

第4回

# JavaScriptから始める プログラミング2016

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

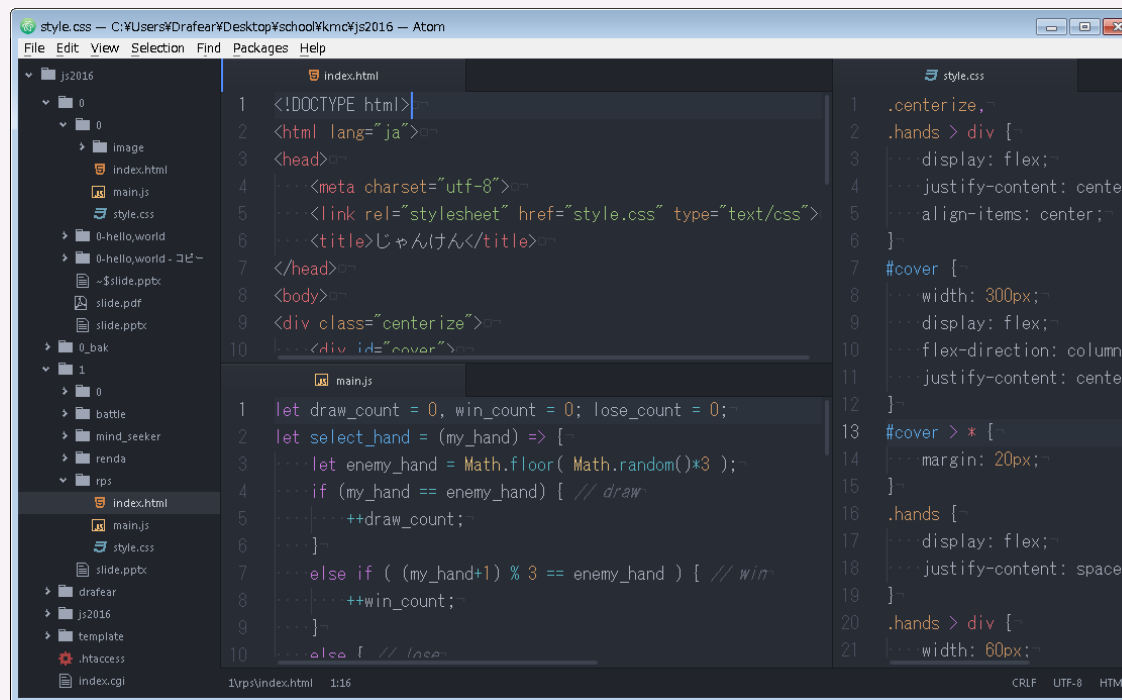
# 自己紹介

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・宣伝

# 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/>



# 今日の目標

- 復習する

# 本日の内容

- atomの便利なプラグイン紹介！！
- 復習
- 何かを作ろう



# 1. atomの便利なプラグイン

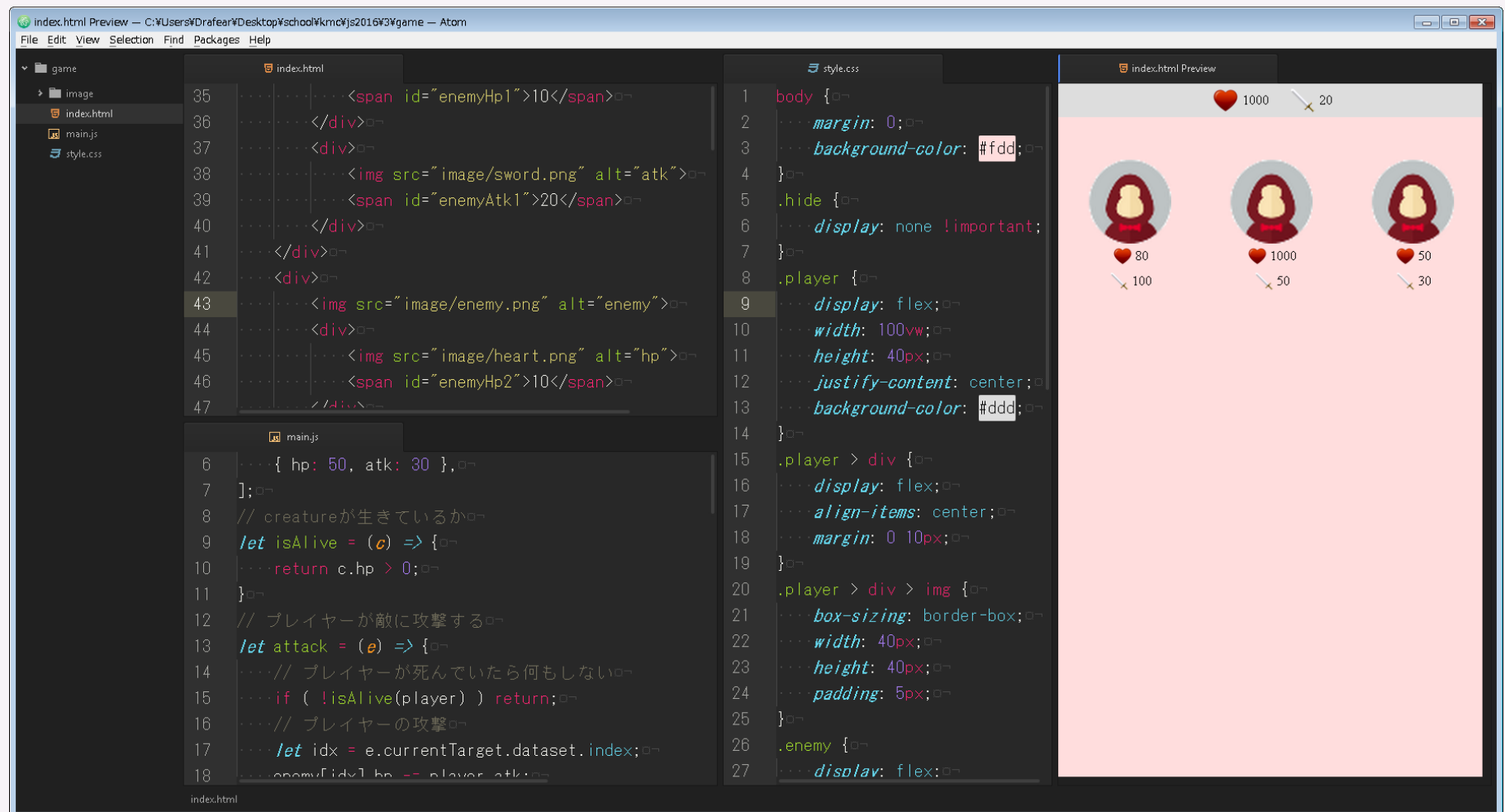
# おことわり

- pptxではなくpdfの場合は  
gifアニメが再生されませんがご了承下さい



# atom-html-preview

- Ctrl + Shift + H (default)

