

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, どらふあー)
- 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 ※青: 新入生プロジェクト
 - **これ**, **競プロ**, ctf, 終焉のC++, coq, 組み合わせ最適化読書会



@drafear



@drafear_ku



@drafear_carryok



@drafear_evolve



@drafear_sl



@gekimon_1d1a



@cuigames

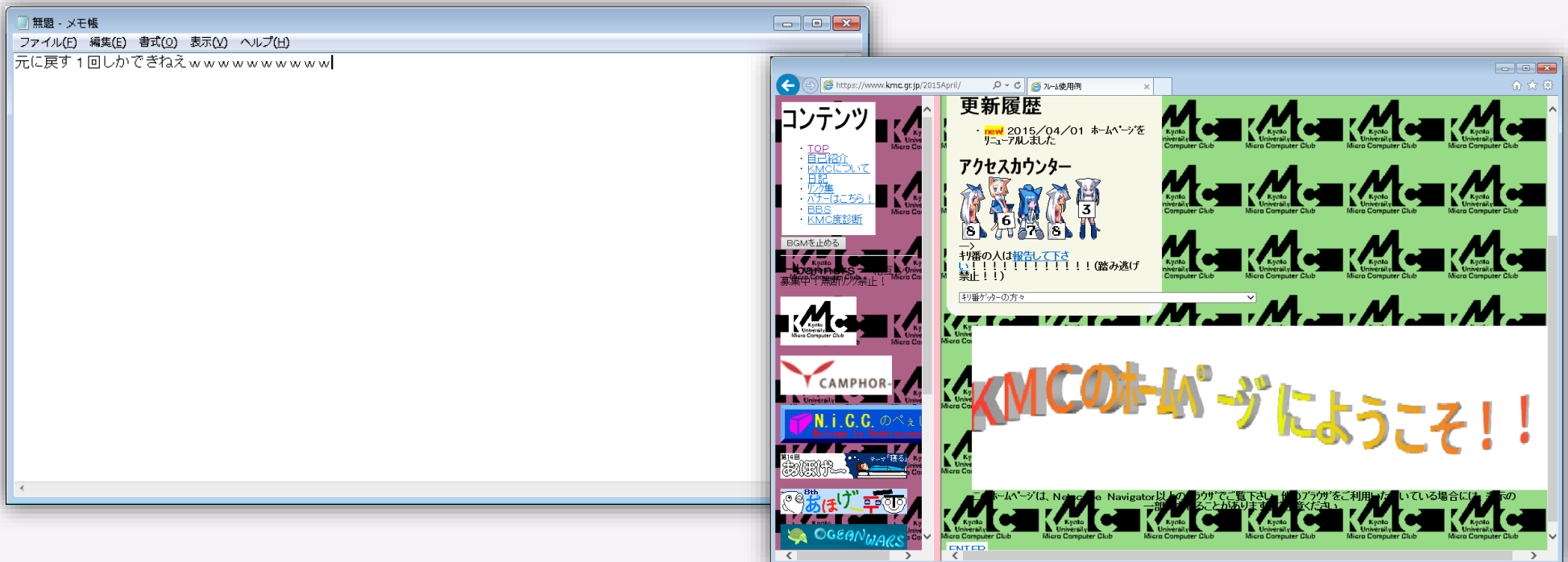
自己紹介

1. KMC 2回生
2. 新入生
3. KMC 3回生
4. KMC 4回生
5. KMC n回生


- 学部学科
- id (入部してたら) or 名前
- 趣味
- 特技(あれば)

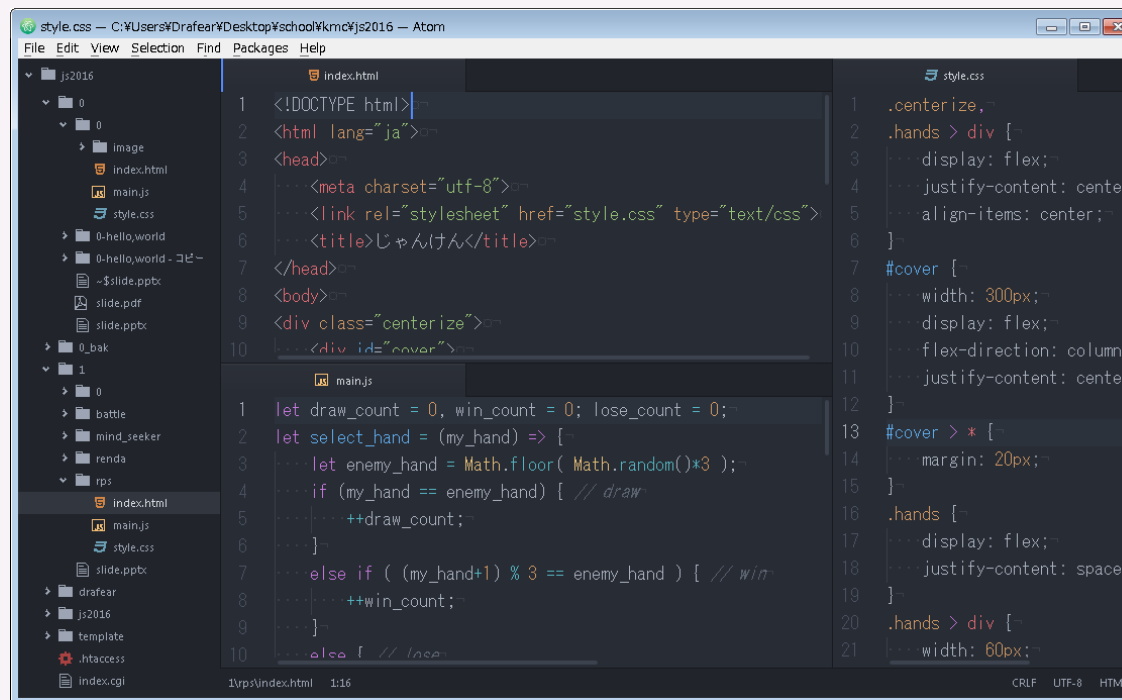
始める前に...

- 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>
- Atom 
 - <https://atom.io/>



atom 初期設定

- パッケージのインストール
 - Ctrl + , キーで設定画面を開きます
 - Install をクリックします
 - 「less-than-slash」を検索して, Install してみましょう
- ラピッドコーディング祭りに参加された方
 - autocomplete-plus を disable しましたが,
便利なので packages から検索して enable しましょう

エディタの使い方

- Ctrl + Z 元に戻す
- Ctrl + Y 元に戻すを戻す
- Ctrl + C コピー
- Ctrl + V ペースト(貼り付け)
- Ctrl + X 選択範囲切り取り or カーソル行切り取り
 - 切り取り・・・削除してコピー
- Ctrl + ↑ or ↓ 行入れ替え
- Ctrl + G 指定した行へ移動
- Ctrl + F 文字列検索, 文字列置換
- Shift + Tab インデントを1上げる

おねがい

- 分からないところがあれば, ちょっとした事でも, 前に説明があったかもしれないところでも 遠慮せずにどんどん聞いて下さい!

注意

- 知らないことがたくさん出てくるかもしれませんが全部覚える必要はありません
- こんなのがあったなーくらいでおkです
- 必要になれば調べれば良いので
- 重要な項目は、やってるうちに覚えます

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, HTML要素のテキスト変更ができる
- 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)

