

第1回

JavaScriptから始める プログラミング

京都大学工学部情報学科

計算機科学コース3回

KMC2回 drafear

自己紹介

- id
 - drafear(どらふいあ, どらふぁー)
- 所属
 - 京都大学 工学部 情報学科 計算機科学コース 3回
- 趣味
 - ゲーム(特にパズルゲー), ボドゲ, ボカロ, twitter
- 参加プロジェクト
 - **これ**, **競プロ**, ctf, 終焉のC++, coq, 組み合わせ最適化読書会
- 言語
 - Japanese, C++, JavaScript, English, C#, Python, Perl, Ruby



りーさん
(がっこうぐらし)

※青: 新入生プロジェクト

頻度(高)

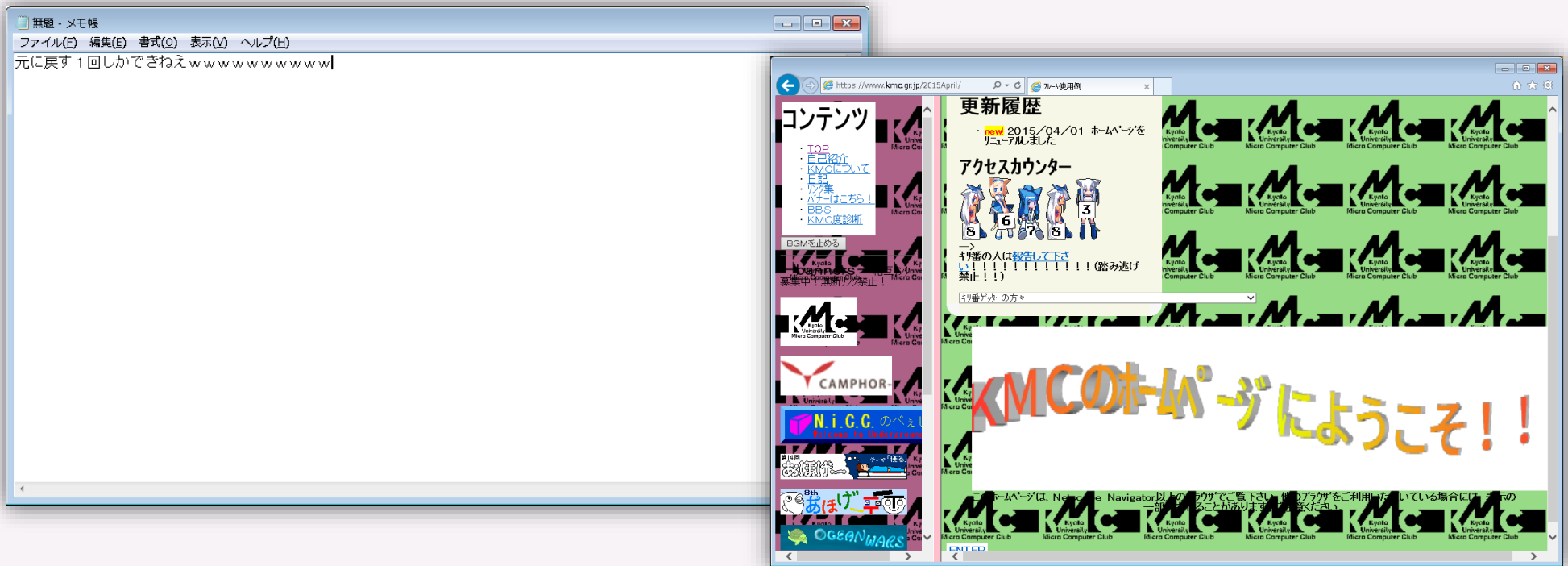
頻度(低)

自己紹介



- KMC 2回生
- 新入生
- KMC 3回生
- KMC 4回生
- KMC n回生
- 学部学科
- id
- 趣味
- 好きなエディタ(あれば)

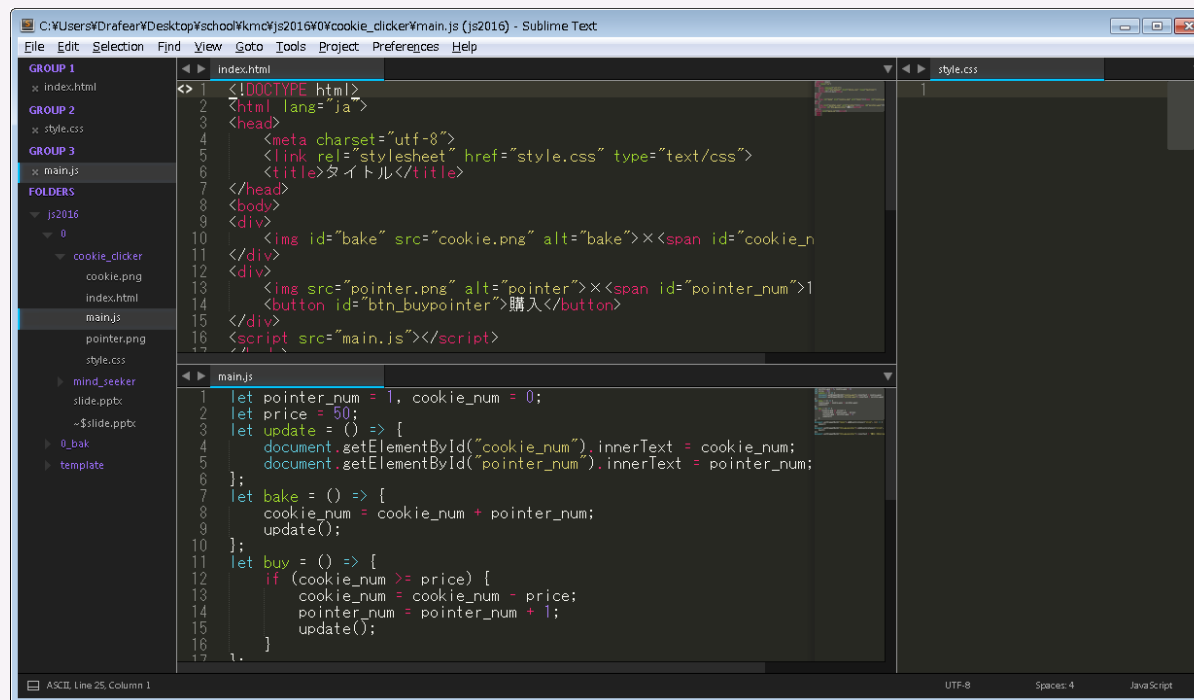
始める前に...

- ブラウザとエディタの環境を整えよう
 - Internet Explorerとメモ帳でも開発できるけどTSURAI
 - 俺はこれでやるんだ！！って人は入れなくておk



この講座で使用するブラウザとエディタ

- Google Chrome 
 - <https://chrome.google.com>
- Sublime Text 3 
 - <https://www.sublimetext.com/3>



今日の目標

- HTML, (CSS,) JavaScriptの役割を理解する
- 色々学ぶ
 - HTML
 - 画像が貼れる
 - ボタンを設置できる
 - JavaScript
 - 変数, 関数, オブジェクトの概念を理解する
 - 数値と文字列の違いを理解する
 - 単純なイベント処理, タグの中身のテキストの変更ができる
- C○○kie Clicker のようなゲームを作る

やっと本題

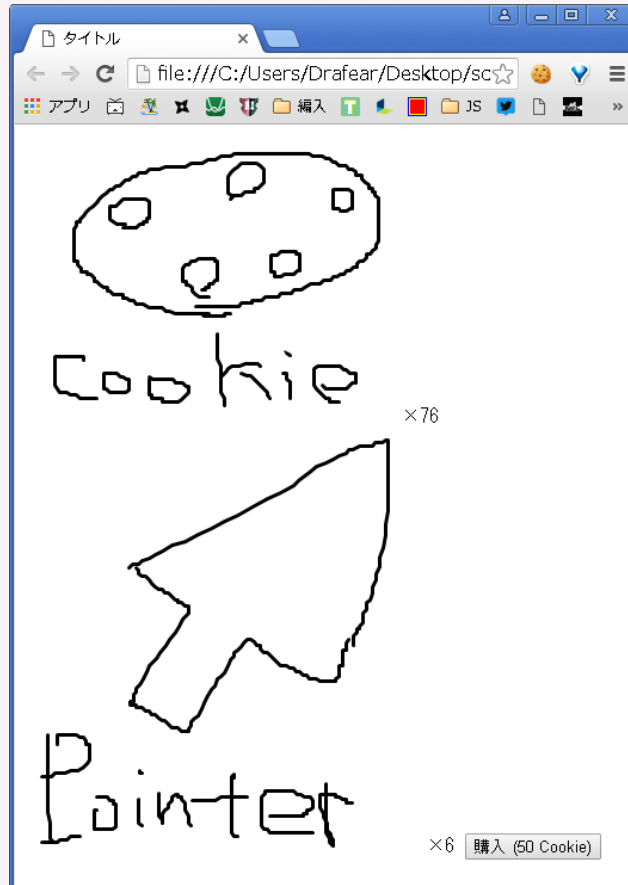
- HTML, CSS, JavaScript の役割
 - HTMLで主に構造と各要素の意味を記述し (骨組み)
 - CSSでレイアウトやデザインを行い (肉付け)
 - JavaScriptでWebページを動的なものにする (動かす)

おねがい

- 分からなかったらすぐに聞いて下さい！
- あなたの分からないは みんなの分からないです

今日作るもの

- Cookie Clicker のようなもの
 - https://inside.kmc.gr.jp/~drafear/js2016/works/drafear/cookie_clicker/