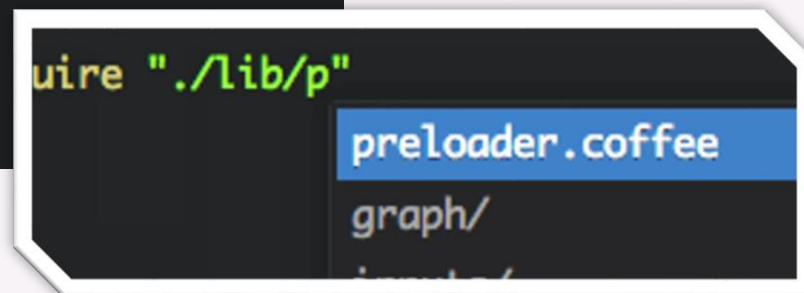


# run-in-browser

- HTMLファイル編集集中に Ctrl + Alt + R を押すとブラウザで開く

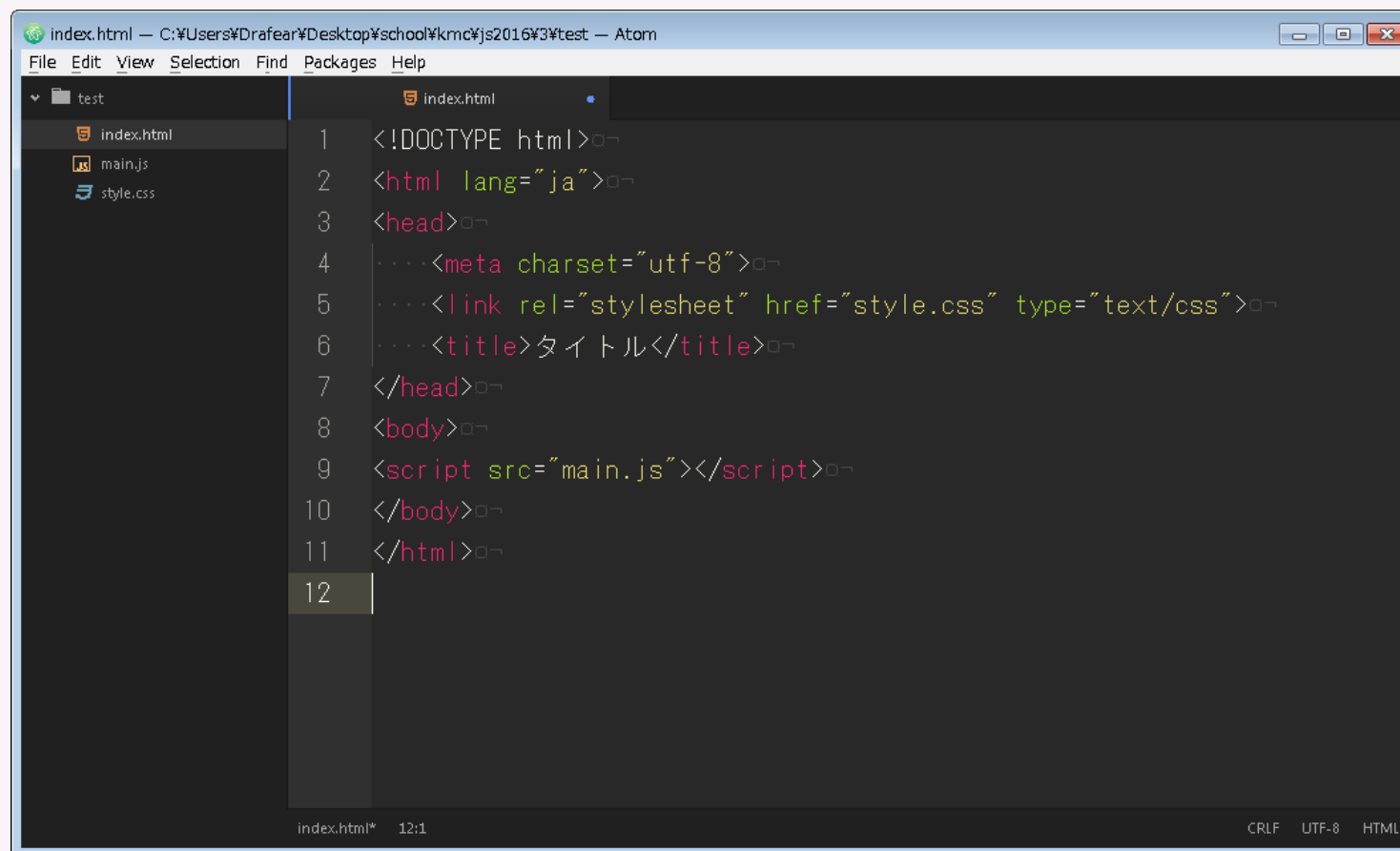
# autocomplete-paths

- ファイルやディレクトリ名を補完！最高！



# file-icons

- ファイル名にアイコンが付く！
- 分かりやすいしやる気に繋がる！！

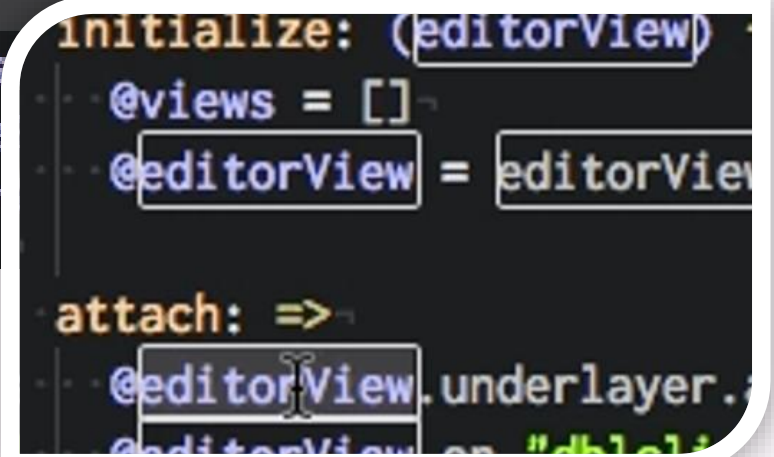


```
index.html — C:\Users\Drafeard\Desktop\school\kmc\js2016\3\test — Atom
File Edit View Selection Find Packages Help
test
  index.html
  main.js
  style.css
1 <!DOCTYPE html>
2 <html lang="ja">
3 <head>
4   <meta charset="utf-8">
5   <link rel="stylesheet" href="style.css" type="text/css">