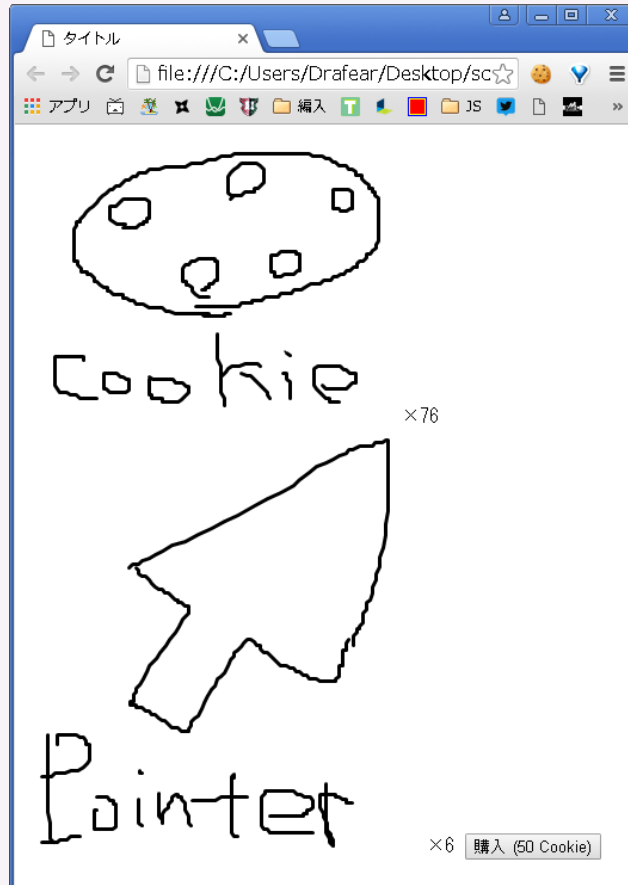
やっと本題

と言っても分からないと思うので

実際に触っていきましょう

今日作るもの

- Cookie Clicker のようなもの
 - <http://drafear.ie-t.net/js2016/cc/>



その前に

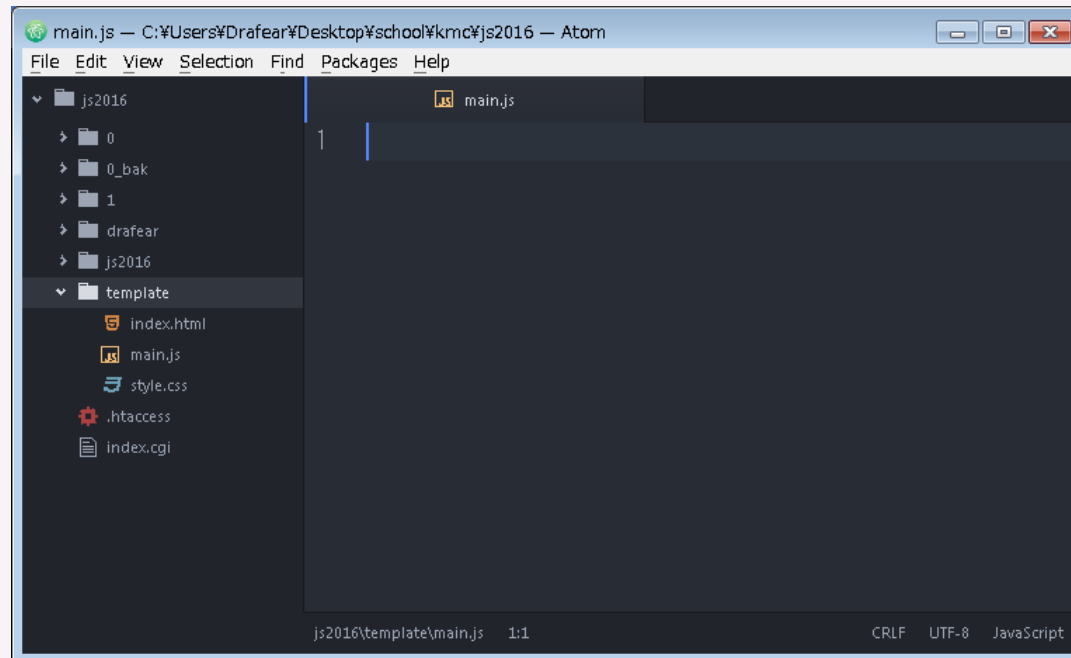
作り始める前に

まずは JavaScriptの基礎 を

勉強していきましょう

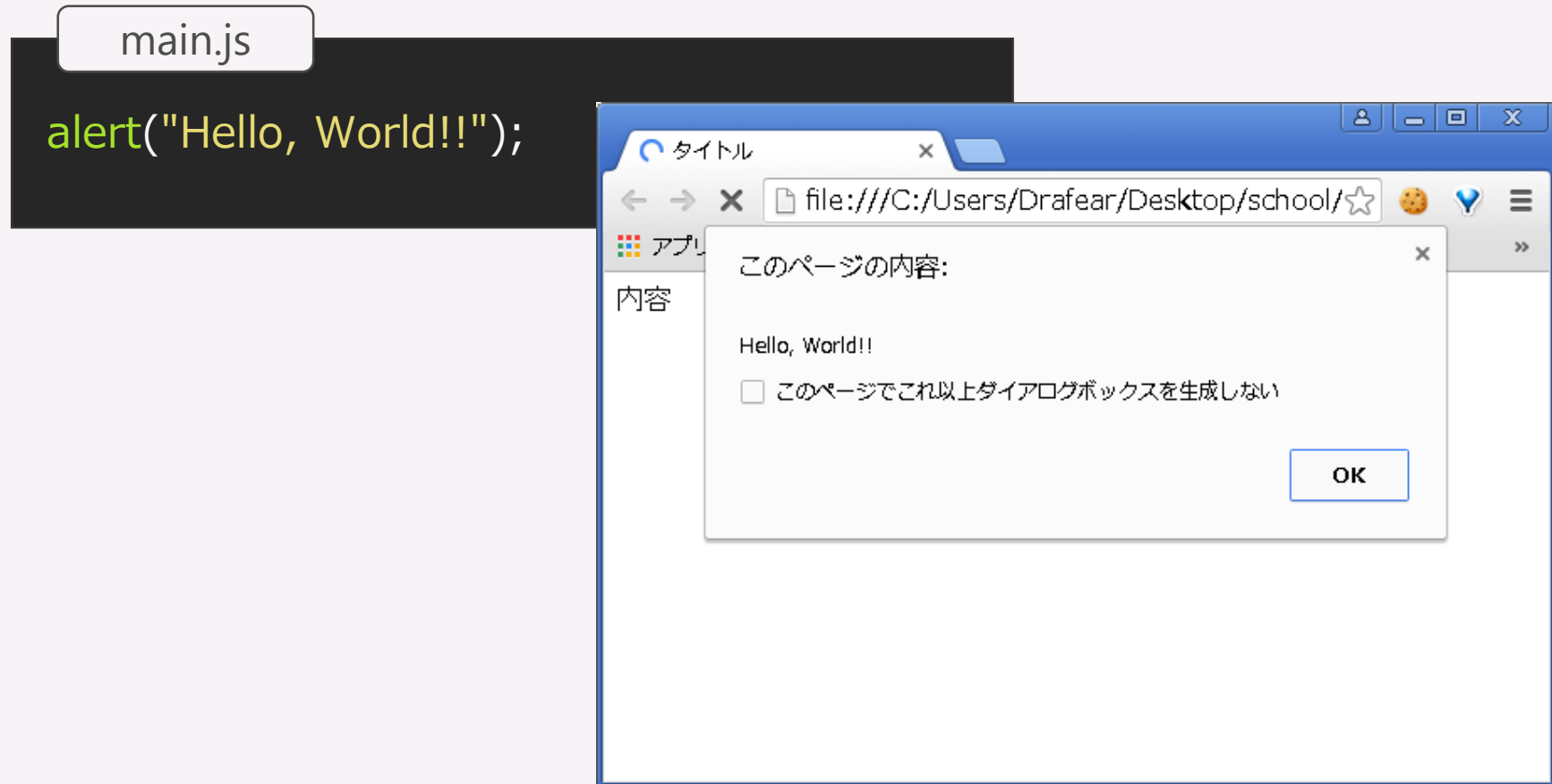
てんぷれ

- 以下から雛形をダウンロードしてください
 - <https://github.com/kmc-jp/js2016>
- ダウンロードできたら main.js を開いて下さい
- 画面にフォルダをドラッグ&ドロップするとサイドバーが出る



Hello, World!!

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

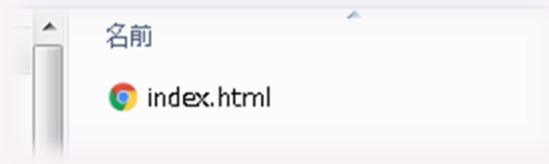
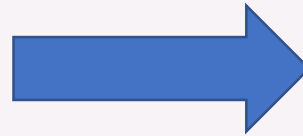
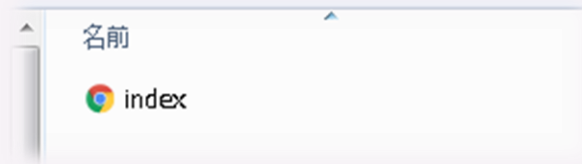


拡張子

- ".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

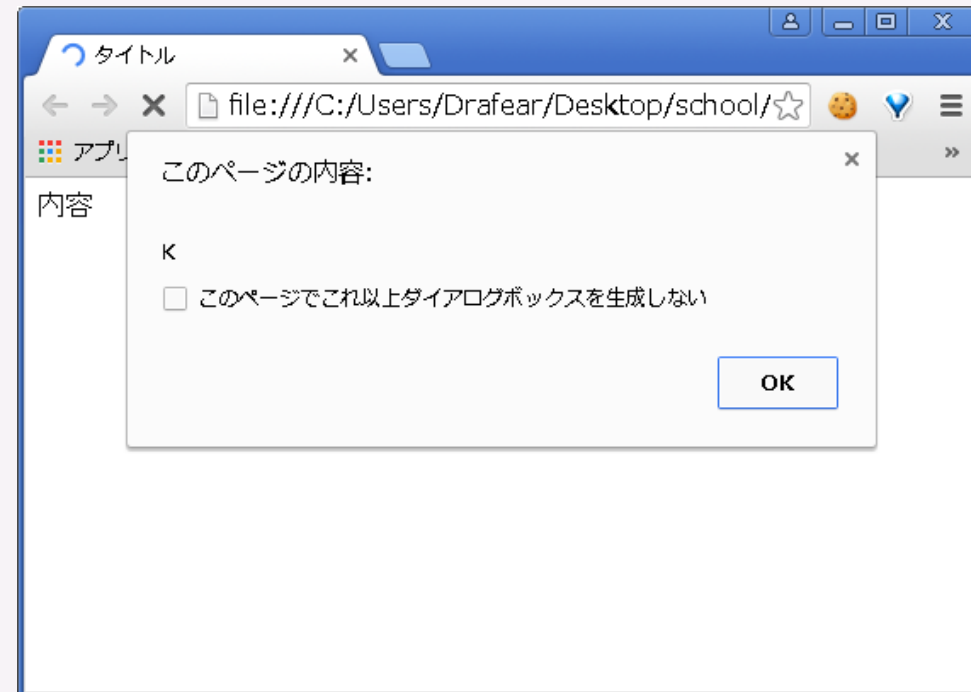
- "win8 拡張子 表示" などでggして下さい
 - コントロールパネルの「ファイルとフォルダーの検索オプションの変更」
「表示」タブから、「登録されている拡張子は表示しない」のチェックを外す