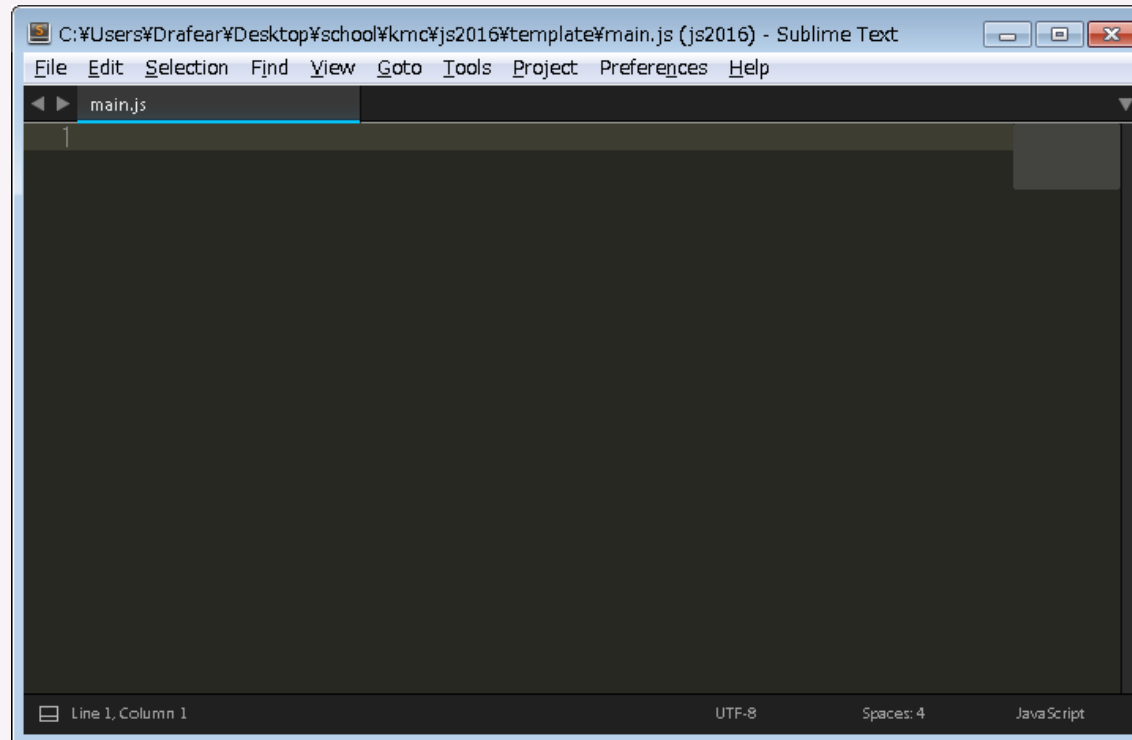


その前に

- 作り始める前に、まずは JavaScriptの基礎 を勉強していきましょう

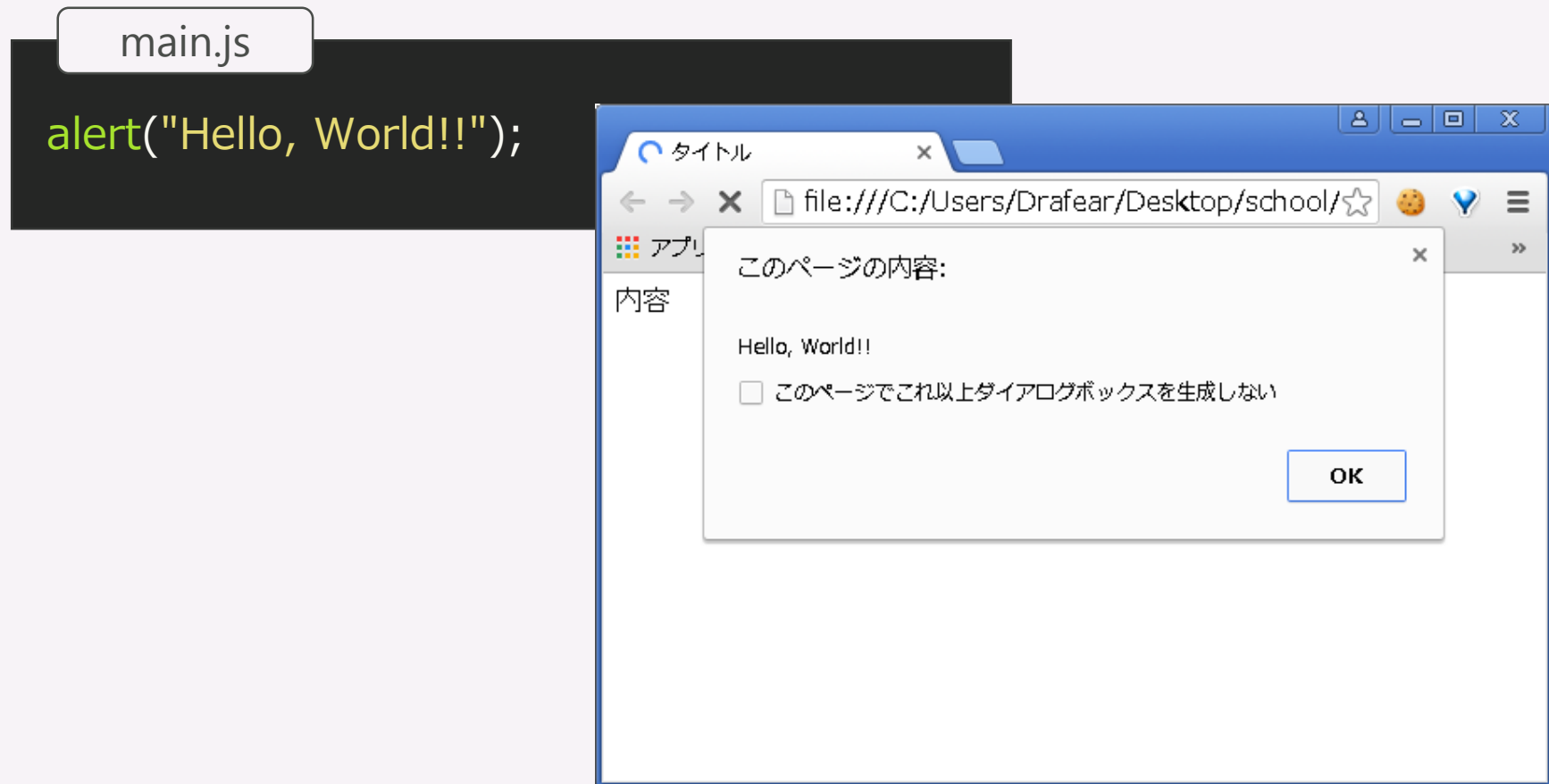
てんぷれ

- 以下から(またはwikiから)雛形をダウンロードしてください
 - <https://inside.kmc.gr.jp/~drafear/js2016/template.zip>
- ダウンロードできたら main.js を開いて下さい



Hello, World!!

- main.js に以下のように記述して index.html をChromeで開くと

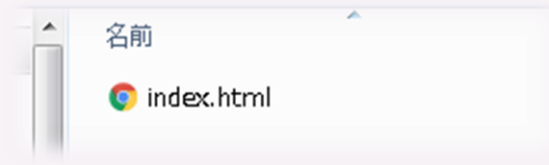
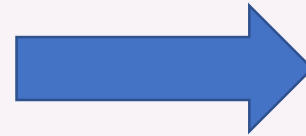
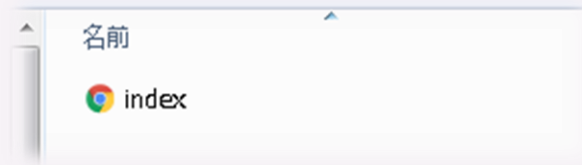


拡張子

- “.html” とは？
 - ファイル名の最後の「.」以降の文字列を拡張子といいます
 - a.png なら png
 - hoge.fuga.piyo なら piyo
 - 拡張子はファイルの種類を識別するもの
 - png: png形式の画像ファイル
 - gif: gif形式の画像ファイル
 - html: HTMLファイル
 - txt: テキストファイル

拡張子が表示されない場合

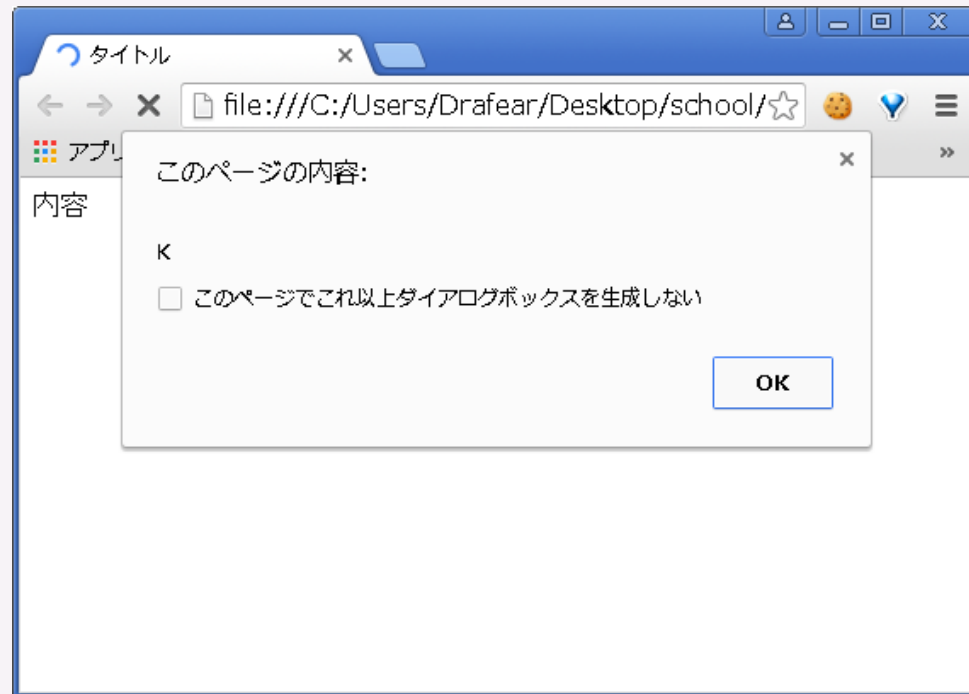
- “win8 拡張子 表示”などでggして下さい
 - コントロールパネルの「ファイルとフォルダーの検索オプションの変更」
「表示」タブから、「登録されている拡張子は表示しない」のチェックを外す



Hello, World!!

- alert("内容") で内容を表示します
- alert を3つ並べてみます
 - K, M, Cの順に表示される
 - プログラムは上から順に実行される

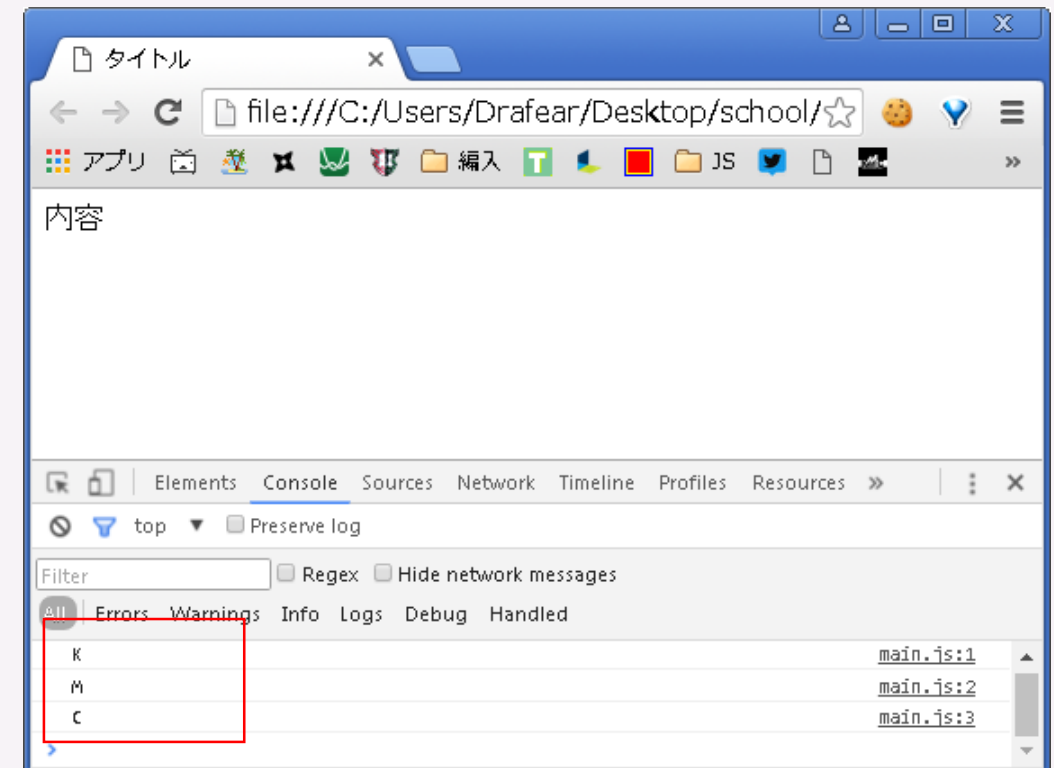
```
main.js  
  
alert("K");  
alert("M");  
alert("C");
```



console.log

- 3回の出力を見るのに一々 [OK] を押していくのは面倒
 - `console.log` を使おう
 - Chrome上で **F12** または **Ctrl+Shift+I** を押すと出てくる開発者ツール内のconsoleに出力される
 - ユーザーには見えない

```
main.js
console.log("K");
console.log("M");
console.log("C");
```



セミコロン と エラー

- セミコロン

- 1つの命令ごとに, おしりにセミコロンを置きます
 - 置きたくない人は「js 自動セミコロン挿入」などでggって下さい

- エラー

- JavaScriptでエラーが発生した場合, コンソールに表示されます
 - コンソール便利!