6   <title>タイトル</title>
7 </head>
8 <body>
9   <script src="main.js"></script>
10 </body>
11 </html>
12
index.html* 12:1 CRLF UTF-8 HTML
```

# highlight-selected

- 同じ単語をハイライト！
- ミスが減る！！

```
9  initialize: (editorView) ->
10    @views = []
11    @editorView = editorView
12
13  attach: =>
14    @editorView.underlayer.append(this)
15    @editorView.on "dblclick", @handle
16    @editorView.on "click", @removeMar
17
```

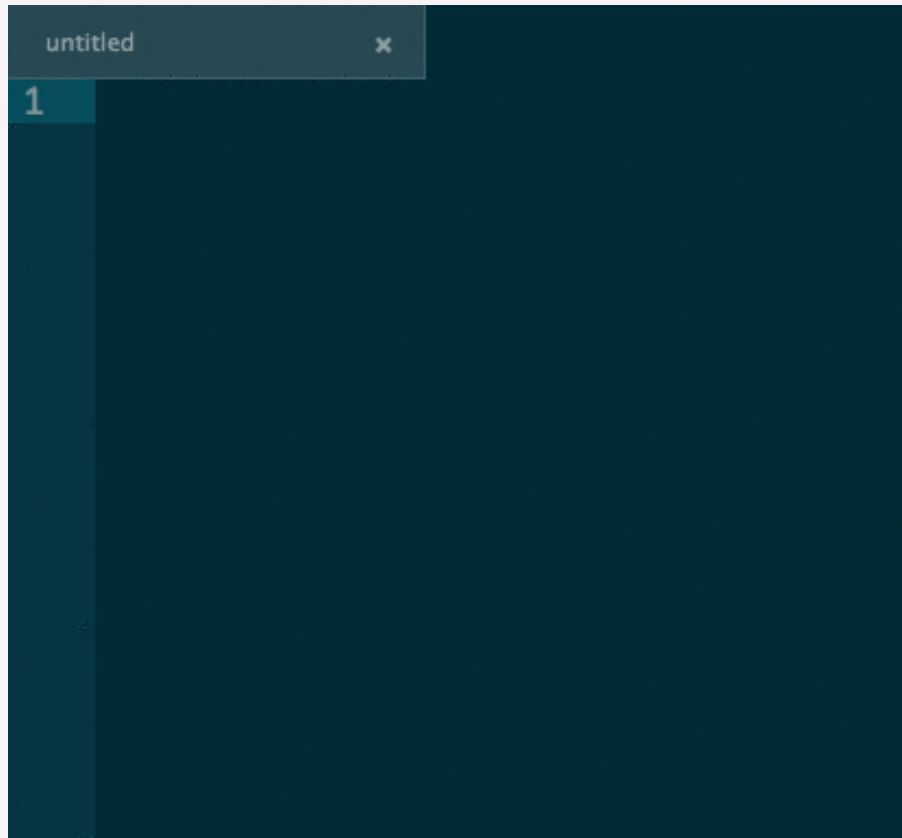


```
initialize: (editorView)
@views = []
@editorView = editorView

attach: =>
  @editorView.underlayer.
```