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");
```

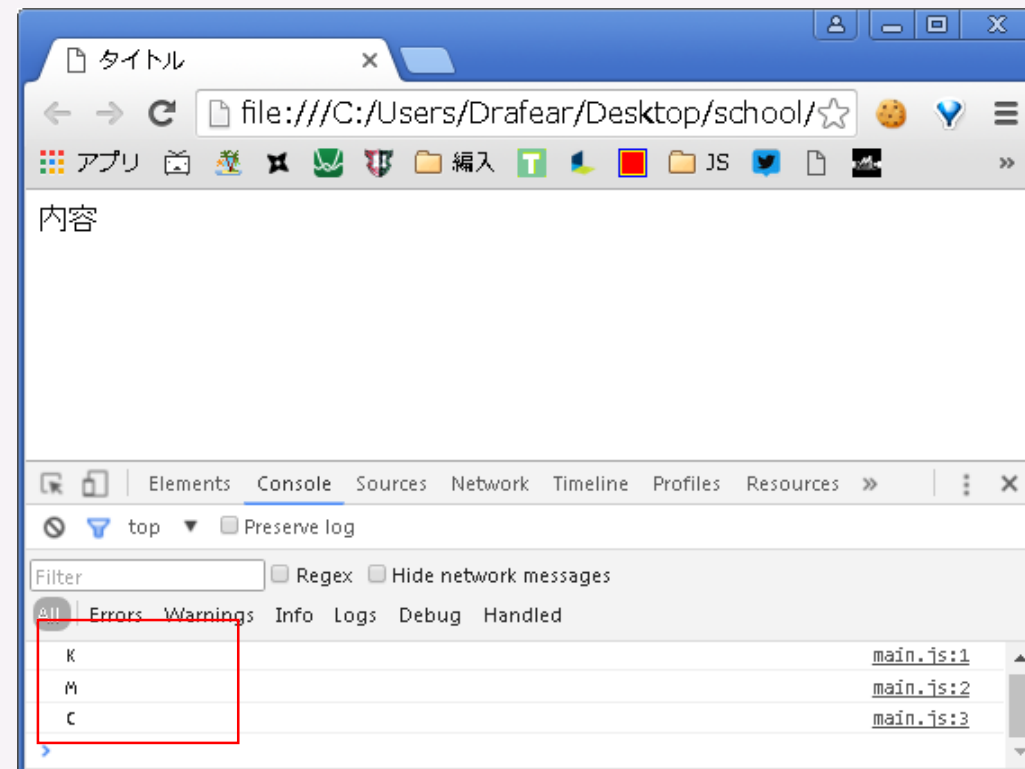


console.log

- 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** 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");
```

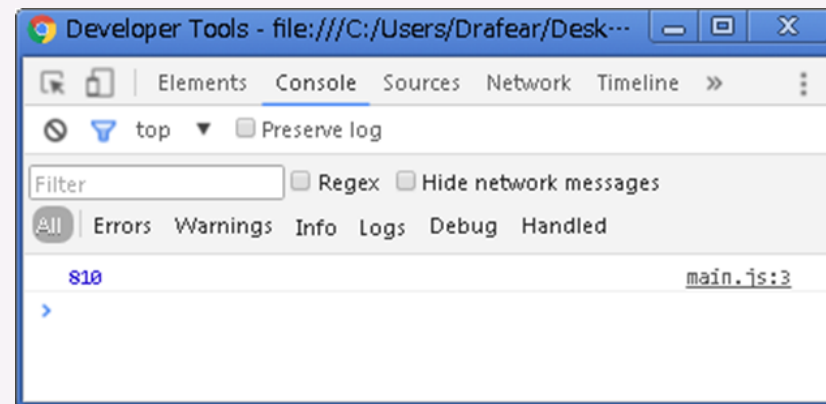
変数

- 変数
 - 変数とは, 1つの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
 - 変数名にはアルファベットなどが使えます
 - `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

- 変数
 - 変数とは, 1つの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
 - 変数名にはアルファベットなどが使えます
- 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 = 810;  
console.log(a);
```

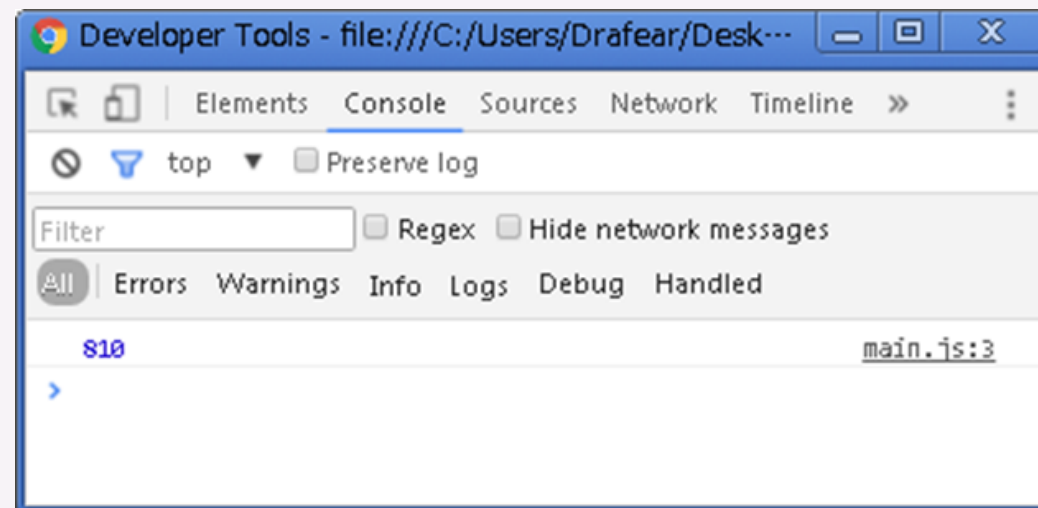


変数

- 変数
 - 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

main.js

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



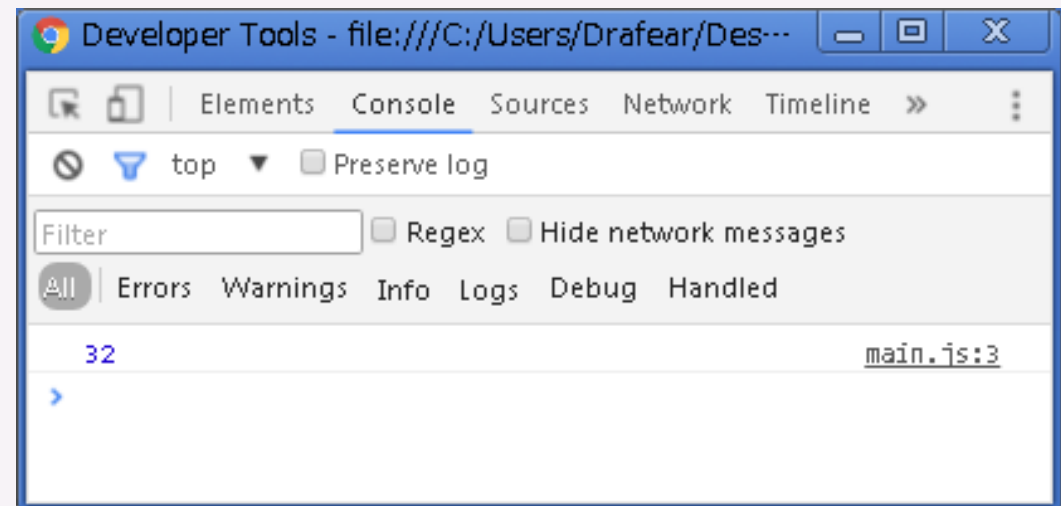
四則演算

- 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);
```