main.js

```
console.log("K");  
console.log("M");  
console.log("C");
```

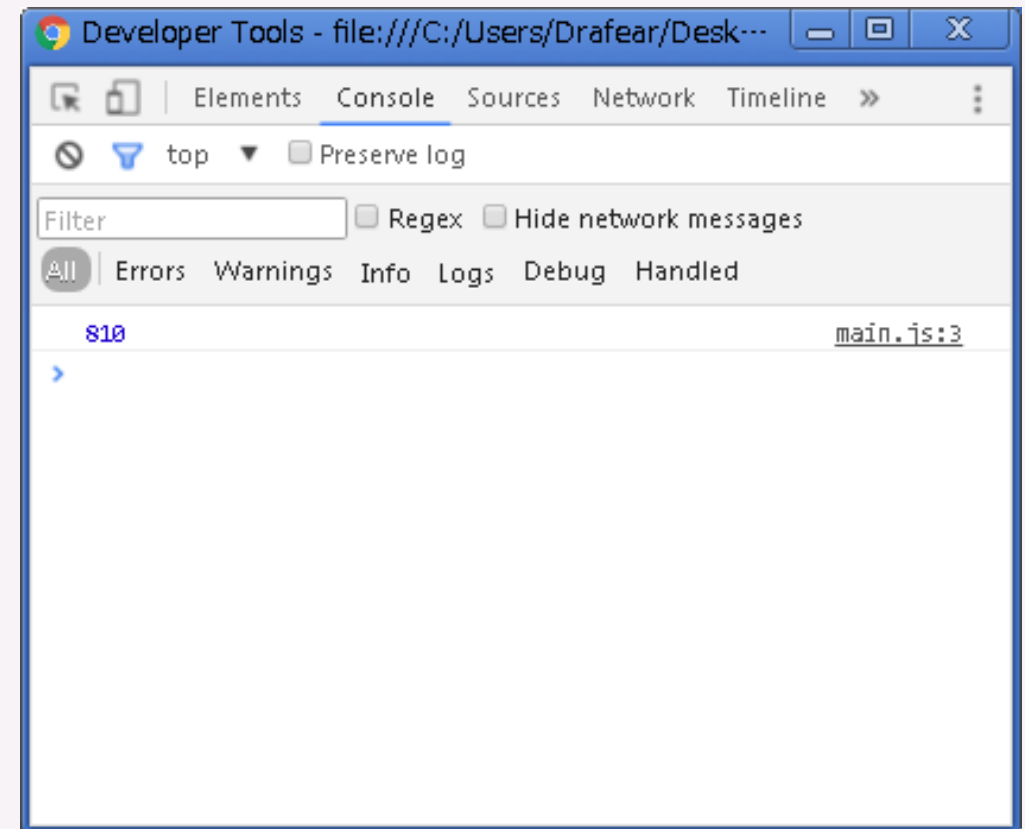
変数

- 変数
 - 変数とは, 値を格納できる箱のことです
 - `let 変数名 = 初期値;`
 - 上のように記述すると, その変数(箱)を使用できるようになります
 - 変数名は数字から始まってはいけません
 - `変数名 = 値;`
 - 変数の値を右辺の値で上書きします
 - この操作を代入といいます
 - 数学のイコールとは違います

変数

- console.log(変数名)
 - その変数の中身をコンソールに出力します

```
main.js
let a = 50;
a = 810;
console.log(a);
```



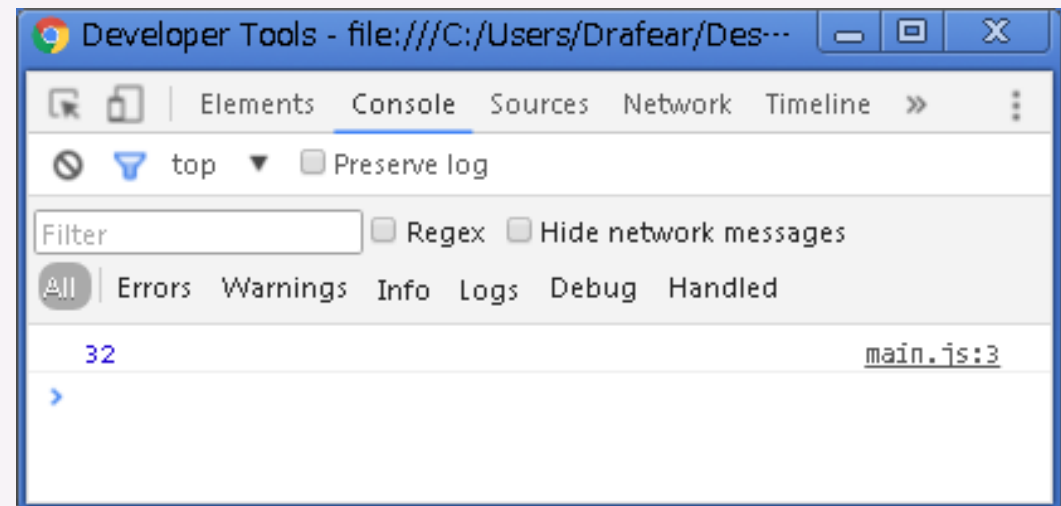
四則演算

- 四則演算

- 「+」 「-」 「*」 「/」 でそれぞれ 和, 差, 積, 商 を求められます
 - 例) `a = a + 4;` (aに4を足す)
- 「%」 で剰余を求めます
 - 例) `a = b % 5;` (bを5で割った余りをaに代入する)
- 「+」 「-」 「*」 「/」 「%」 「=」 などを**演算子**といいます

main.js

```
let a = 4;  
let b = 7;  
let c = a + a * b;  
console.log(c);
```

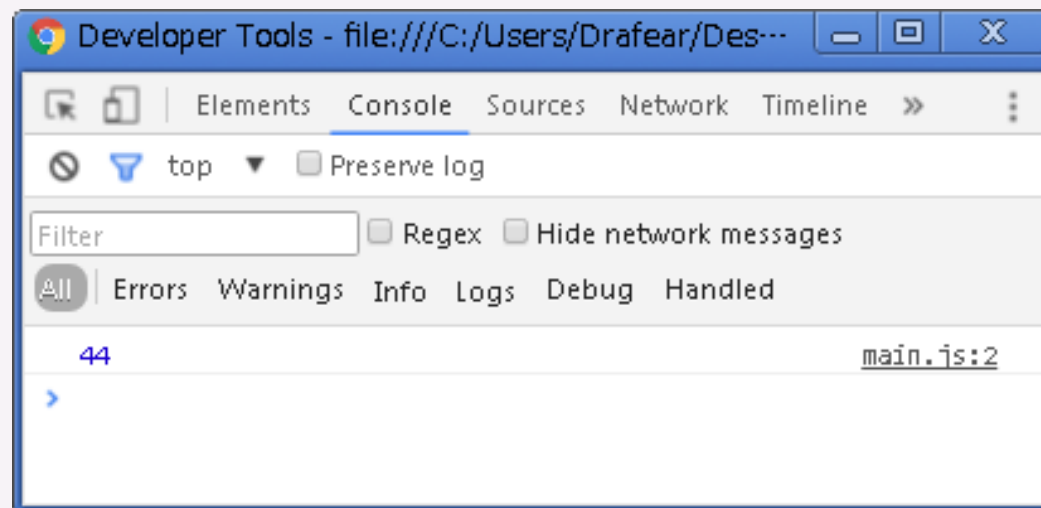


四則演算

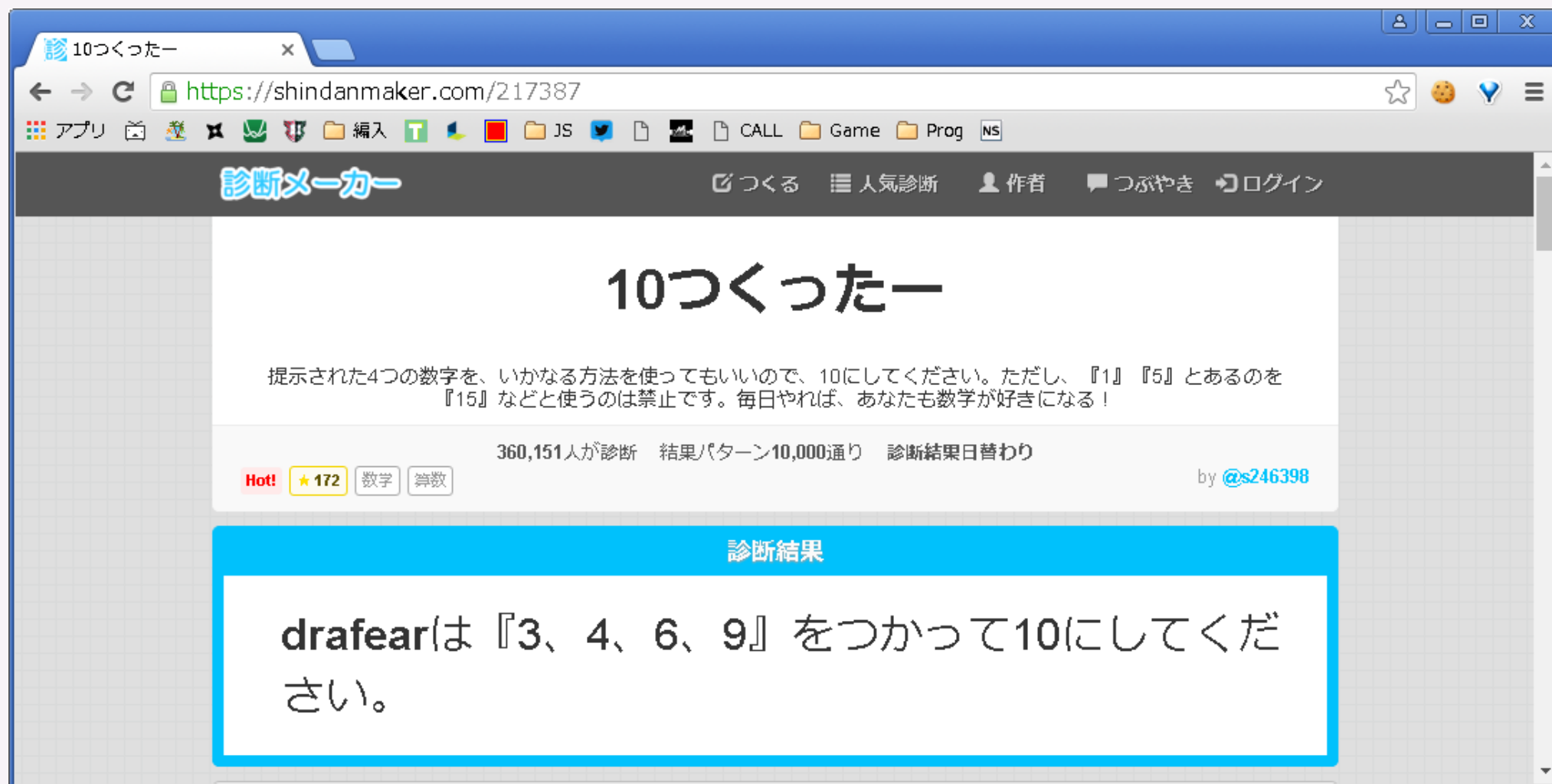
- 四則演算
 - ()内の演算が優先されます

main.js

```
let a = 4 * (8 + 3);  
console.log(a);
```



四則演算



四則演算

- 演習

- a, b, c, d を1回ずつと
適宜 (,), +, -, *, /, % を使って10を作ってください

main.js

```
let a = 3;  
let b = 4;  
let c = 6;  
let d = 9;  
let result = fill in here ;  
console.log(result);
```

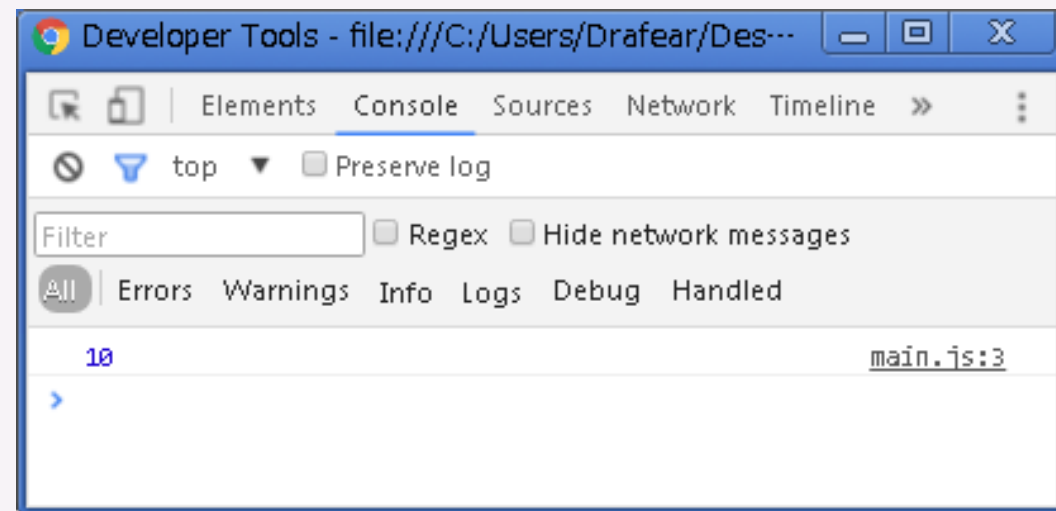
四則演算

• 演習

- a, b, c, d を1回ずつと適宜 (,), +, -, *, /, % を使って10を作ってください
- これをプログラムで解くのに興味がある方は是非、金曜日の「競技プログラミング練習会」にご参加下さい！！

main.js

```
let a = 3;  
let b = 4;  
let c = 6;  
let d = 9;  
let result = (b + c) % (a + d);  
console.log(result);
```