# less-than-slash

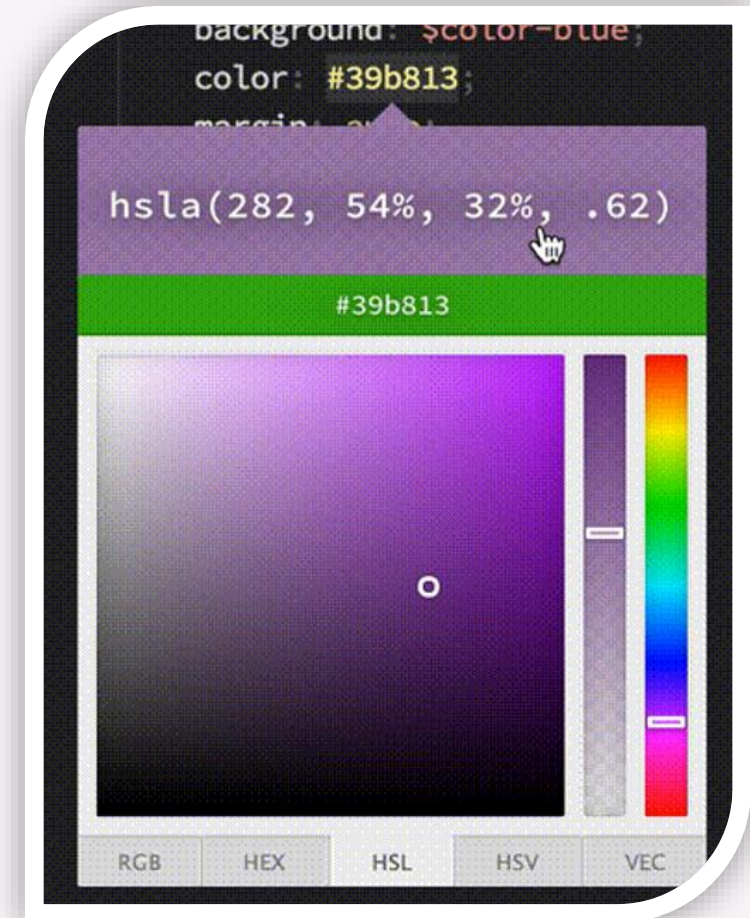
- 最初に入れてもらったプラグイン
- " </ " と入力すると閉じタグが自動的に入る！