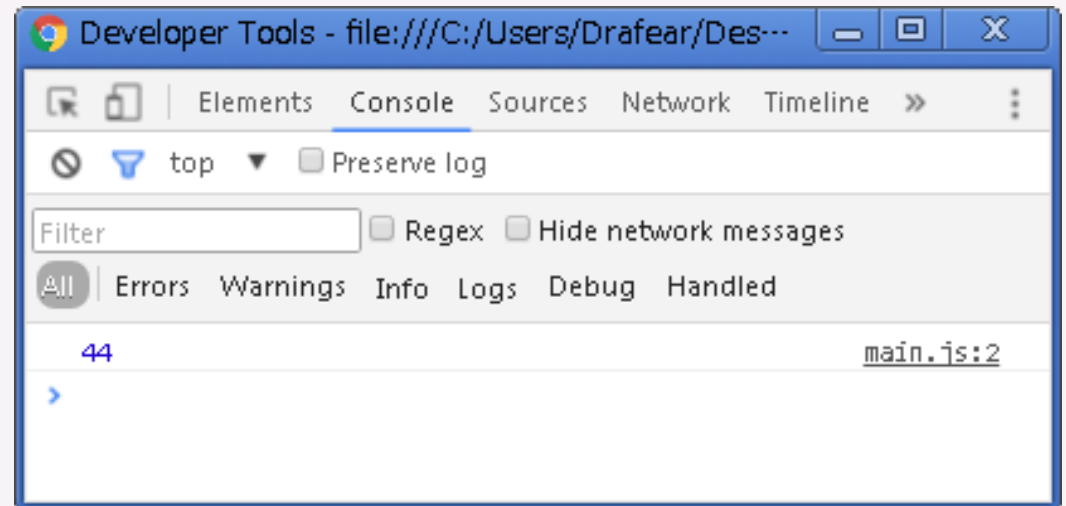


四則演算

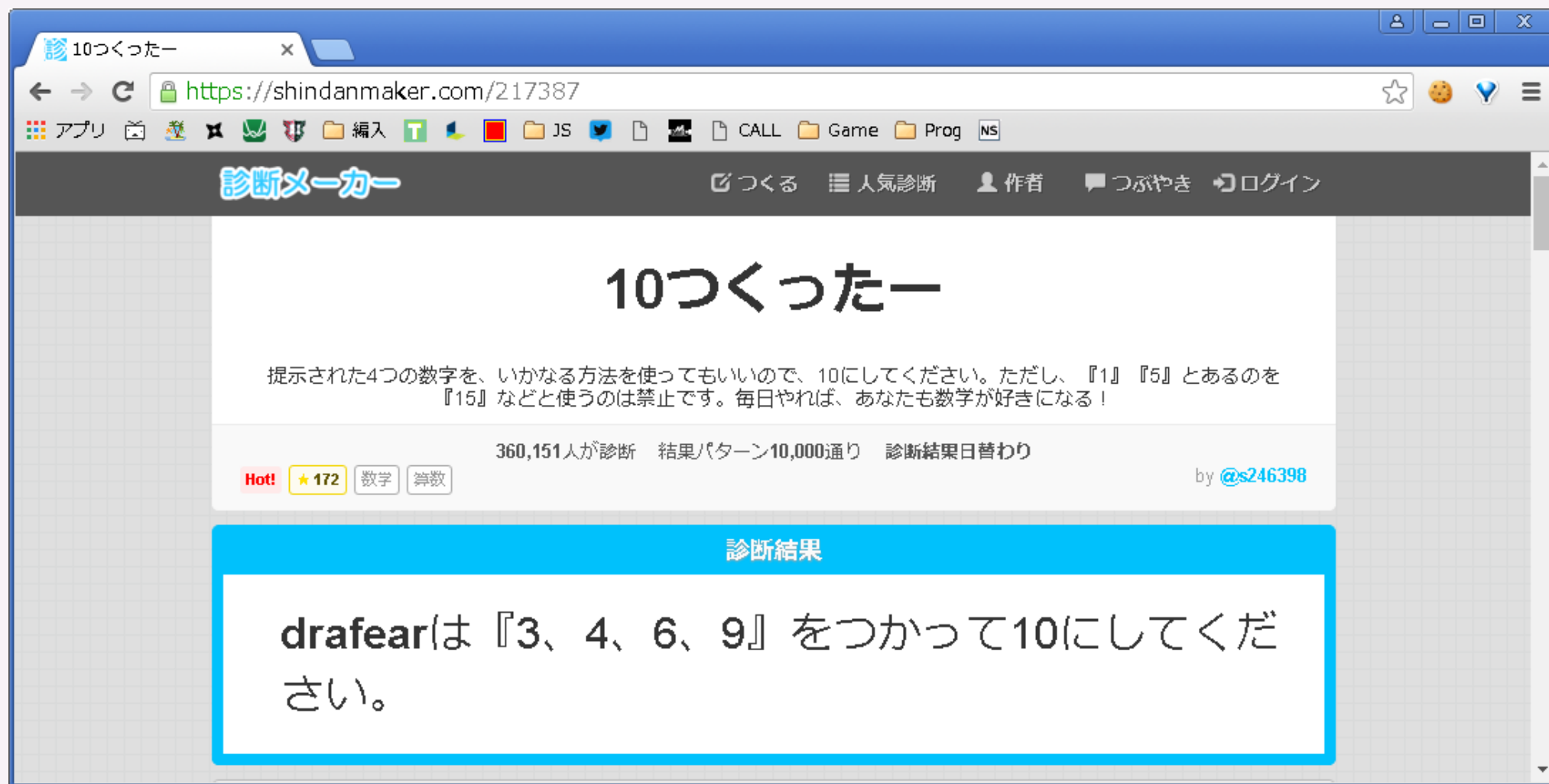
- 四則演算
 - ()内の演算が優先されます

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);
```

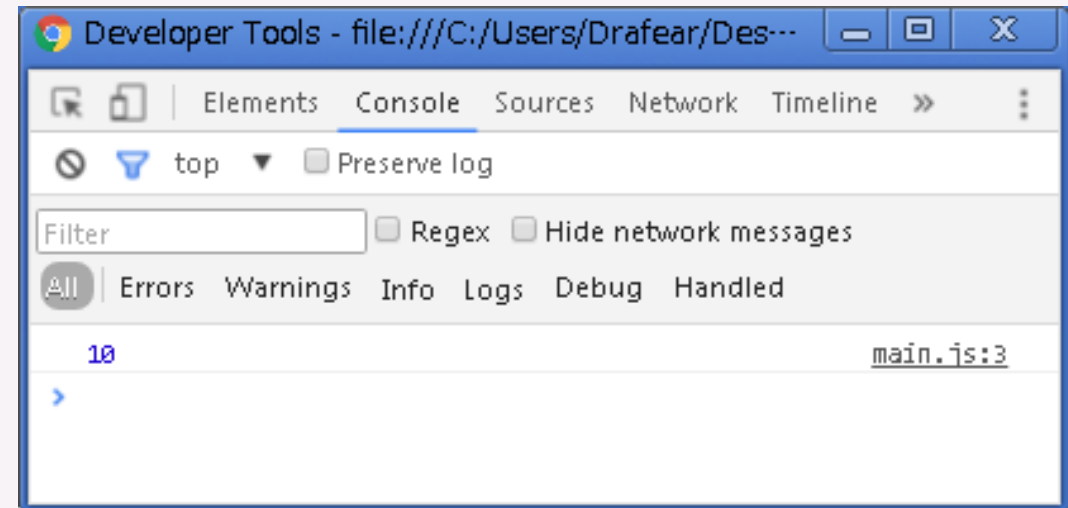
四則演算

• 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

- 他にも解はたくさんあります (交換法則などによる重複あり)

$9 - 6 + 4 + 3$	$9 \% 6 + 3 + 4$	$9 \% 6 + (3 + 4)$	$4 \% (3 * 9) + 6$	$3 + 9 - 6 \% 4$	$9 + (3 + 4 - 6)$
$9 \% 6 + 4 + 3$	$9 + 3 - 6 + 4$	$9 + 4 + 3 - 6$	$3 + (9 + 4) - 6$	$3 + 9 + (4 - 6)$	$9 + (3 + 4) \% 6$
$9 + 4 - 6 + 3$	$9 \% 3 + 6 + 4$	$9 \% 4 + 3 + 6$	$3 + 9 \% 4 + 6$	$9 + (6 + 4) \% 3$	$9 - (3 - 4) \% 6$
$(9 - 4) * 6 / 3$	$3 + 9 - 6 + 4$	$4 + 9 + 3 - 6$	$3 + (4 + 9) - 6$	$9 - (6 - 4 - 3)$	$9 + (4 - (6 - 3))$
$(9 * 4 - 6) / 3$	$9 - (6 - 3) + 4$	$9 + 3 + 4 - 6$	$9 + 4 - (6 - 3)$	$9 - (6 \% 4 - 3)$	$9 + 4 \% (6 - 3)$
$9 \% 4 + 6 + 3$	$9 + (3 - 6) + 4$	$9 \% 3 + 4 + 6$	$9 + 4 + (3 - 6)$	$9 + (4 + 6) \% 3$	$9 + (4 + (3 - 6))$
$4 + 9 - 6 + 3$	$9 - 3 \% 6 + 4$	$3 + 9 + 4 - 6$	$9 + 4 - 3 \% 6$	$9 + (4 - 6 + 3)$	$9 + 4 \% (3 - 6)$
$(4 * 9 - 6) / 3$	$3 - 6 + 9 + 4$	$9 + (4 + 3) - 6$	$(9 - 4) * (6 / 3)$	$9 + (4 \% 6 - 3)$	$9 + (4 - 3 \% 6)$
$9 - (6 - 4) + 3$	$6 \% (9 + 3) + 4$	$9 + (3 + 4) - 6$	$9 \% 4 + (6 + 3)$	$9 + 4 \% 6 \% 3$	$9 + 4 \% (3 \% 6)$
$9 - 6 \% 4 + 3$	$6 \% (9 * 3) + 4$	$4 + 3 + 9 - 6$	$9 \% 4 + (3 + 6)$	$9 + (6 + 3) \% 4$	$9 + (3 - (6 - 4))$
$9 + (4 - 6) + 3$	$6 + 9 \% 3 + 4$	$3 + 4 + 9 - 6$	$4 + 9 - (6 - 3)$	$9 - (6 - 3 - 4)$	$9 + 3 \% (6 - 4)$
$9 + 4 \% 6 - 3$	$6 - 9 \% 3 + 4$	$4 + (9 + 3) - 6$	$4 + 9 + (3 - 6)$	$9 + (3 + 6) \% 4$	$9 + (3 - 6 \% 4)$
$4 - 6 + 9 + 3$	$6 \% (3 + 9) + 4$	$4 \% (9 + 3) + 6$	$4 + 9 - 3 \% 6$	$9 + (3 - 6 + 4)$	$9 + 3 \% (6 \% 4)$
$4 \% 6 + 9 - 3$	$6 \% (3 * 9) + 4$	$4 \% (9 - 3) + 6$	$9 + 3 - (6 - 4)$	$9 - (3 \% 6 - 4)$	$9 + (3 + (4 - 6))$
$6 * (9 - 4) / 3$	$3 + (9 - 6) + 4$	$4 \% (9 * 3) + 6$	$9 + 3 - 6 \% 4$	$9 - (6 - (4 + 3))$	$9 + 3 \% (4 - 6)$
$6 + 9 \% 4 + 3$	$3 + 9 \% 6 + 4$	$4 + 9 \% 3 + 6$	$9 + 3 + (4 - 6)$	$9 - (6 - (3 + 4))$	$9 - (3 - 4 \% 6)$
$4 + (9 - 6) + 3$	$3 - (6 - 9) + 4$	$4 - 9 \% 3 + 6$	$9 - 3 + 4 \% 6$	$9 + (4 + 3 - 6)$	$(6 + 4) \% 3 + 9$
$4 + 9 \% 6 + 3$	$9 - 6 + (4 + 3)$	$4 + (3 + 9) - 6$	$9 \% 3 + (6 + 4)$	$9 + (4 + 3) \% 6$	$(4 + 6) \% 3 + 9$
$4 - (6 - 9) + 3$	$9 - 6 + (3 + 4)$	$4 \% (3 + 9) + 6$	$9 \% 3 + (4 + 6)$	$9 + (4 - 3) \% 6$	$4 - 6 + 3 + 9$
$9 - 6 + 3 + 4$	$9 \% 6 + (4 + 3)$	$4 \% (3 - 9) + 6$	$3 + 9 - (6 - 4)$	$9 + 4 \% 3 \% 6$	$4 \% 6 - 3 + 9$

四則演算

- 他にも解はたくさんあります (交換法則などによる重複あり)

$4 \% 6 \% 3 + 9$	$3 + (4 - 6) + 9$	$3 + 6 + 9 \% 4$	$3 + 4 - (6 - 9)$	$4 + 6 \% (3 + 9)$	$3 + (4 + (9 - 6))$
$(6 + 3) \% 4 + 9$	$3 \% (4 - 6) + 9$	$3 - 6 + (9 + 4)$	$4 + (9 - 6 + 3)$	$4 + 6 \% (3 * 9)$	$3 + (4 + 9 \% 6)$
$(3 + 6) \% 4 + 9$	$(6 + 4) \% (9 + 3)$	$3 - 6 + (4 + 9)$	$4 + (9 \% 6 + 3)$	$4 + (3 + (9 - 6))$	$3 + (4 - (6 - 9))$
$3 - 6 + 4 + 9$	$(6 + 4) \% (9 * 3)$	$6 + (9 \% 4 + 3)$	$4 - (6 - 9 - 3)$	$4 + (3 + 9 \% 6)$	
$4 + 3 - 6 + 9$	$6 + 4 + 9 \% 3$	$6 + (9 \% 3 + 4)$	$4 + (9 + 3 - 6)$	$4 + (3 - (6 - 9))$	
$(4 + 3) \% 6 + 9$	$6 + 4 - 9 \% 3$	$6 - (9 \% 3 - 4)$	$4 + (9 \% 3 + 6)$	$3 + (9 - 6 + 4)$	
$(4 - 3) \% 6 + 9$	$(6 + 4) \% (3 + 9)$	$6 + 4 \% (9 + 3)$	$4 - (9 \% 3 - 6)$	$3 + (9 \% 6 + 4)$	
$4 \% 3 \% 6 + 9$	$(6 + 4) \% (3 * 9)$	$6 + 4 \% (9 - 3)$	$4 + (3 + 9 - 6)$	$3 - (6 - 9 - 4)$	
$3 + 4 - 6 + 9$	$(4 + 6) \% (9 + 3)$	$6 + 4 \% (9 * 3)$	$4 + (9 - (6 - 3))$	$3 + (9 + 4 - 6)$	
$(3 + 4) \% 6 + 9$	$(4 + 6) \% (9 * 3)$	$6 + (4 + 9 \% 3)$	$4 + (9 + (3 - 6))$	$3 + (9 \% 4 + 6)$	
$4 - (6 - 3) + 9$	$4 + 6 + 9 \% 3$	$6 + (4 - 9 \% 3)$	$4 + (9 - 3 \% 6)$	$3 + (4 + 9 - 6)$	
$4 \% (6 - 3) + 9$	$4 + 6 - 9 \% 3$	$6 + 4 \% (3 + 9)$	$4 - (6 - 3 - 9)$	$3 + (9 - (6 - 4))$	
$4 + (3 - 6) + 9$	$(4 + 6) \% (3 + 9)$	$6 + 4 \% (3 - 9)$	$4 + (3 - 6 + 9)$	$3 + (9 - 6 \% 4)$	
$4 \% (3 - 6) + 9$	$(4 + 6) \% (3 * 9)$	$6 + 4 \% (3 * 9)$	$4 - (3 \% 6 - 9)$	$3 + (9 + (4 - 6))$	
$4 - 3 \% 6 + 9$	$4 - 6 + (9 + 3)$	$6 + (3 + 9 \% 4)$	$4 - (6 - (9 + 3))$	$3 - (6 - 4 - 9)$	
$4 \% (3 \% 6) + 9$	$4 - 6 + (3 + 9)$	$4 + 3 + (9 - 6)$	$4 + 6 \% (9 + 3)$	$3 - (6 \% 4 - 9)$	
$3 - (6 - 4) + 9$	$4 \% 6 + (9 - 3)$	$4 + 3 + 9 \% 6$	$4 + 6 \% (9 * 3)$	$3 + (4 - 6 + 9)$	
$3 \% (6 - 4) + 9$	$4 \% 6 - (3 - 9)$	$4 + 3 - (6 - 9)$	$4 + (6 + 9 \% 3)$	$3 - (6 - (9 + 4))$	
$3 - 6 \% 4 + 9$	$6 + 3 + 9 \% 4$	$3 + 4 + (9 - 6)$	$4 + (6 - 9 \% 3)$	$3 + (6 + 9 \% 4)$	
$3 \% (6 \% 4) + 9$	$6 / 3 * (9 - 4)$	$3 + 4 + 9 \% 6$	$4 - (6 - (3 + 9))$	$3 - (6 - (4 + 9))$	

四則演算

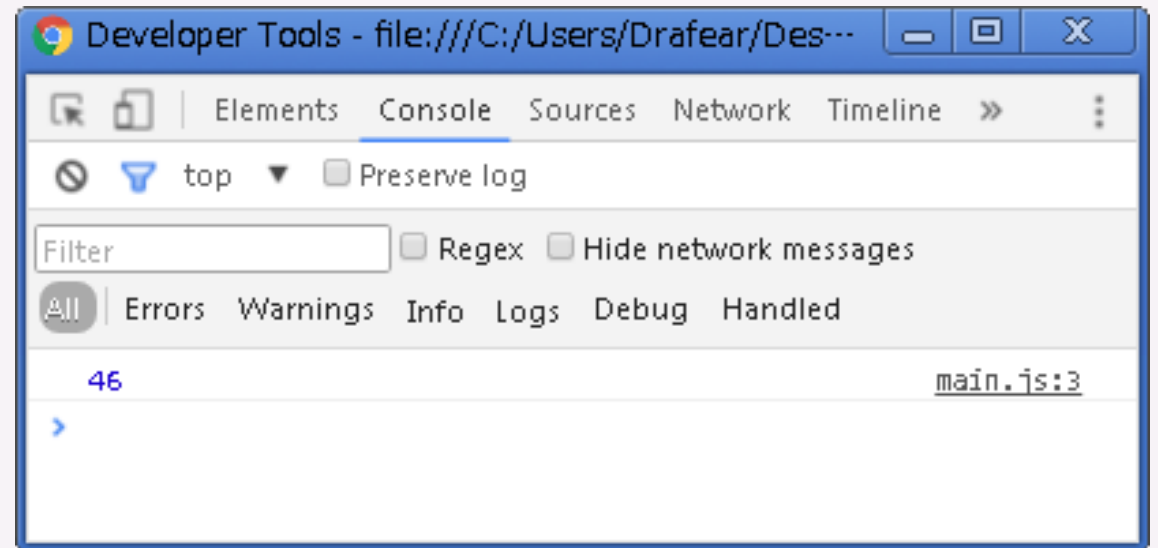
- 四則演算

- 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);
```