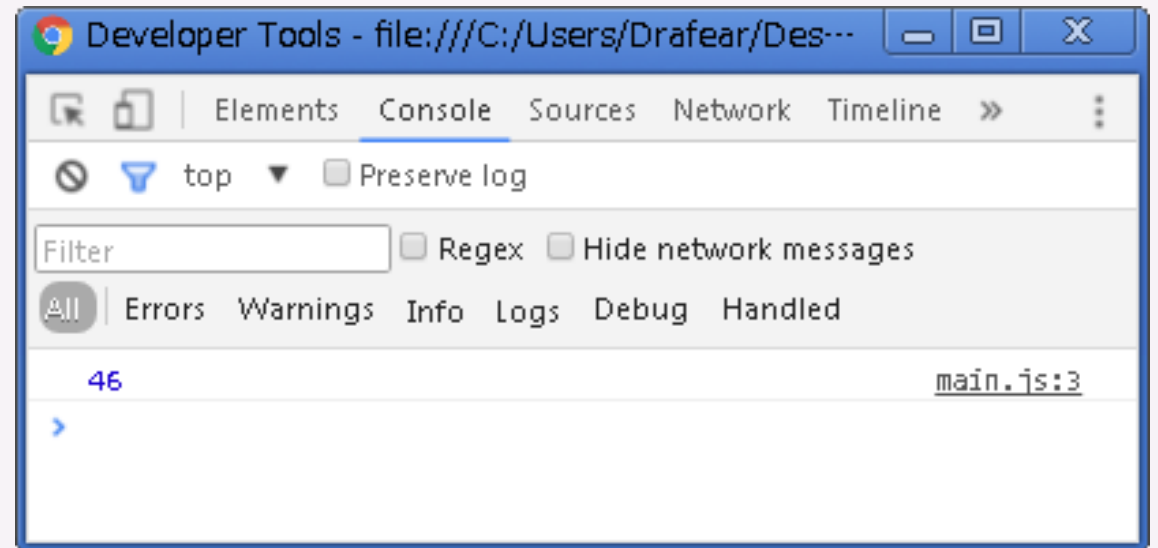


四則演算

- 四則演算

- 一部の演算子は「a = a [演算子] b」を省略して「a [演算子] = b」と書けます
 - 例) a = a * 4; ⇔ a *= 4; (aを4倍する)

```
main.js
let a = 31;
a += 15;
console.log(a);
```

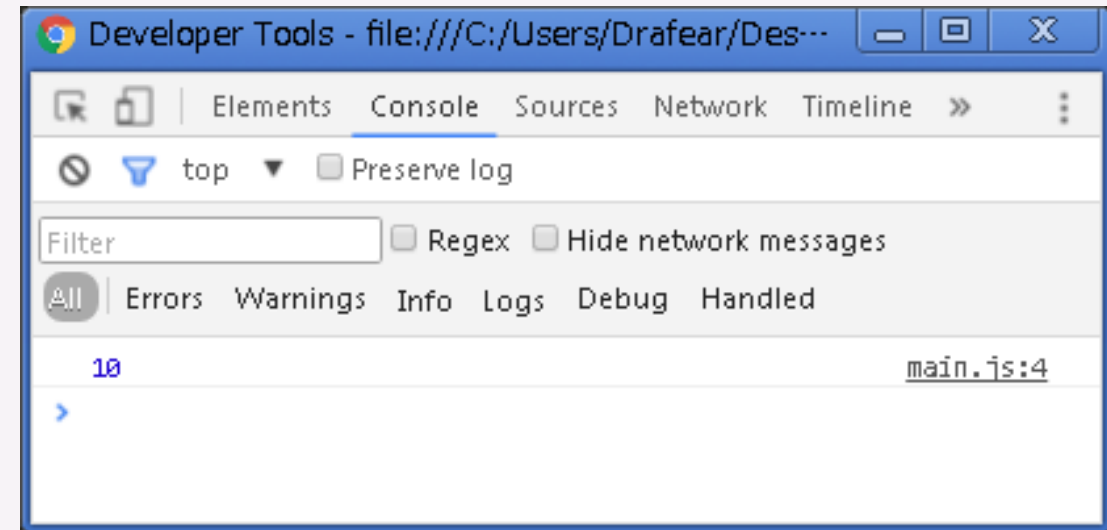


コメント

- コメント

- コメントは注釈であり, プログラムとして実行されない
- 「//」以降のその行の文字はコメントとなる
- 「/*」と「*/」で囲まれた部分はコメントとなる

```
main.js
/* コメント */
let a = 10; // コメント
// a *= 3;
console.log(a);
```



関数

- 関数

- 一連の手続きをまとめたものを関数という
- (仮引数1, 仮引数2, ..., 仮引数n) => { 処理 }
- 関数に渡されるものを引数という

main.js

```
// 第一引数と第二引数の和をコンソールに出力する
let print_sum = (a, b) => {
  console.log(a + b);
}
// 15 + 21 を出力する (36が出力される)
print_sum(15, 21);
```

(一応)仮引数といいます

(一応)実引数といいます

関数

- 関数

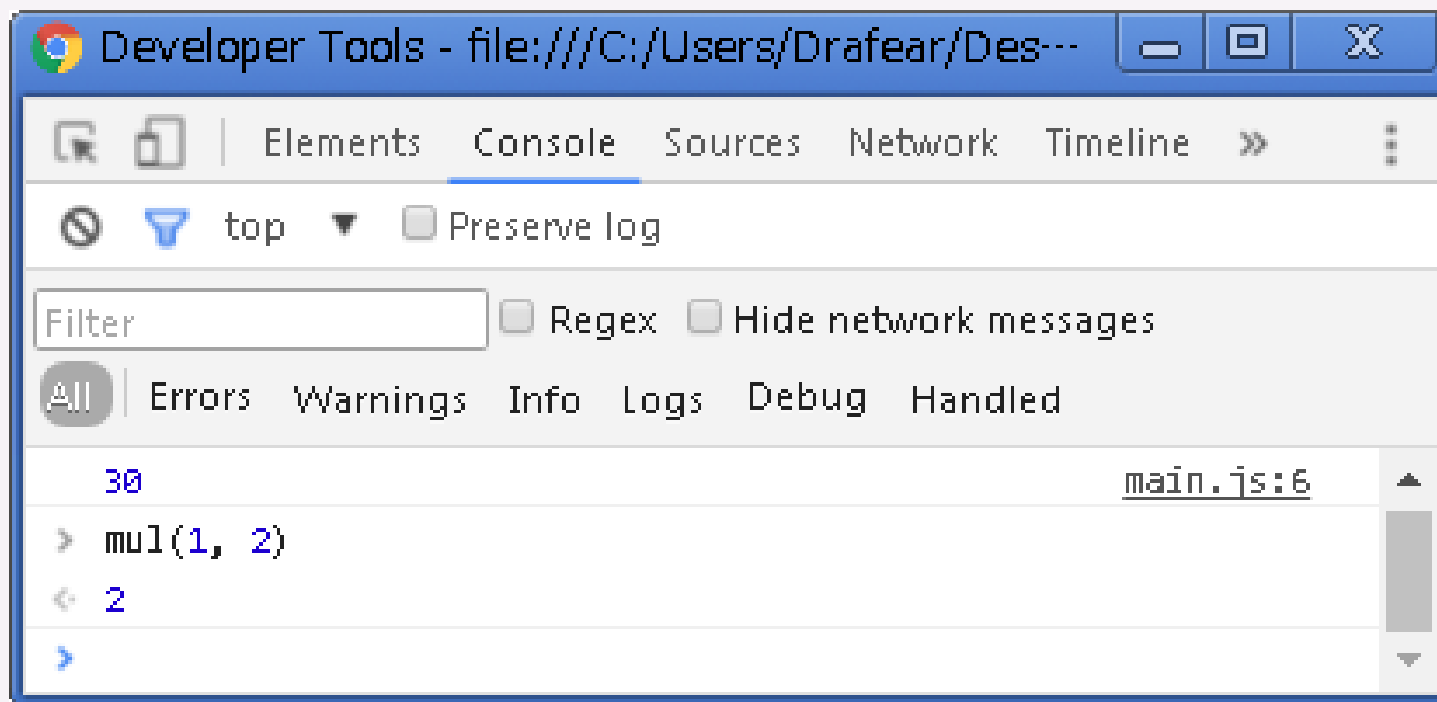
- (仮引数1, 仮引数2, ..., 仮引数n) => { 処理 }
- 関数は return 文により値を返すことができ,
返される値を 戻り値, 返り値, 返却値 などと呼ぶ

main.js

```
// 第一引数と第二引数の積を返却する
let mul = (a, b) => {
  return a * b;
}
// 6 * 5 を出力 (30が出力される)
console.log( mul(6, 5) );
```

コンソールからの実行

- コンソールからも JavaScript を実行できます



演習

1. a, b を受け取って, $a - b$ を返す関数 `sub` を書いてみましょう
 - 引数は a, b の2つ
2. x を受け取って x^2 を計算する関数 `square` を書いてみましょう
 - 引数は x の1つ

演習

1. a, b を受け取って, $a - b$ を返す関数 `sub` を書いてみましょう
- 引数は a, b の2つ

main.js

```
let sub = (a, b) => {  
  return a - b;  
}
```

2. x を受け取って x^2 を計算する関数 `square` を書いてみましょう
- 引数は x の1つ

main.js

```
let square = (x) => {  
  return x * x;  
}
```

文字列

- 文字列
 - 文字列は `""` または `"` または ``` で囲む
 - 例) `let str = "kmc";`
 - 「+」演算子で文字列を連結できる

main.js

```
let str1 = "to";  
let str2 = "kyoto";  
let str3 = str1 + str2;  
console.log(str3); // tokyoto と表示
```


オブジェクト

- オブジェクト (超重要)
 - { key₁: value₁, key₂: value₂, ..., key_n: value_n }
 - obj.key または obj["key"] でアクセス

main.js

```
let human = {  
  name: "drafear",  
  age: 21,  
  skill_level: {  
    js: 0,  
    "c++": 1,  
  },  
  talk: () => {  
    console.log("Hello!");  
  }  
};
```

```
console.log(human.name); // drafear  
console.log(human["age"]); // 21  
console.log(human.skill_level.js); // 0  
console.log(human.skill_level["c++"]); // 1  
human.talk(); // Hello!  
console.log(human); // Object { name: ... }
```

オブジェクト

- console.log の構造
 - 大体こんなかんじ

main.js

```
let console = {  
  log: (...) => { ... }  
};
```

演習

- 内積を求める関数 **dot** を作って下さい

main.js

```
let a = { x: 4, y: 3};  
let b = { x: -10, y: 5 };  
let result = dot(a, b);  
console.log(result); // -25
```

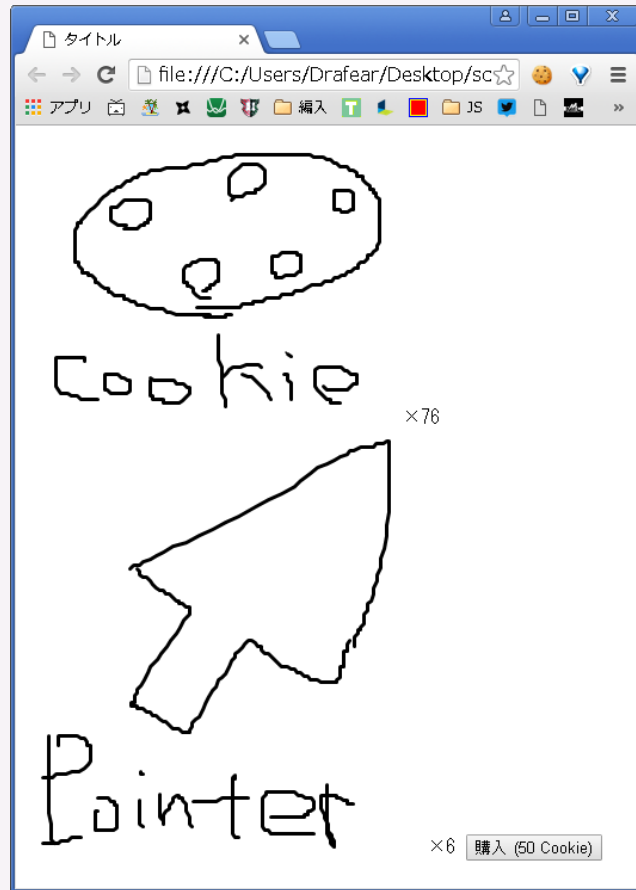
演習

- 内積を求める関数 **dot** を作って下さい

main.js

```
let dot = (p1, p2) => {  
  return p1.x * p2.x + p1.y * p2.y;  
}  
let a = { x: 4, y: 3};  
let b = { x: -10, y: 5 };  
let result = dot(a, b);  
console.log(result); // -25
```