# color-picker

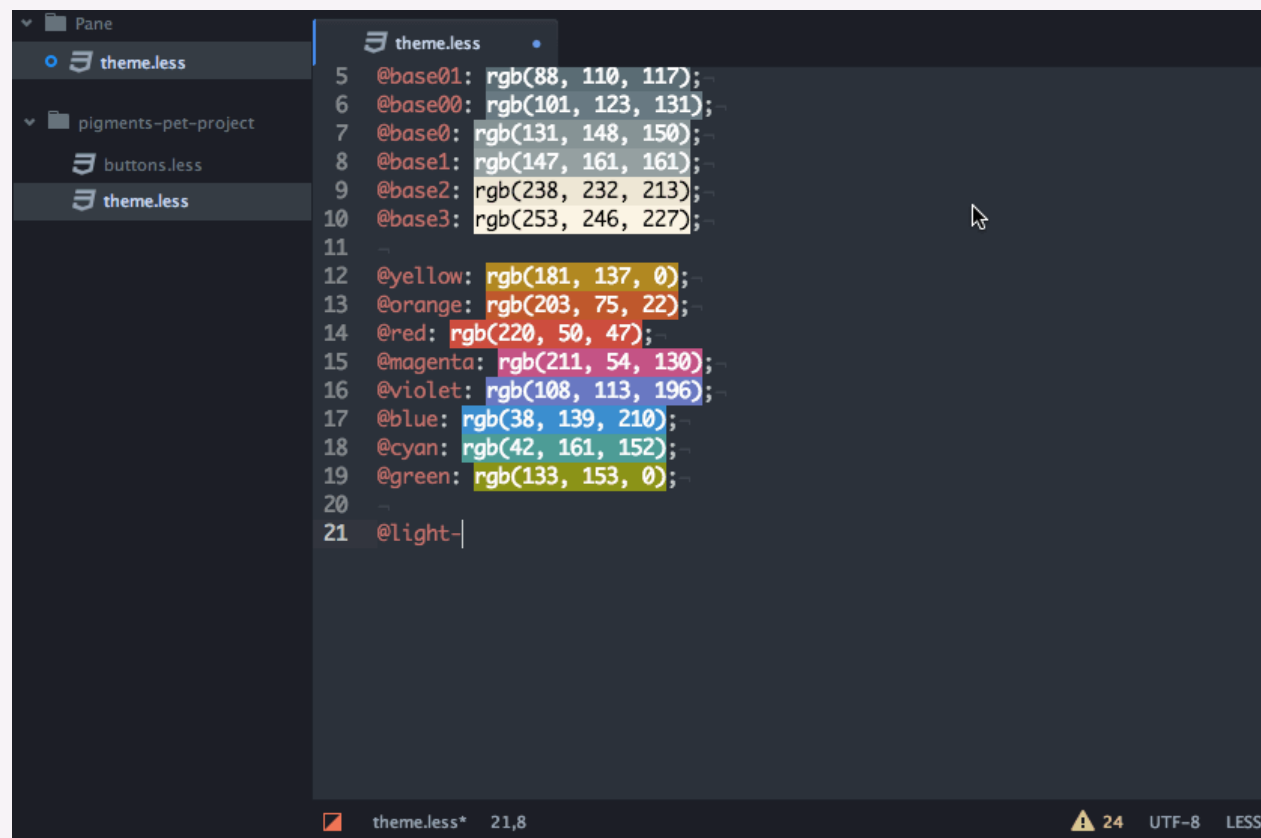
- RGBの値を見て, 正確に色を想像できますか？

```
4  
5 .Content {  
6   background: $color-blue;  
7   color: #39b813;  
8   margin: auto;  
9   position: relative; }  
10
```



# pigments

- rgb(...) に色が付く
- 分かりやすい！



```
5 @base01: rgb(88, 110, 117);
6 @base00: rgb(101, 123, 131);
7 @base0:  rgb(131, 148, 150);
8 @base1:  rgb(147, 161, 161);
9 @base2:  rgb(238, 232, 213);
10 @base3:  rgb(253, 246, 227);
11
12 @yellow:  rgb(181, 137, 0);
13 @orange:  rgb(203, 75, 22);
14 @red:     rgb(220, 50, 47);
15 @magenta: rgb(211, 54, 130);
16 @violet:  rgb(108, 113, 196);
17 @blue:    rgb(38, 139, 210);
18 @cyan:    rgb(42, 161, 152);
19 @green:   rgb(133, 153, 0);
20
21 @light-
```





## 2. 復習

# 全般

- 第1回
  - 拡張子 : hoge.fuga.png
  - 絶対パス : http://google.com/
  - 相対パス : image/hoge.png

# JavaScript

- 第1回

- alert(val)
- console.log(val)
- エラー : コンソールに出力される
- 変数 : `let a = 10;`
- 代入 : `a = 5;`
- 四則演算 : `(a * 5 + 8 / 2 - 1) % 10`
- 糖衣構文 : `a = a + 5 ⇔ a += 5`
- コメント : `// 一行コメント, /* 複数行コメント */`
- 関数 : `(param1, param2) => { ... }`
- 文字列 : `"to" + "kyoto"`
- オブジェクト : `{ a: 10, b: 5 }.a`
- テンプレート文字列 : ``${a} + ${b} = ${a + b}``

# JavaScript

- 第2回 (+ $\alpha$ )
  - `Math.max(a, b)`, `Math.min(a, b)` : aとbの小さい方/大きい方
  - `Math.floor(x)`, `Math.ceil(x)`, `Math.round(x)` : 切り捨てなど
  - `Math.pow(x, n)` :  $x^n$
  - `Math.sqrt(x)` :  $\sqrt{x}$
  - `Math.sin(x)`, `Math.cos(x)`, `Math.tan(x)`
  - `Math.asin(x)`, `Math.acos(x)`, `Math.atan(x)`  
:  $\sin^{-1} x \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$   $\cos^{-1} x \in [0, \pi]$   $\tan^{-1} x \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$
  - `Math.atan2(y, x)` :  $\tan^{-1} \frac{y}{x} \in [-\pi, \pi]$
  - `Math.abs(x)` :  $|x|$
  - `Math.PI` :  $\pi$
  - `Math.random()` :  $[0, 1)$  の乱数

# JavaScript

- 第2回

- true, false
- $a > b$ ,  $a < b$
- $a \geq b$ ,  $a \leq b$
- $a === b$ ,  $a !== b$
- $!a$  : NOT a
- $a \&\& b$  : a AND b
- $a || b$  : a OR b
- `if (条件1) { 処理1 } else if (条件2) { 処理2 } else { 処理3 }`
- `return` : そこで中断