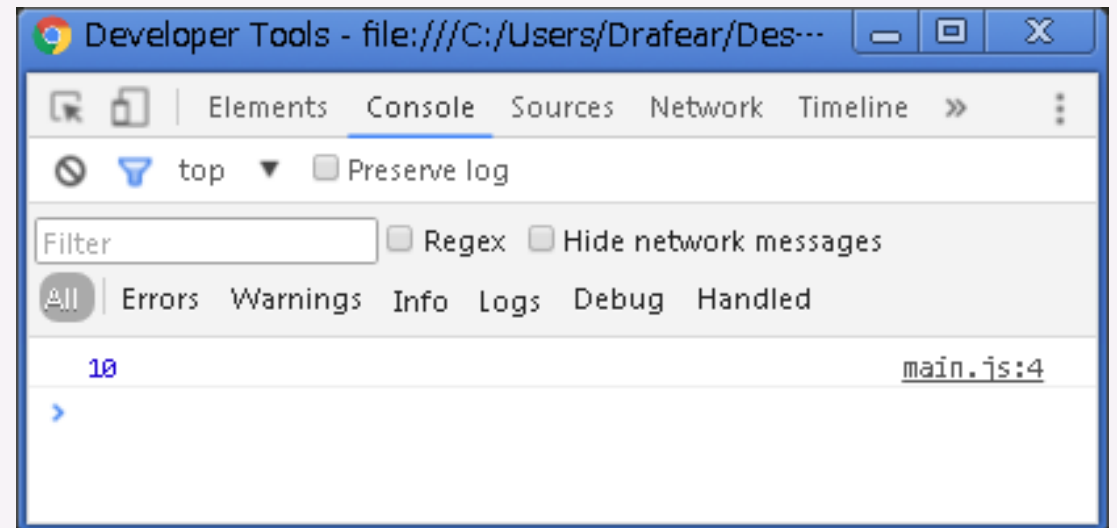


コメント

- 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ません
- 「//」以降の その行の文字はコメントとなります
- 「/*」と「*/」で囲まれた部分はコメントとなります
- atomでは, コメント化したい行を選択し, **Ctrl + /** を押せばコメント化(コメントアウト)できます

```
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います
- (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います

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 printSum = (a, b) => {
  console.log(a + b);
}
// 15 + 21 を出力する (36が出力される)
printSum(15, 21);
```

(一応)仮引数(parameter)
といいます

(一応)実引数(argument)といいます

関数

- 関数

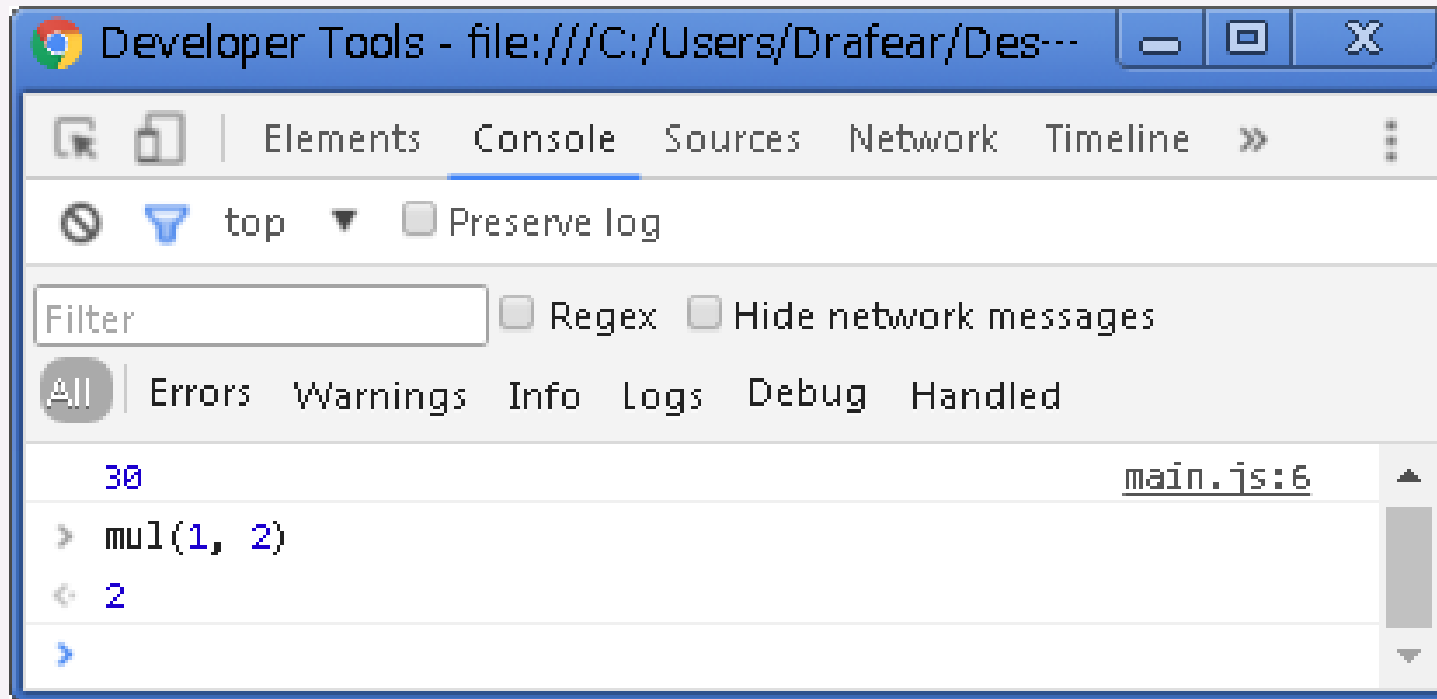
- (仮引数1, 仮引数2, ..., 仮引数n) => { 処理 }
- 関数は return 文により値を返すことができ,
返される値を 戻り値, 返り値, 返却値 などと呼びます