C○○kie Clicker作っていきましょう

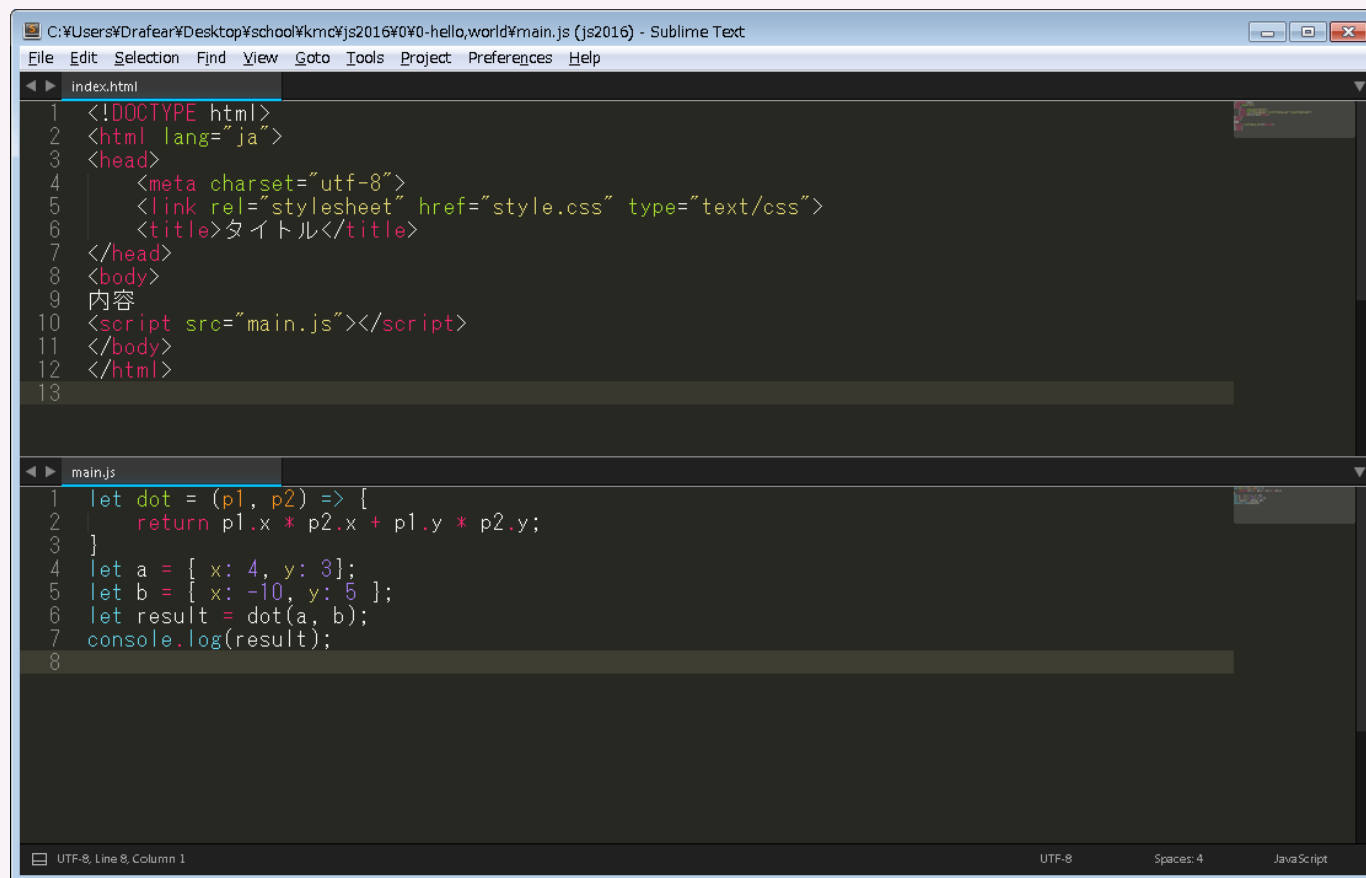


Start HTML

- というわけでHTMLを触っていきます

分割！

- Ctrl + Alt + 8 で 二分割して
上に index.html, 下に main.js を読み込みましょう



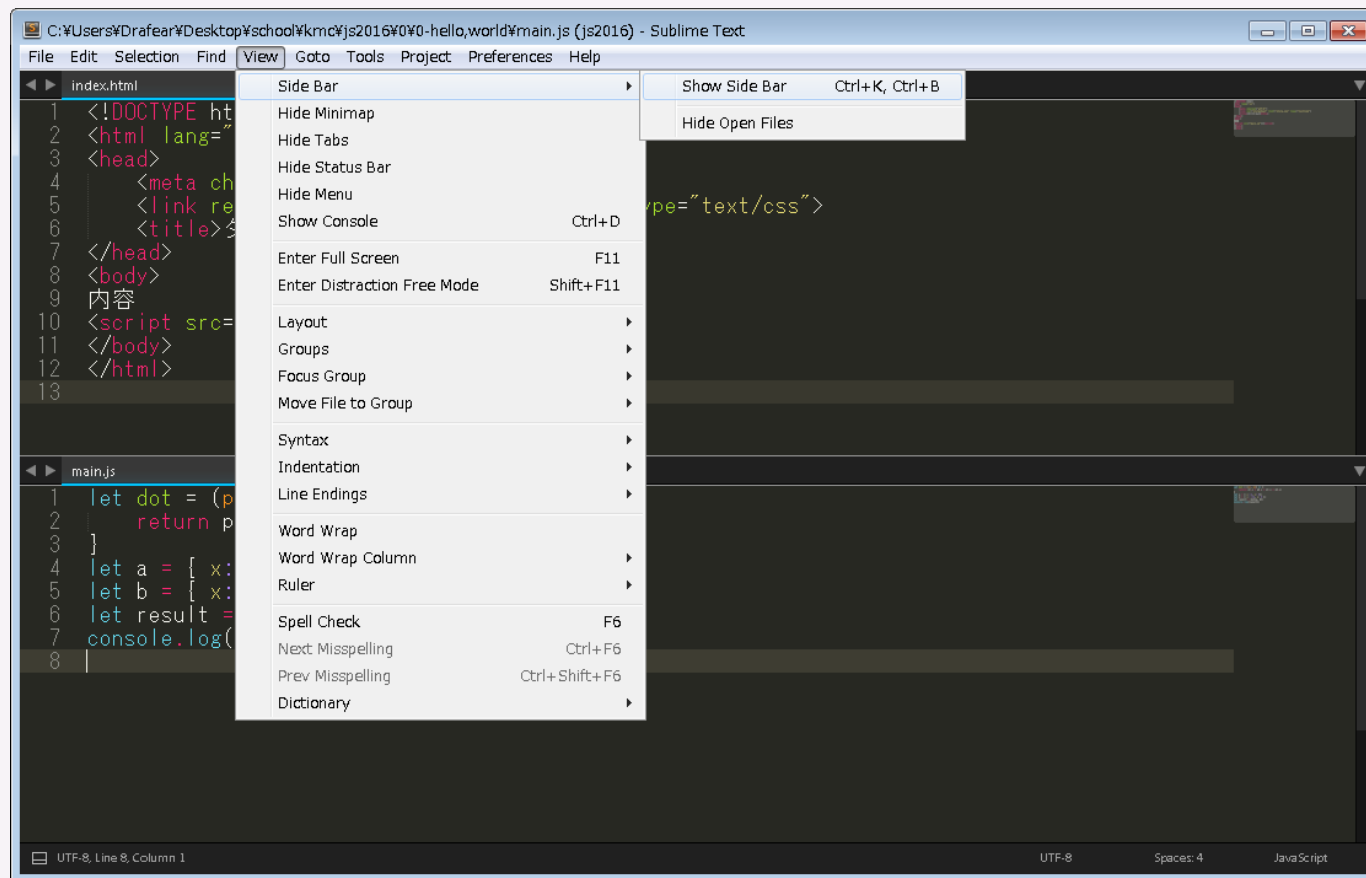
The screenshot shows the Sublime Text editor interface with a split view. The top pane displays the content of index.html, and the bottom pane displays the content of main.js. The index.html file contains a basic HTML structure with a head section including a meta charset, a link to style.css, and a title, and a body section with a script tag for main.js. The main.js file contains a JavaScript function dot and two objects a and b, with a console.log statement.

```
index.html
1 <!DOCTYPE html>
2 <html lang="ja">
3 <head>
4   <meta charset="utf-8">
5   <link rel="stylesheet" href="style.css" type="text/css">
6   <title>タイトル</title>
7 </head>
8 <body>
9 内容
10 <script src="main.js"></script>
11 </body>
12 </html>
13

main.js
1 let dot = (p1, p2) => {
2   return p1.x * p2.x + p1.y * p2.y;
3 }
4 let a = { x: 4, y: 3};
5 let b = { x: -10, y: 5 };
6 let result = dot(a, b);
7 console.log(result);
8
```