# JavaScript

- 第3回

- 変数のスコープ : 変数の見える範囲
- 再帰 : 関数の中で自分自身を呼ぶ
- インクリメント : `++i, i++`
- デクリメント : `--i, i--`
- 無限ループ : `while (1) { 処理 }, for (;;) { 処理 }`
- while文 : `while (継続条件) { 処理 }`
- for文 : `for (let i = 0; i < n; ++i) { 処理 }`
- continue : 次の繰り返しに移る
- break : 繰り返し文を抜ける
- 配列 : `a = [114, 514, 1919, 810]; b = a[3]; // 1919`
- 配列とfor文 : `for (let i = 0; i < a.length; ++i) { 処理 }`

# HTML

- 第1回

- `<h1> ~ </h1>` : 見出し
- 開始タグ : `<tagName>`
- 終了タグ(閉じタグ) : `</tagName>`
- HTML要素 : 開始タグ~終了タグ で1つの要素
- `<タグ名 属性1="属性値1" 属性2="属性値2">`
- `<img>` : src属性で指定した画像を表示  
閉じタグ不要, alt属性(代替テキスト)必須
- `<div>, <span>` : 汎用コンテナ

# HTML

- 第3回
  - `<p>` : 段落
  - `<ul>` : 順序なしリスト
  - `<li>` : ulの要素



# CSS

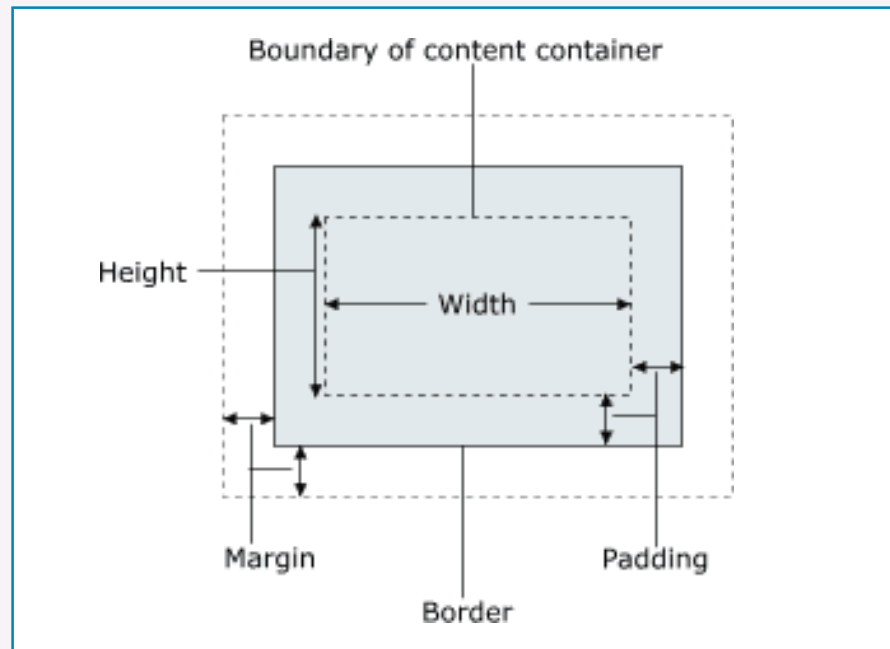
- 第2回

- class属性 : 複数クラス指定可
- `.className { property1: parameter1; ... }`
- 文字色 : `color: red;`
- 文字サイズ : `font-size: 40px;`
- 太字 : `font-weight: bold;`
- 下線 : `text-decoration: underline;`
- 要素非表示 : `display: none;`
- カーソルの形 : `cursor: pointer;`
- マウスオーバー : `.className:hover { ... }`
- notセレクタ : `.className1:not(.className2) { ... }`

# CSS

- 第2回

- ボックス : width, height, margin, padding, border
- 背景色 : background-color: rgb(114, 51, 4);
- 背景画像 : background-image: url(*imagePath*);

