A1%BC%A5%B8%A4%CE%BD%F1%A4%AD%B4%B9%A4%A8#p10
要素の操作方法色々
http://www.tohoho-web.com/js/element.htm#Element

Q. なにか動かした時に操作を行いたい!

A. イベントハンドラを使おう!

普段何気なくマウスで操作した時や、キーボードを押した時、などの イベント にプログラムを割り当てることができます。そのためのものが イベントハンドラ です。
イベント は、HTML タグ内の、イベントハンドラ というところに書きます。
この イベントハンドラ は、on～ という名前の属性になっています。
そして、イベントハンドラ に対応した イベント が起こると、その処理が実行されます。
ブラウザの種類やバージョンによって、各タグで使用可能なイベントハンドラは異なります。以下はサンプルです。

```
<!DOCTYPE HTML>
<html lang="ja">
  <head>
     <meta charset="UTF-8">
     <script type="text/javascript">
     <!--
     function clickButton() {
          alert("click ボタンがクリックされました!");
     }
     function onmouse() {
          alert("マウスが乗っています");
     }
     function load() {
          alert("ページが読み込まれました");
     }
     -->
     </script>
  </head>
  <body onload="load();">
     <div>
        <button onClick="clickButton();">click!</button>
        <div id="mouse" onMouseOver="onmouse();">
         マウスを乗っけてね
        </div>
     </div>
  </body>
</html>
```

上記のように、

```
<body onload="関数();">
<button onClick="clickButton();">
<div onMouseOver="関数();">
```

のように HTML タグの中に属性として、イベントハンドルを付与し、関数を割り当てることができます。
例えば onload であれば、画面を読み込んだ時に実行します。
onClick であれば、属性をつけたタグにクリックされた時に実行します。
onMouseOver であれば、属性をつけたタグにマウスを上に載せることで実行します。
実際にサンプルをコピーして試してみるとよいでしょう。

このように、イベントハンドラを使うことによって、なにかイベントが起こる時にプログラムを実行することができます。

下記に参考リンクを記載しておきます。各種のイベントハンドラです。
http://www.tohoho-web.com/js/onevent.htm
イベントハンドラ の中で、よく使うものを以下に示します。

属　性	発生タイミング
onClick	クリックされた時
onMouseOver	マウスカーソルが乗った時
onMouseOut	マウスカーソルが離れた時
onLoad	Web ページが読込まれた直後
onUnload	他のページに移動する直前
onFocus	フォームの部品などが操作対象になった時
onBlur	フォームの部品などが操作対象から外れた時
onChange	フォームの部品の内容が変更された時
onSubmit	フォームの実行がなされた時

コピー元

http://crocro.com/write/manga_javascript/wiki.cgi?p=%A5%A4%A5%D9%A5%F3%A5%C8%A4%CB%A4%E8%A4%EB%BD%E8%CD%FD#p8

Q. 時間ごとに処理を行いたい!

A. setTimeout() や setInterval() を使いましょう!

二つの間で何が違うのかというと

`setTimeout()` ： 設定時間の後に関数を呼び出す命令です。処理が終わってから次の処理を始める。

`setInterval()` ： 設定時間毎に関数を呼び出す命令です。処理が終わっていなくても指定の時間毎に処理を始める

このような違いがあります。

一見、setInterval がループ、setTimeout が一回きりとも取れるかもしれませんが、それはループになるようにプログラムを組むことによって、似たような処理を実行出来ます。時と場合によって使い分けてください。ネットの一例としては、

“setInterval は指定時間間隔で処理を機械的にします。
なので、重い処理だと、完了していないのに、時間切れで次の処理を実行して大変な修羅場になります。”

“もしあなたがユーザの操作を必要としないなら、提供されたコードを繰り返し実行する setInterval を使うのが一番です。”

といった話があるようです。

さてここまで二つの関数の特徴を記載しましたが、
次は二つの関数の使い方について記載します。

- 第一引数は関数
- 第二引数は指定時間ミリ秒単位(1000 ミリ秒 = 1 秒)

となります。

では、実際のプログラムを見てみましょう。

```
var count = 0;
setInterval(
    function () {alert((++count * 10) + '秒');}// 第一引数
    , 10000);// 第二引数
```

このプログラムは 10 秒毎に、何秒たったかを通知するプログラムです。

第一引数の関数にはすでに定義した関数を使うことも可能です。

`setTimeout("関数名()", 指定時間);`

関数名() を `""` で囲ったものを第一引数で渡すことにより可能となります。

[付録] GitHub について

バージョン管理システム

- VCS – Version Control System
  BTS と並ぶ OSS の必須ツール。
- Repository
  ソースの履歴情報保管場所

バージョン管理システムの種類

- 集中管理型
  CVS , Subversion (SVN) など
  レポジトリが 1 か所にあり、作業コピーだけを取得して編集
- 分散管理型
  git , mercurial (hg) , bazaar など
  レポジトリ丸ごとをコピー
  手元で自由に差分や履歴を参照できる

Git とは

分散管理型 VCS の代表
GitHub がよくできているので人気
是非使ってみましょう。

レポジトリサービス

GitHub とは

- Git レポジトリ
- Issues (BTS)
- Wiki
- Web (GitHub Pages)

などの機能がある Web サービス。

GitHub を使ってみよう

1.Git のインストール

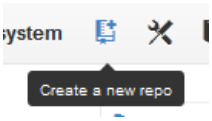
Windows MinGW や Cygwin などでインストール

Mac `brew install git`

Linux `sudo apt-get install git`

2, github.com にアクセスしてサインアップ

3.レポジトリの作成



画面右上の Create a new repo ボタンをクリックして、必要項目を入力

レポジトリができたら説明されてる通りにする前に SSH をの設定を。
というか SSH がオススメ。

4.SSH キーを作成しよう

```
// SSH キーペアの生成
ssh-keygen
// 指示に従ってキーを作成
// パスフレーズは絶対忘れちゃダメ!
// ~/.ssh/id_rsa が秘密鍵: 絶対公開しちゃダメ
// ~/.ssh/id_rsa.pub が対応する公開鍵
// 公開鍵の中身を確認
cat ~/.ssh/id_rsa.pub
// この 1 行テキストを GitHub 等に登録する
```

あとは、アカウント設定のページで SSH Keys に id_rsa.pub の内容をペーストしよう。

5. レポジトリの操作

```
// ファイルを作成
touch README.md
// リポジトリを初期化
```

```
git init
// ファイルの変更や追加を登録
git add README.md
// 新しいリビジョン(履歴)として記録
git commit -m "first commit"
// 手元のリポジトリを GitHub と関連づける
git remote add origin git@github.com:<user>/<repo>.git
// SSH を使わない場合はこちら:
// https://github.com/<user>/sandbox.git
// 手元のリポジトリのリビジョンを GitHub に送信
git push -u origin master
```

github に更新されたのを確認してみましょう。

```
// 最新リビジョンを GitHub から取得
git pull
// ファイルを編集 (適当なエディタで)
vi README.md
// ファイルの変更や追加を登録
git add README.md
// 新しいリビジョン(履歴)として記録
git commit -m "README 書き換えた!"
// 新しいリビジョンを GitHub に送信
// origin master は初期化時以外は省略可
git push
```

このような流れです。`git diff` や `git status` など便利なコマンドもあります。

他にもいろいろあります。

こわくない git

http://www.slideshare.net/kotas/git-15276118