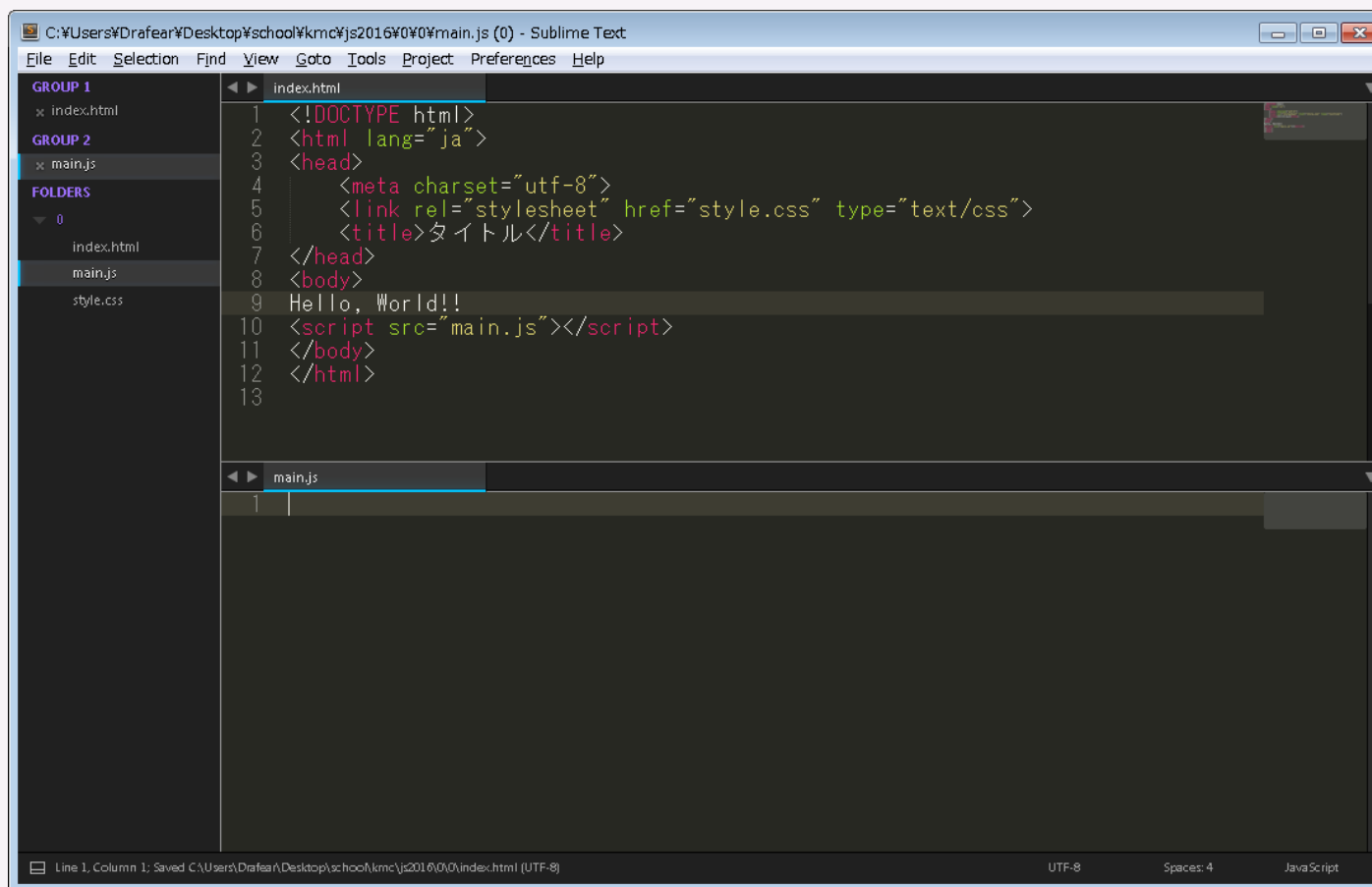
サイドバー

- View -> Side Bar -> Show Side Bar
でサイドバーを出しましょう



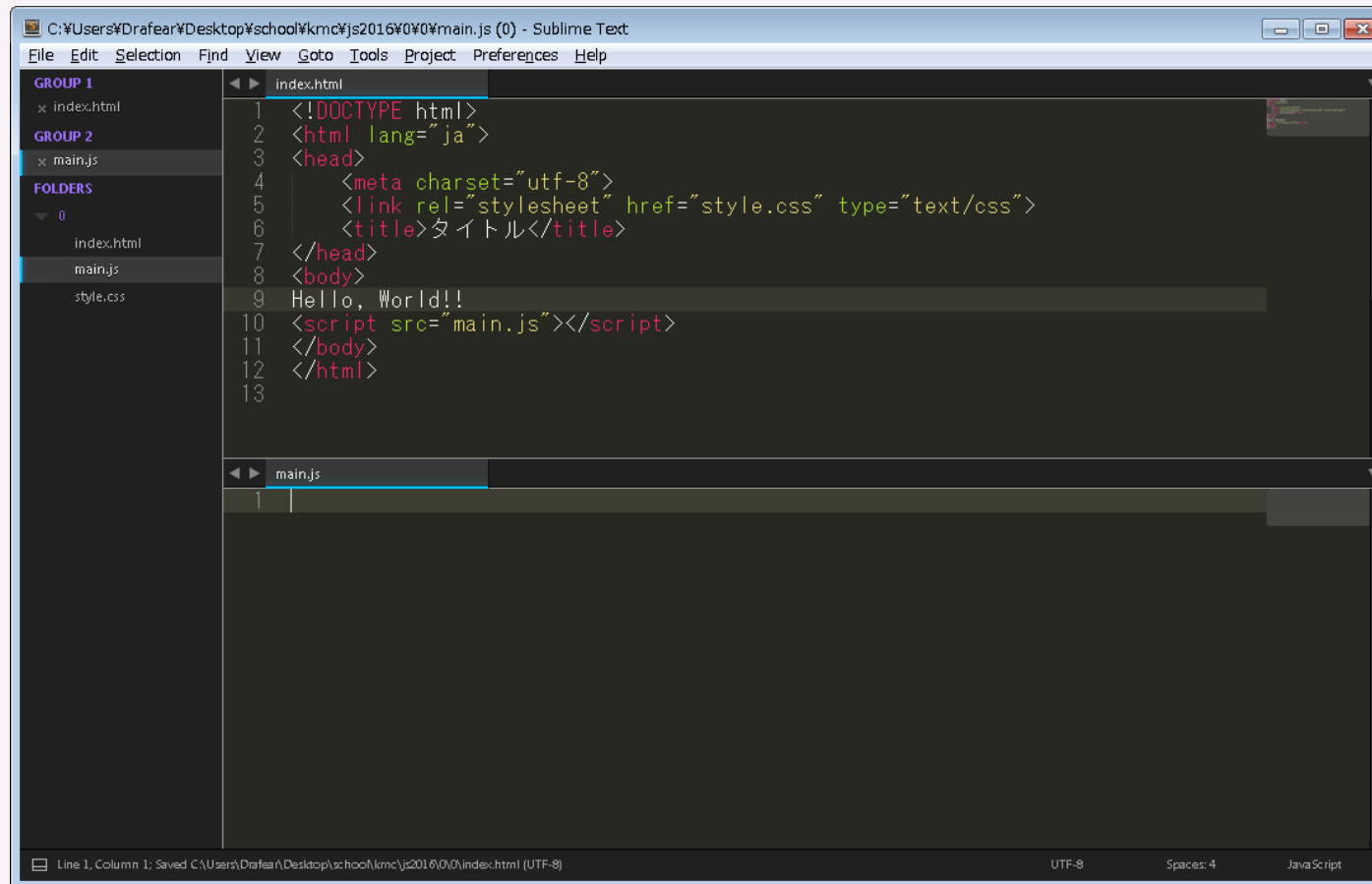
サイドバー

- サイドバーにフォルダをドラッグ&ドロップするとすぐにアクセスできるようになる



Hello, World!!

- main.js を全消去して, index.htmlの「内容」を「Hello, World!!」に書き換えてみましょう



The screenshot shows a Sublime Text editor window with the following content:

```
C:\Users\Drafean\Desktop\school\kmc\js2016\0\main.js (0) - Sublime Text
File Edit Selection Find View Goto Tools Project Preferences Help

GROUP 1
x index.html
GROUP 2
x main.js
FOLDERS
0
index.html
main.js
style.css

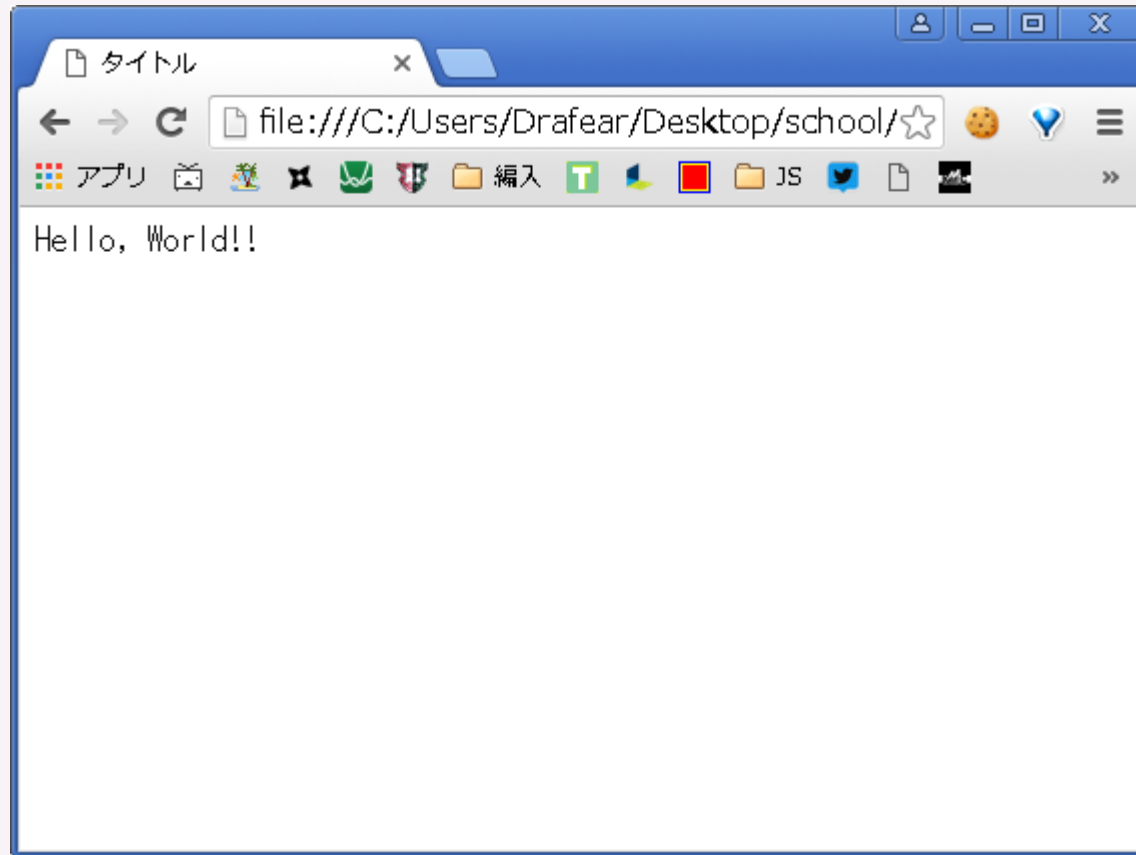
index.html
1 <!DOCTYPE html>
2 <html lang="ja">
3 <head>
4     <meta charset="utf-8">
5     <link rel="stylesheet" href="style.css" type="text/css">
6     <title>タイトル</title>
7 </head>
8 <body>
9     Hello, World!!
10 <script src="main.js"></script>
11 </body>
12 </html>
13

main.js
1
```

At the bottom of the window, the status bar displays: Line 1, Column 1; Saved C:\Users\Drafean\Desktop\school\kmc\js2016\0\index.html (UTF-8) UTF-8 Spaces: 4 JavaScript

Hello, World

- Chromeで開くと, "Hello, World!!" と表示されました！！
- めでたい！！！！！！ 1 1 1 1 1



Hello, Worldを装飾しよう

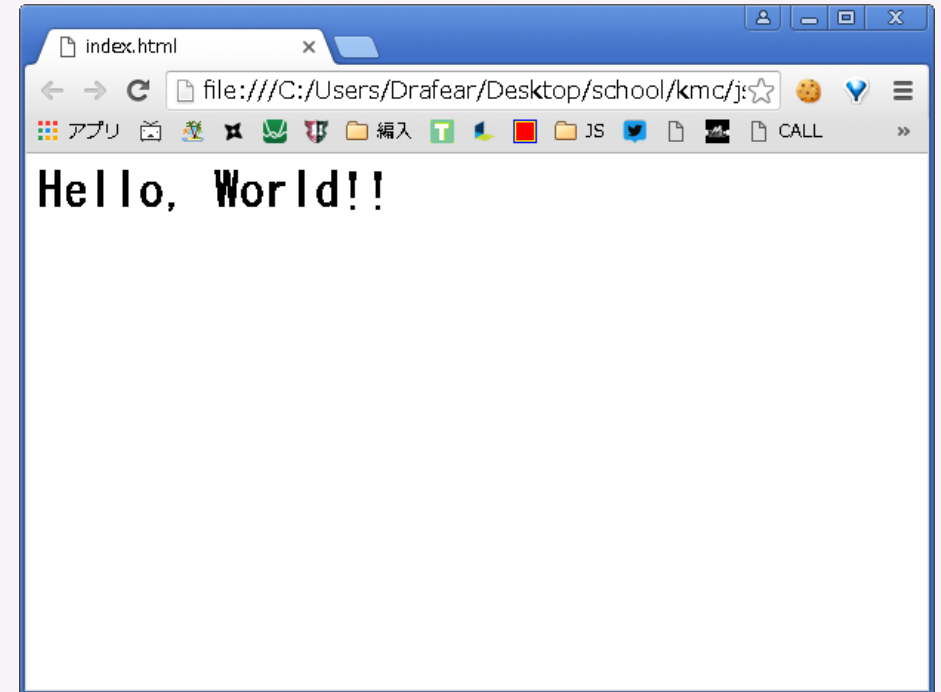
- Hello, World!! を<h1>~</h1>で囲むと...!?

index.html

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="utf-8">
  <link rel="stylesheet" href="style.css" type="text/css">
  <title>タイトル</title>
</head>
<body>
  <h1>Hello, World!!</h1>
  <script src="main.js"></script>
</body>
</html>
```

Hello, Worldを装飾しよう

- でかくなった！！
- `<***>` をタグといいます
 - `<***>`: 開始タグ
 - `</***>`: 終了タグ(閉じタグ)
- `<h1>` をh1タグと呼びます
 - `<h1>`: 開始タグ
 - `</h1>`: 終了タグ(閉じタグ)
- タグには意味があり, h1タグは見出しを意味します
 - h2, h3, ..., h6 もあります



画像を貼ってみよう

- C○○kieとなるものを描きましょう
 - 描いたら image というディレクトリを作ってその中に cookie.png で保存しましょう



```
□ root
└─□ image
   └─ cookie.png
└─ index.html
└─ main.js
   └─ style.css
```

画像を貼ってみよう

- ``
 - srcで指定した画像を表示します
 - 画像が見つからなかった場合代替テキストを表示します
 - 代替テキストの指定は必須です
 - 閉じタグは不要です

index.html