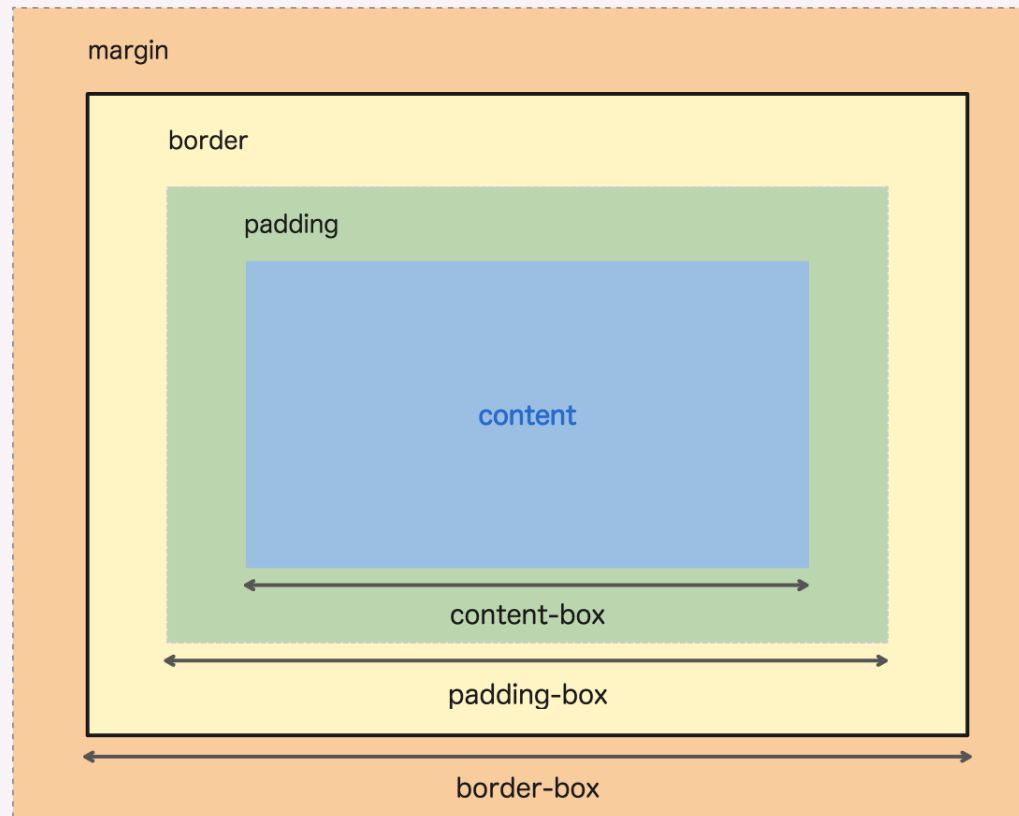


<https://www.addedbytes.com/articles/for-beginners/the-box-model-for-beginners/>

# CSS

- 第2回

- width等の計算方法 : `box-sizing: border-box;`



# CSS

- 第2回

- 色の指定 : red, #abcdef, #fff, rgb(255,255,255)
- 半透明な色 : rgba(255,255,255,1.0)
- 縁取り非表示 : outline: none;
- アニメーション : transition: all .3s;

# CSS

- 第3回

- セレクタ

*#id*

*.className*

*tagName*

*selector1 selector2* : 子孫

*selector1 > selector2* : 子

*selector1 + selector2* : 直後の兄弟

*selector1 ~ selector2* : それ以降の兄弟

*selector.className* : *selector* かつ *className*

*selector#id* : *selector* かつ *id*

# CSS

- 第3回

- ブロックボックス : □, 直後に改行  
`display: block;`
- インラインボックス : 囲むだけ  
`display: inline;`



<https://nulab-inc.com/ja/blog/nulab/css-basics-for-engineer-boxmodel/>



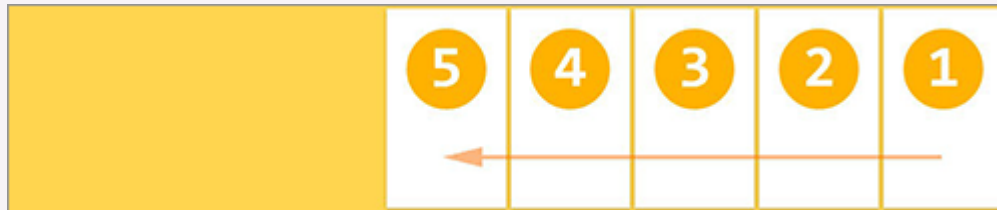
<https://nulab-inc.com/ja/blog/nulab/css-basics-for-engineer-boxmodel/>

# CSS

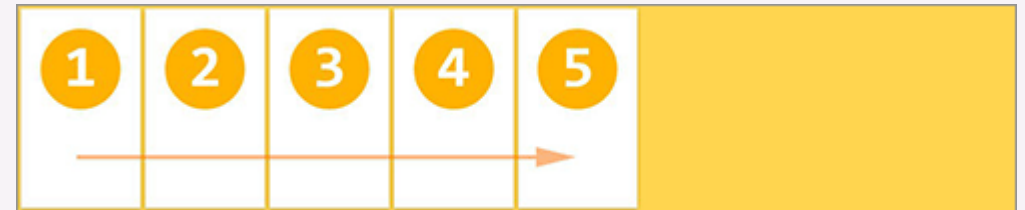
- 第3回
  - flexbox : 便利  
display: flex;