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. a を受け取って, a を2倍した値 を返す関数 `double` を書いてみましょう
 - 引数は a の1つ
3. x を受け取って x^3 を計算する関数 `cube` 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

main.js

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

1. a を受け取って, a を2倍した値 を返す関数 **double** を書いてみましょう
- $2a$ と書かないようにしましょう

main.js

```
let double = (a) => {  
  return 2 * a;  
}
```

演習

3. x を受け取って x^3 を計算する関数 `cube` を書いてみましょう
- x^3 と書かないようにしましょう

main.js

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

文字列

- 文字列

- 文字列は "文字列" (ダブルクォーテーション)
または '文字列' (シングルクォーテーション)
または `文字列` (バッククォート) 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  
};
```

```
console.log(human.name); // drafear  
console.log(human["age"]); // 21
```

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,  
  skillLevel: {  
    js: 0,  
    "c++": 1,  
  },  
  talk: () => {  
    console.log("Hello!");  
  }  
};
```

```
console.log(human.skillLevel.js); // 0  
console.log(human.skillLevel["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
```

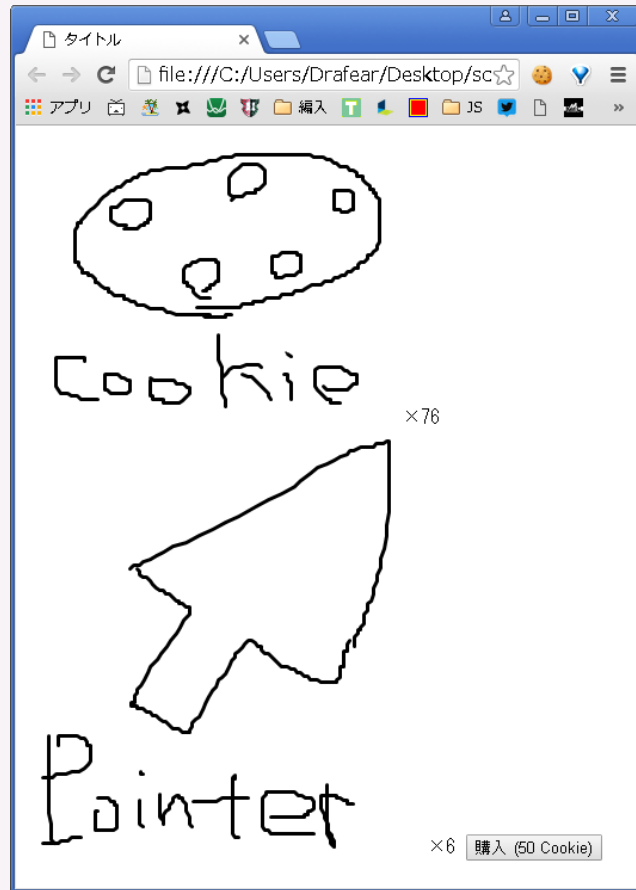
演習

- 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
```

Cookie Clicker作っていきましょう

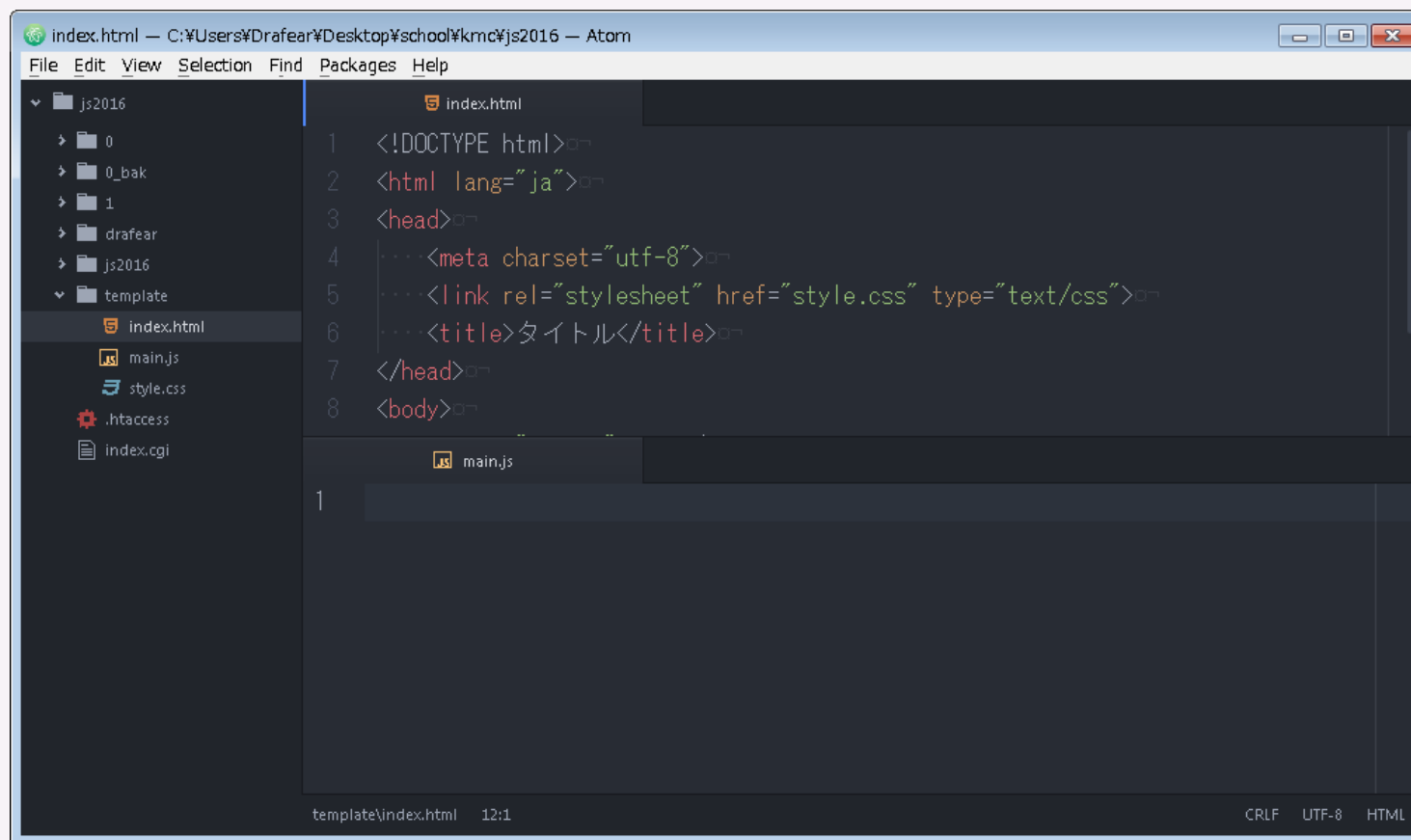


Start HTML

- というわけでHTMLを触っていきます

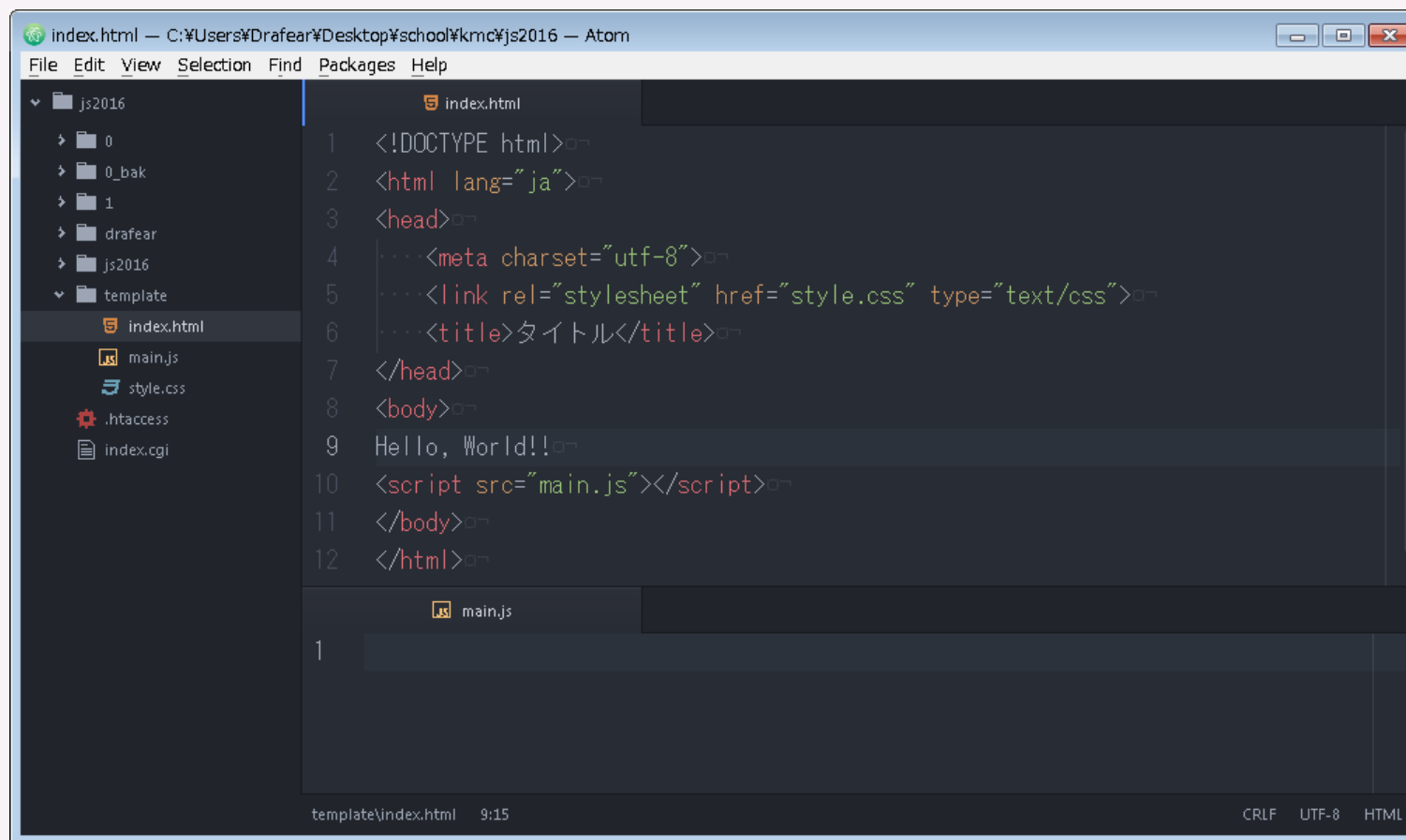
分割！

- 右クリック → [Split Down] で 二分割して
上に index.html, 下に main.js を読み込みましょう



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