```
...  
<body>  
  
<script src="main.js"></script>  
</body>  
...
```

やったぜ。



リンクを張ってみよう

- 属性

- ``のhrefは属性といいます
- `<タグ名 属性1="属性値1" 属性2="属性値2">内容`
- 指定できる属性はタグごとに決まっています

- パス

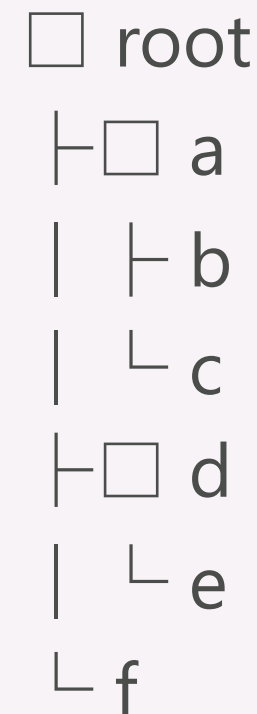
- hrefに指定する先は、絶対パスまたは相対パスで書きます
- 絶対パスとは、`https://www.google.co.jp/` のような、絶対的なものを指します
- 相対パスとは、今のディレクトリから相対的に指定する方法です

相対パス

- | | |
|--------------------|---------------------|
| 1. ファイルbからファイルcを指定 | ./c または c |
| 2. ファイルfからファイルbを指定 | a/b |
| 3. ファイルbからファイルfを指定 | ../f |
| 4. ファイルbからファイルeを指定 | ../d/e |
| 5. ファイルbからファイルcを指定 | ../../a/../../a/./c |

Point

- 「.」は自分自身のディレクトリ
- 「..」は親ディレクトリ
- パスは「ディレクトリ1/.../ディレクトリn/ファイル名」で表す



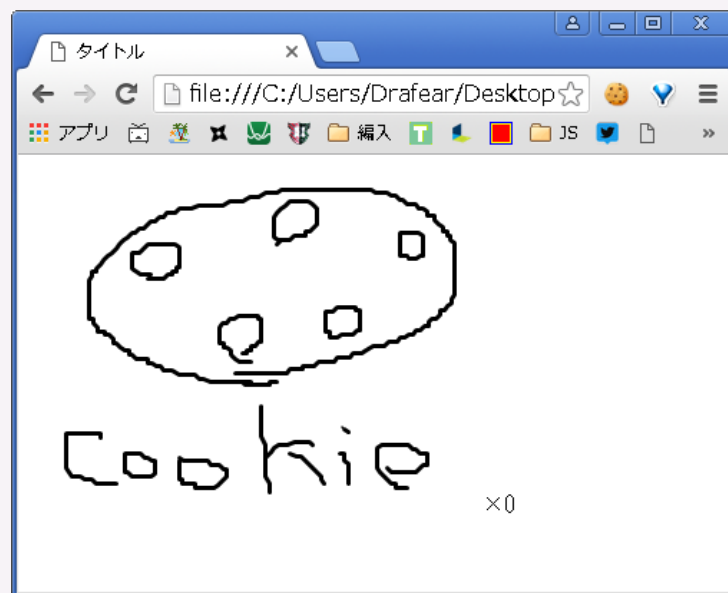
※□はディレクトリです

数を表示しよう

- imgタグの後に ×0 を入れましょう

index.html

```
...  
×0  
...
```



ポインターも作ろう

- 同様にポインターも描きましょう
 - ~~描き直せよ~~



```
□ root
  └─ □ image
      │ └─ pointer.png
      │ └─ cookie.png
      └─ index.html
      └─ main.js
      └─ style.css
```

ぽいんたーも貼るぜ

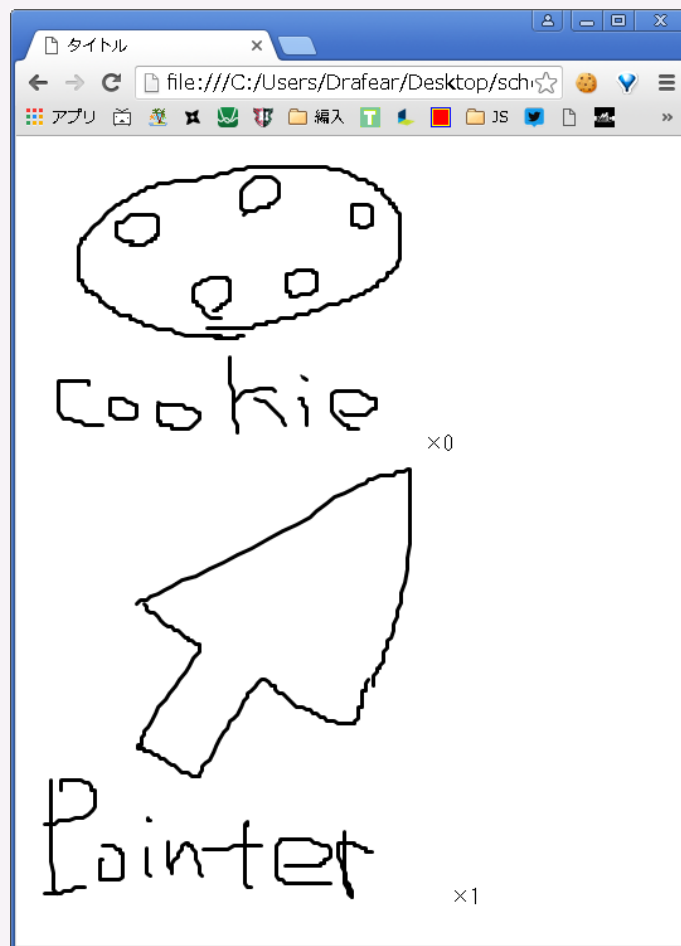
- bodyの中を次のように変更します
 - `</body>` 前の `<script ...>` `</script>` は残しておいて下さい

index.html

```
<div>  
  ×0  
</div>  
<div>  
  ×1  
</div>
```

ぽいんたーも貼るぜ

- それっぽくなってきた



宣伝

- 上手く絵を描けるようになりたい人は是非、土曜日の「お絵かき練習プロジェクト2016」にご参加下さい！！
 - ~~お前が参加しろよ~~

要素

- それぞれのタグで生成されたものを要素といいます
 - 開始タグと終了タグで囲まれた部分が1つの要素になります
 - imgタグはそれだけで1つの要素です
- div要素は汎用コンテナで, とりあえずは直後に改行される空のタグだと考えてもらえたらおkです

index.html (再掲)

```
<div>  
  ×0  
</div>  
<div>  
  ×1  
</div>
```


クリックイベント

- クッキーをクリックしたらalertが出るようにします
- まず, クッキーの要素をJavaScriptからアクセスしやすいようにid属性を振ってやります

index.html

```
<div>
   ×0
</div>
<div>
   ×1
</div>
```

Event

- main.js をいじっていきます
 - document.getElementById("要素id")
 - idが一致する要素を返します
 - elem.addEventListener("event名", (e) => { 処理 });
 - 要素elemが [event名] された時 {処理} を実行します
 - event名は click や keydown や mouseover など色々あります
 - e には左クリックされたか右クリックされたかなどの情報が入ります

main.js

```
let elem_cookie = document.getElementById("cookie");
elem_cookie.addEventListener("click", (e) => {
  alert("clicked!!");
});
```

Event

- ふつう, elemに一旦入れずに, 1行で書きます

main.js

```
document.getElementById("cookie").addEventListener("click", (e) => {  
    alert("clicked!!");  
});
```

Event

1. “click” を “mouseover” に変えてみよう
2. “click” を “mousedown” に変えてみよう
3. “click” を “mouseout” に変えてみよう

クッキーを焼く！

- 内部的にクッキーの数とポインターの数を持つようにしてクッキーがクリックされた時にポインターの数だけクッキーが増えるようにします

main.js

```
let cookie_amount = 0;
let pointer_amount = 1;
let bake = () => {
  fill in here
}
document.getElementById("cookie").addEventListener("click", (e) => {
  bake();
});
```

クッキーを焼く！

- 内部的にクッキーの数とポインターの数を持つようにしてクッキーがクリックされた時にポインターの数だけクッキーが増えるようにします

main.js

```
let cookie_amount = 0;
let pointer_amount = 1;
let bake = () => {
  cookie_amount += pointer_amount;
}
document.getElementById("cookie").addEventListener("click", (e) => {
  bake();
});
```

反映させる

- これではHTMLに(画面に)反映されないので反映させられるようにします
 - spanタグは直後に改行されない汎用コンテナだと考えておいて下さい
 - idを変数名と同じにしていますが, 同じにする必要はありません

index.html

```
<div>
  
  × <span id="cookie_amount">0</span>
</div>
<div>
  
  × <span id="pointer_amount">1</span>
</div>
```

反映させる

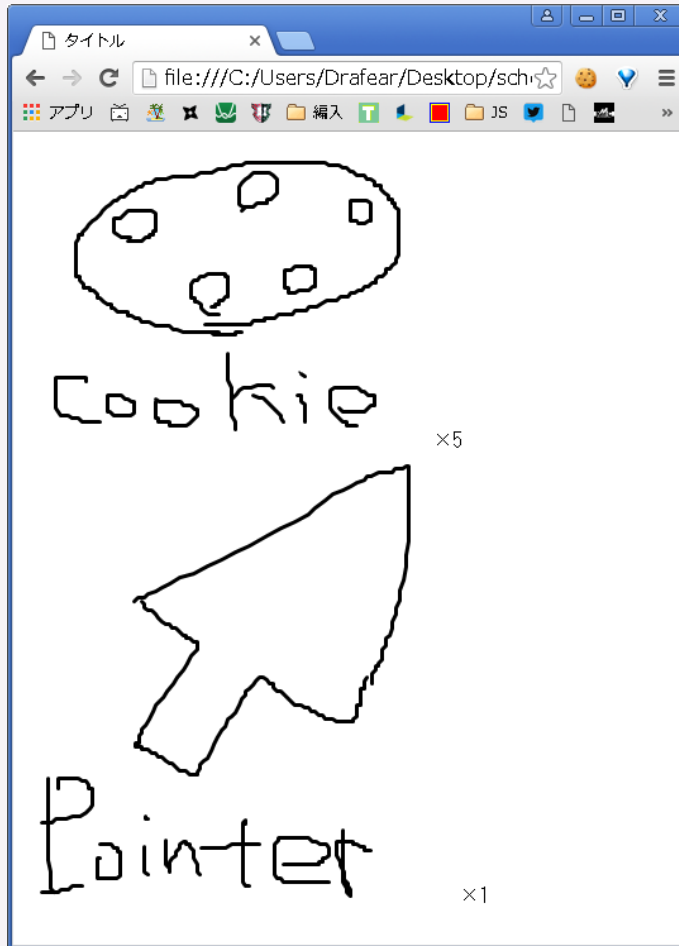
- これでおk
 - 要素の `innerText` をいじればタグで囲まれた中身のテキストが変わる

main.js

```
let cookie_amount = 0;
let pointer_amount = 1;
let bake = () => {
  cookie_amount += pointer_amount;
  update();
}
let update = () => {
  document.getElementById("cookie_amount").innerText = cookie_amount;
  document.getElementById("pointer_amount").innerText = pointer_amount;
}
document.getElementById("cookie").addEventListener("click", (e) => {
  bake();
});
```


焼けた！

- 焼けた！！！！！！！！！！！！！！！！！！！！



ぽいんたー買いたい

- ポインターを買えるようにします！

index.html

```
<div>
  
  × <span id="cookie_amount">0</span>
</div>
<div>
  
  × <span id="pointer_amount">1</span>
  <button id="btn_buypointer">購入 (50 Cookie)</button>
</div>
```

ぽいんたー買いたい

- ポインターを買えるようにします！

main.js

```
let buy = () => {  
  if (cookie_amount >= 50) {  
    cookie_amount -= 50;  
    pointer_amount += 1;  
    update();  
  }  
}  
document.getElementById("btn_buypointer").addEventListener("click", (e) => {  
  buy();  
})
```

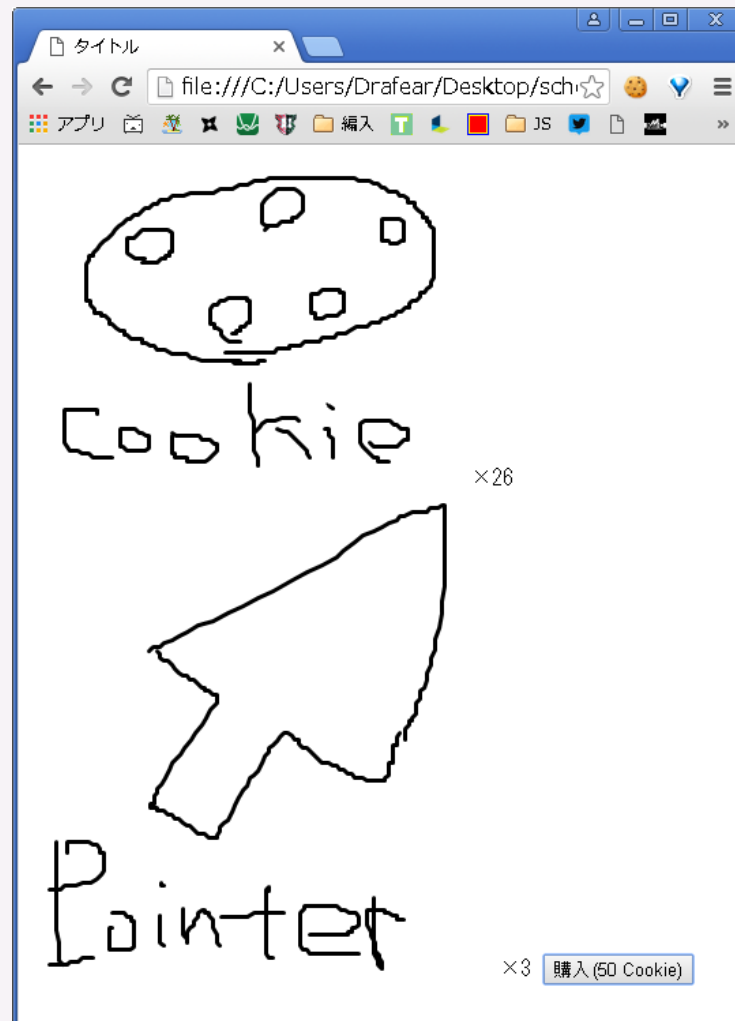
条件分岐

- if (条件) { 処理 }
 - 条件が成立した時のみ処理を行う
 - 次回詳しくやります

main.js(再掲)