```
index.html — C:\Users\Ydrafeary\Desktop\school\kmc\js2016 — Atom
File Edit View Selection Find Packages Help
└─ js2016
  └─ 0
  └─ 0_bak
  └─ 1
  └─ drafear
  └─ js2016
  └─ template
    └─ index.html
    └─ main.js
    └─ style.css
    └─ .htaccess
    └─ index.cgi

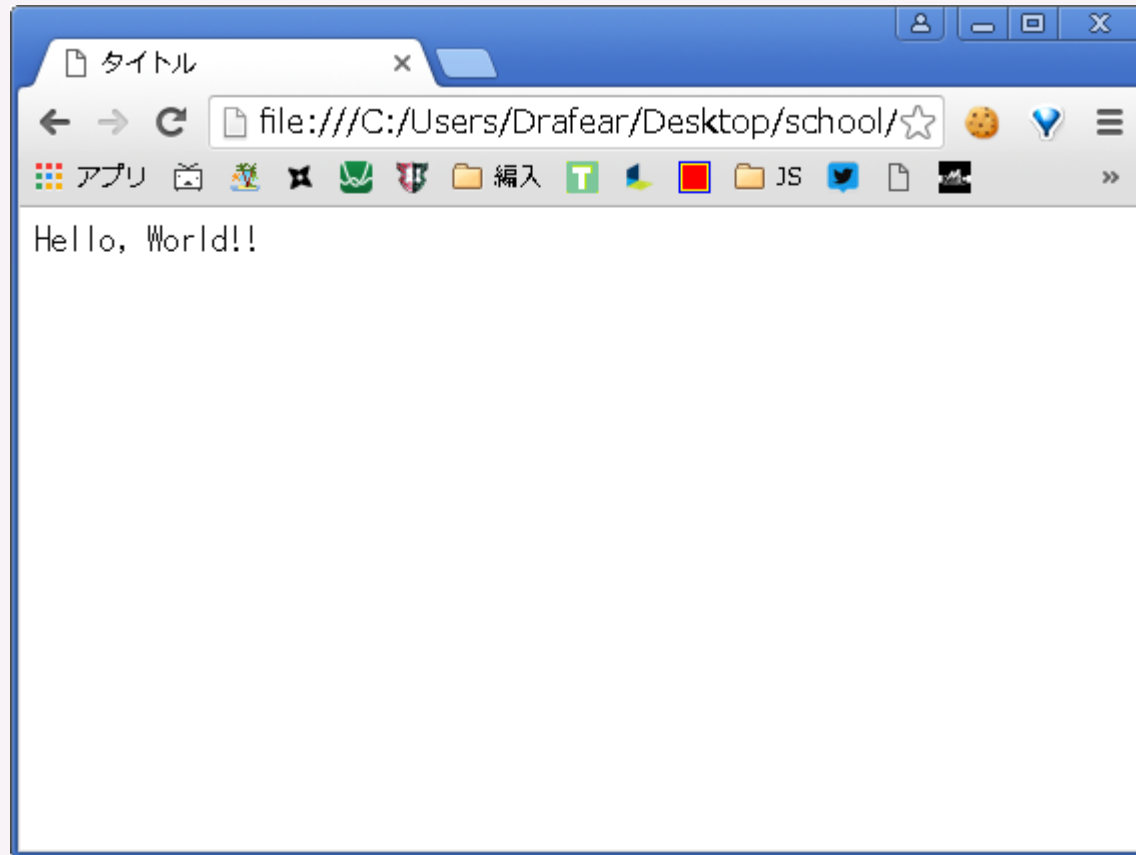
index.html
1 <!DOCTYPE html>
2 <html lang="ja">
3 <head>
4 <<<meta charset="utf-8">
5 <<<link rel="stylesheet" href="style.css" type="text/css">
6 <<<title>タイトル</title>
7 </head>
8 <body>
9 Hello, World!!
10 <script src="main.js"></script>
11 </body>
12 </html>

main.js
1

template\index.html 9:15 CRLF UTF-8 HTML
```

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

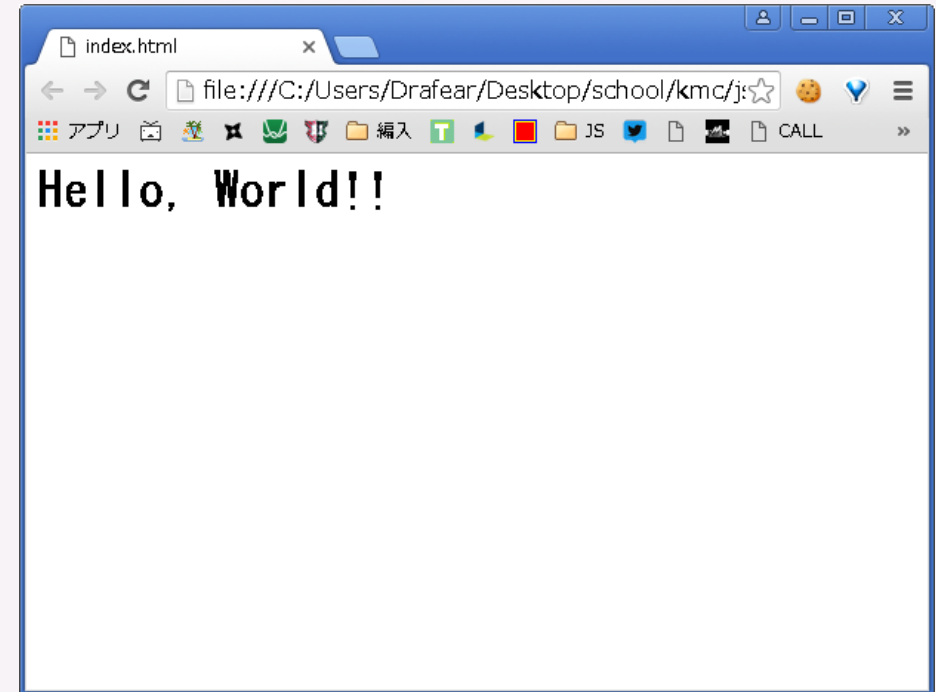
- Hello, World!! を<h1>~</h1>で囲むと...!?

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>
```

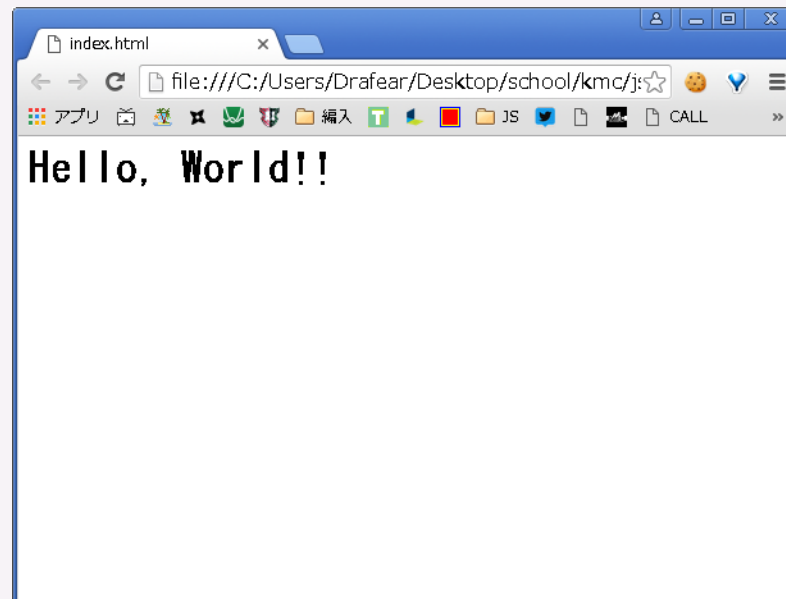
Hello, Worldを装飾しよう

- 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)