```
let buy = () => {  
  if (cookie_amount >= 50) {  
    cookie_amount -= 50;  
    pointer_amount += 1;  
    update();  
  }  
}  
document.getElementById("btn_buypointer").addEventListener("click", (e) => {  
  buy();  
})
```

完成！



おまけ

- 値段変更時に3ヶ所もいじらないといけないのはつらいのでまとめます

main.js

```
let pointer_price = 50;
let buy = () => {
  if (cookie_amount >= pointer_price) {
    cookie_amount -= pointer_price;
    pointer_amount += 1;
    update();
  }
}
document.getElementById("btn_buypointer").innerText =
  `購入 (${pointer_price} Cookie)`;
document.getElementById("btn_buypointer").addEventListener("click", (e) => {
  buy();
});
```

おまけ

- テンプレート文字列
 - 文字列中に簡単に変数の値を埋め込める
 - バッククオート `` で囲んだ文字列中で `${...}` の部分を, `{ }` 内のコードを実行した結果で置き換える
 - 実行して結果を得ることを **評価する** という

main.js

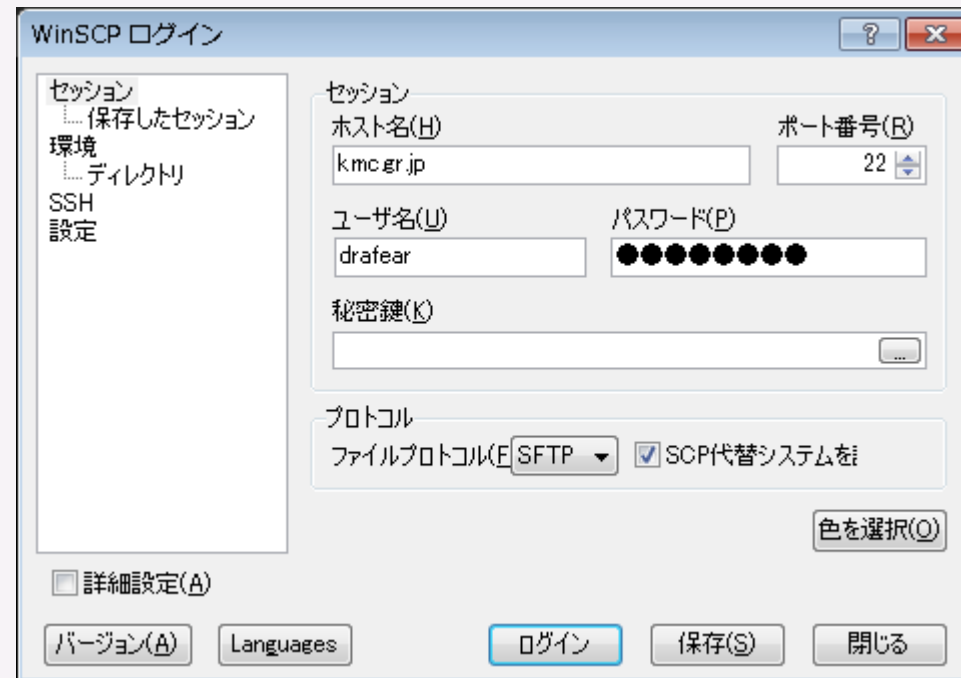
```
let a = 10;  
let b = 5;  
console.log(`${a} + ${b} = ${a + b}`); // 10 + 5 = 15  
console.log(`${a} + ${b} = ${a + b}`); // ${a} + ${b} = ${a + b}
```

作品をアップロードしてみよう

- WinSCPを使って内部ページにアップロードしてみる
 - <http://www.forest.impress.co.jp/library/software/winscp/>

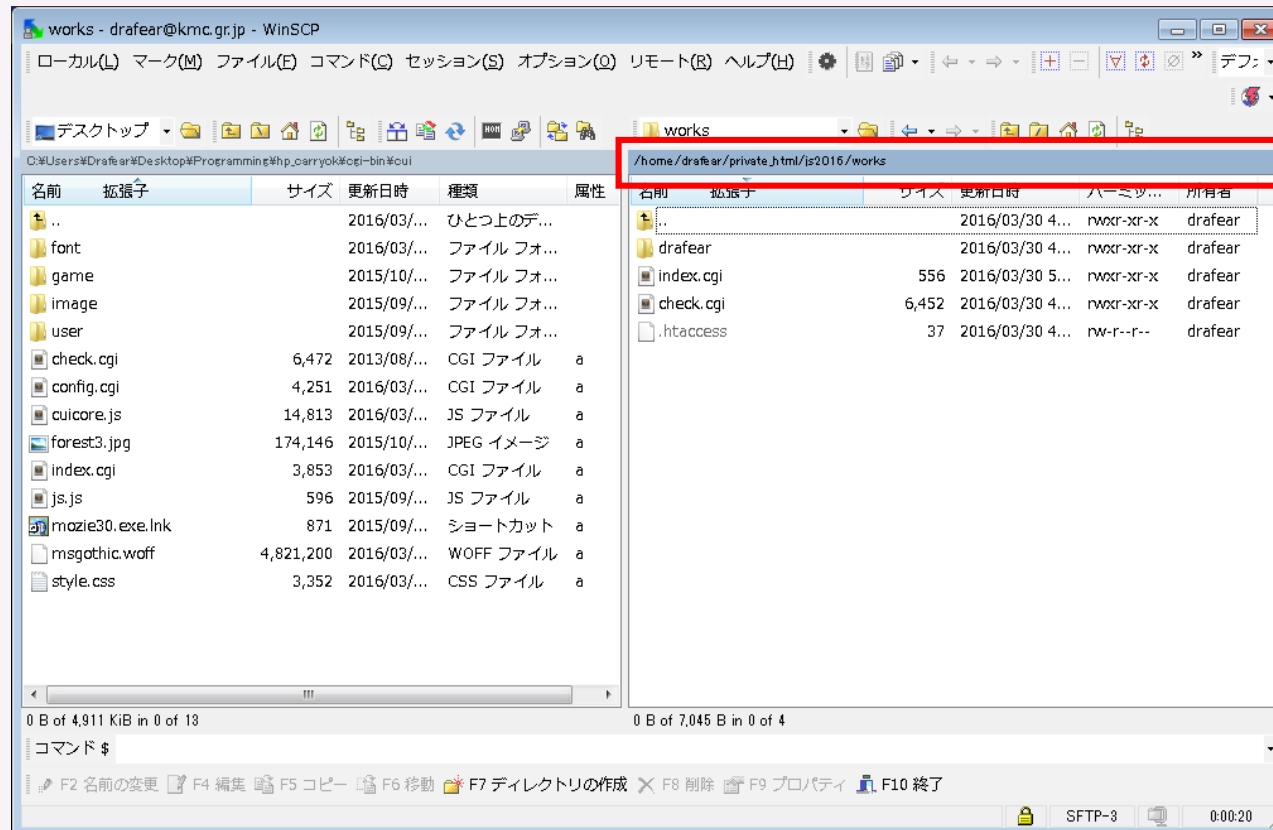
作品をアップロードしてみよう

- 次のように入れてログインして下さい
 - ユーザ名, パスワード は KMC の wiki にログインするときに用いるものを入れて下さい



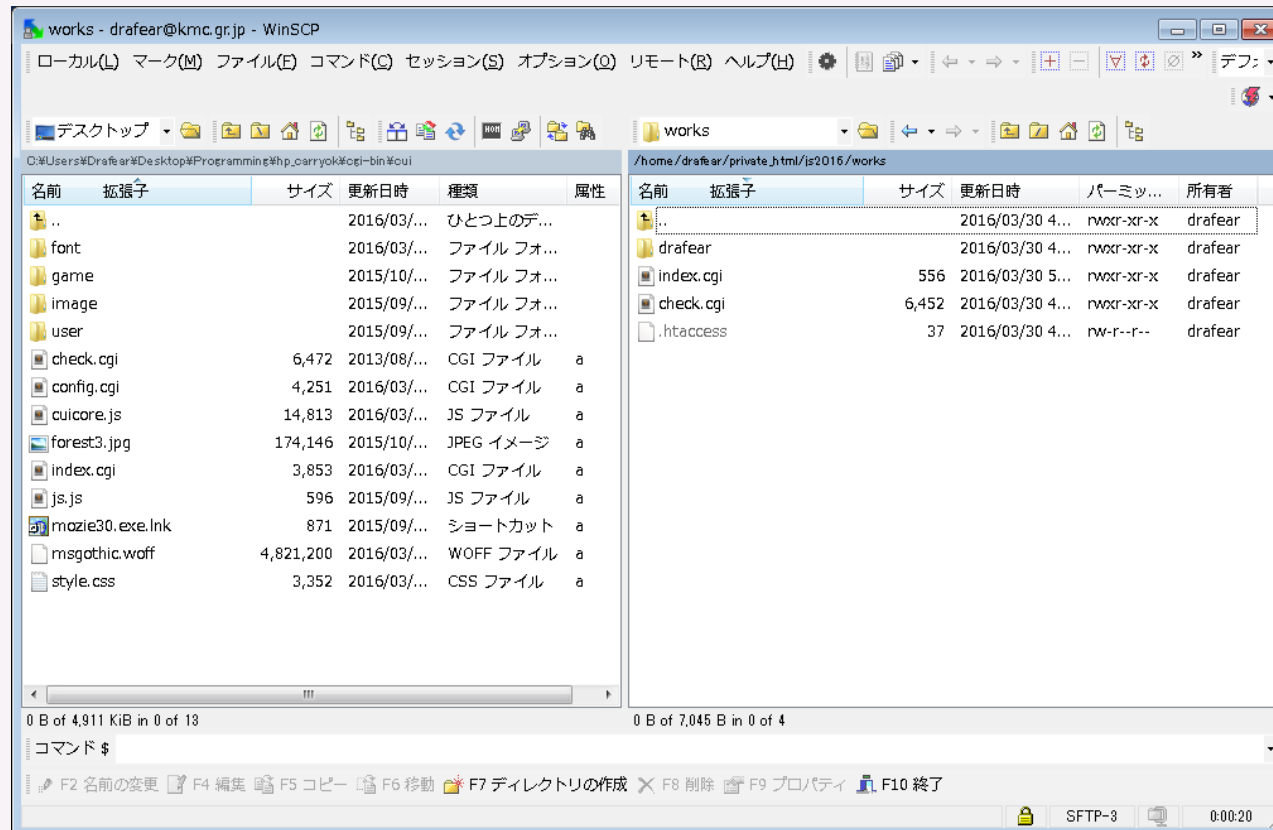
作品をアップロードしてみよう

- ここをダブルクリックして次のパスを入力して下さい
 - /home/drafeear/private_html/js2016/works



作品をアップロードしてみよう

- そこに自分のidの名前のディレクトリを作って,
その中に index.html と main.js と style.css を入れて下さい



次回

- 5/15(日) 13:00～
 - トップページを作る
 - CSSでデザインする
 - あの伝説のゲームを作る
 - 連打ゲームを作る
- 5/22(日) 13:00～
 - ○ubeatのようなゲームを作る
- 5/29(日) 13:00～
 - シューティングゲームを作る