Hello, Worldを装飾しよう

- <タグ名>内容</タグ名>
 - タグと内容で1つのHTML要素となります
- 要素には意味があり, 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。



画像を貼ってみよう

- 属性

- ``のsrc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

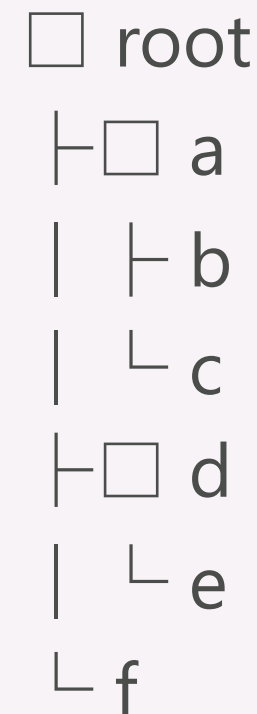
- srcに指定する先は、絶対パスまたは相対パスで書きます
- 絶対パスとは、`https://www.google.co.jp/`のような、絶対的なものを指します
- 相対パスとは、今のディレクトリから相対的に指定する方法です

相対パス

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

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



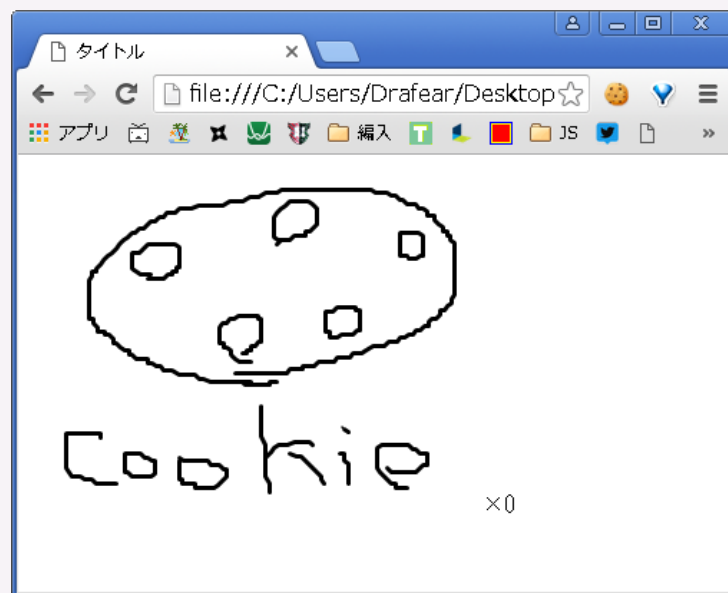
※□はディレクトリです

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

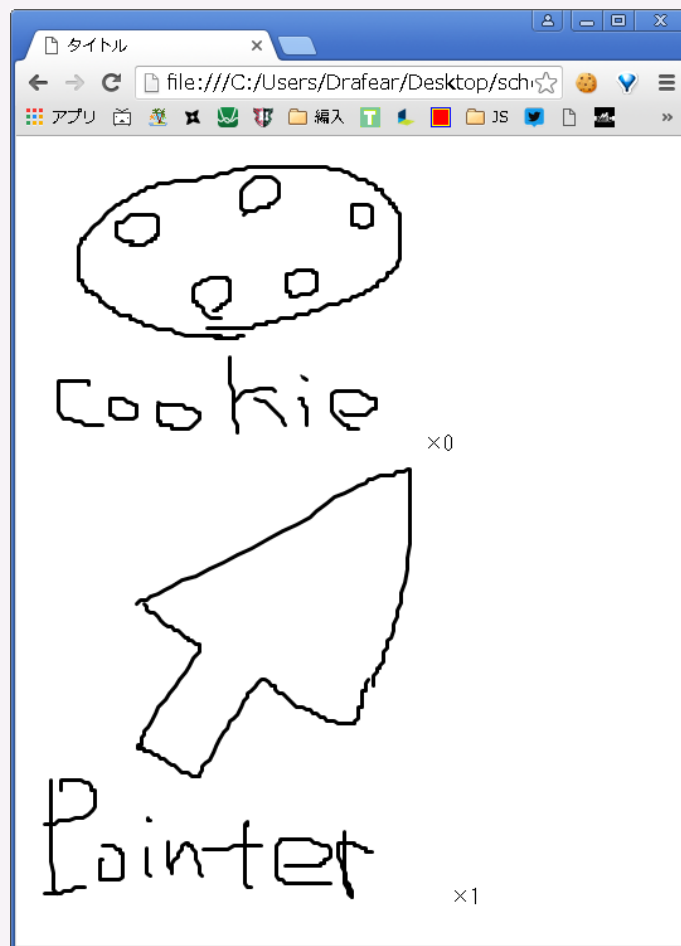
- bodyの中を次のように変更します
 - `</body>` 前の `<script ...>` `</script>` は残しておいて下さい
- `div`要素は汎用コンテナで, とりあえずは直後に改行される空のタグだと考えてもらえたらおkです

index.html

```
<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~~

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 elemCookie = document.getElementById("cookie");
elemCookie.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
 - 要素にマウスを重ねるとイベントが発生
2. "click" を "dragstart" に変えてみよう
 - 要素をドラッグし始めるとイベントが発生
3. "click" を "dblclick" に変えてみよう
 - 要素をダブルクリックするとイベントが発生

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 cookieAmount = 0; // クッキー所持数
let pointerAmount = 1; // ポインター所持数
let bake = () => { // クッキーを焼く
  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 cookieAmount = 0; // クッキー所持数
let pointerAmount = 1; // ポインター所持数
let bake = () => { // クッキーを焼く
    cookieAmount += pointerAmount;
}
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要素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="cookieAmount">0</span>
</div>
<div>
  
  × <span id="pointerAmount">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 cookieAmount = 0; // クッキー所持数
let pointerAmount = 1; // ポインター所持数
let bake = () => { // クッキーを焼く
  cookieAmount += pointerAmount;
  update();
}
let update = () => { // 変数の値を画面に反映する
  document.getElementById("cookieAmount").innerText = cookieAmount;
  document.getElementById("pointerAmount").innerText = pointerAmount;
}
document.getElementById("cookie").addEventListener("click", (e) => {
  bake();
});
```


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="cookieAmount">0</span>
</div>
<div>
  
  × <span id="pointerAmount">1</span>
  <button id="btnBuyPointer">購入 (50 Cookie)</button>
</div>
```

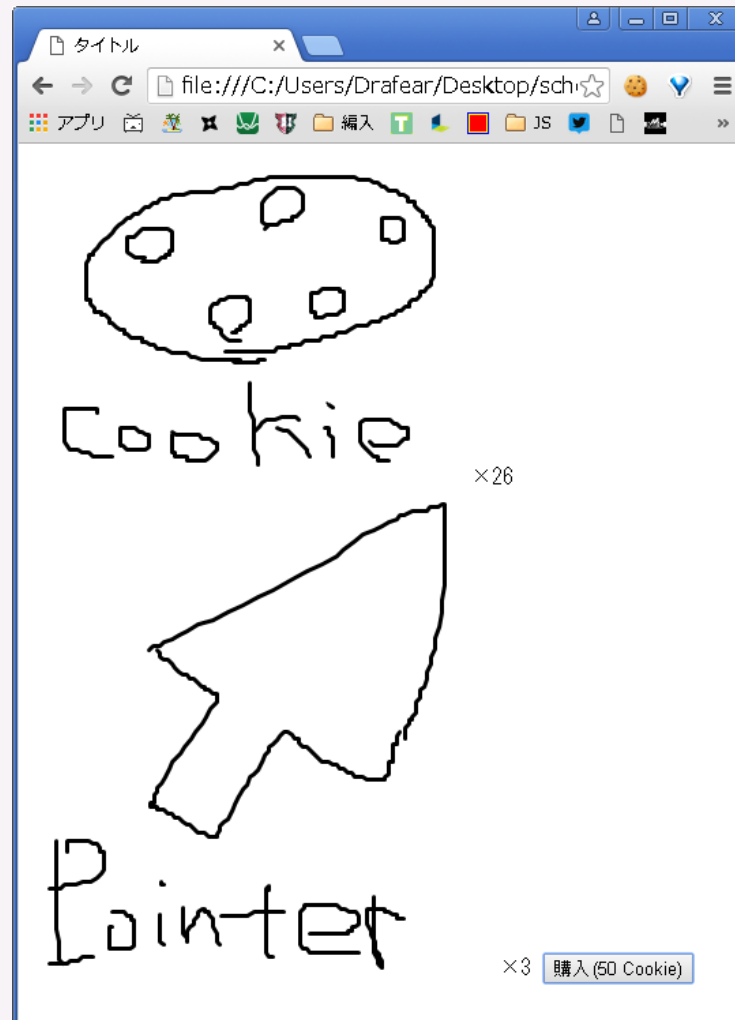
条件分岐

- 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)

```
/* ポインターを1個50クッキーで買う */  
let buy = () => {  
  if (cookieAmount >= 50) { // クッキーを50個以上持っているか  
    cookieAmount -= 50;  
    pointerAmount += 1;  
    update(); // ポインターとクッキーの所持数を画面に反映  
  }  
}  
document.getElementById("btnBuyPointer").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 pointerPrice = 50;
let buy = () => {
  if (cookieAmount >= pointerPrice) { // まとめる
    cookieAmount -= pointerPrice; // まとめる
    pointerAmount += 1;
    update();
  }
}
document.getElementById("btnBuyPointer").innerText =
  `購入 (${pointerPrice} Cookie)`; // まとめる(購入ボタンのテキスト)
document.getElementById("btnBuyPointer").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}
```

今後の予定

- 5/15(日) 13:00 ~ 16:00
 - CSSでデザインする
 - じゃんけんを作る
- 5/22(日) 13:00 ~ 16:00
 - JavaScriptの基礎をマスターする
 - CSSでレイアウトを学ぶ
 - ゴリラを倒すゲームを作る
- 5/29(日) 13:00 ~ 16:00
 - 復習回
 - atomの便利なプラグイン紹介