# flexbox

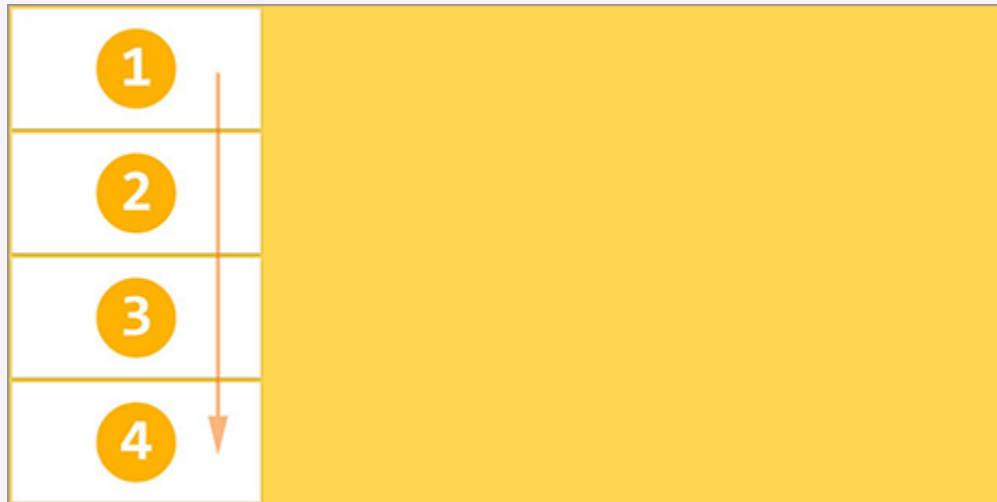
- flex-direction 向き



row(default)



row-reverse



column



column-reverse



# flexbox

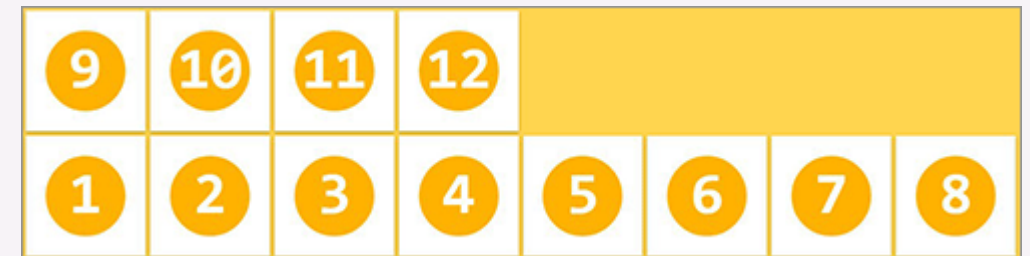
- flex-wrap 折り返し



nowrap(default)



wrap



wrap-reverse

# flexbox

- justify-content 主軸方向の余白の使い方



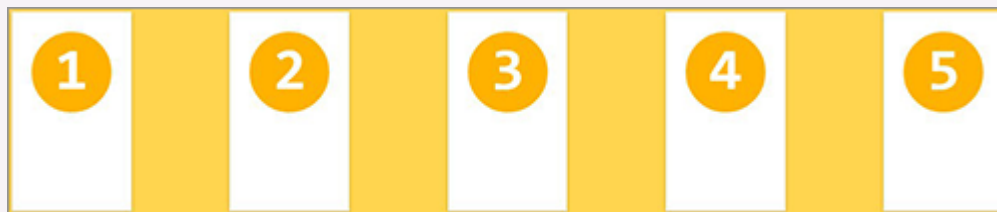
flex-start(default)



flex-end



center



space-between



space-around

# flexbox

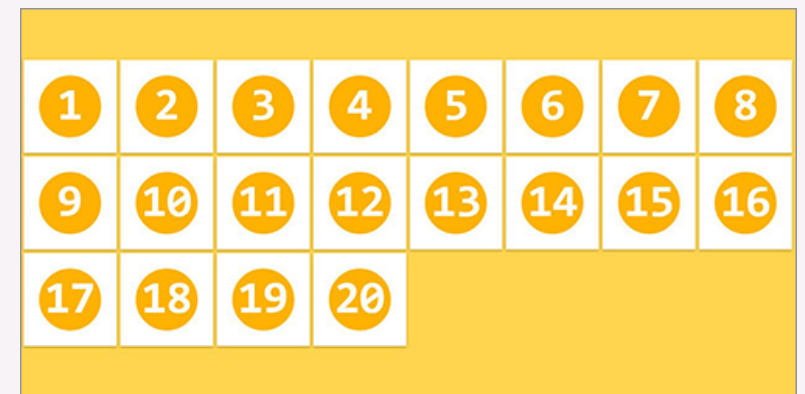
- align-content クロス軸方向全体としての余白の使い方



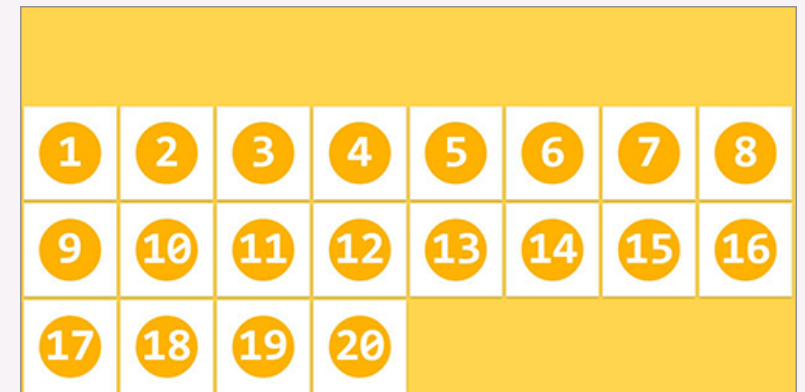
stretch(default)



flex-start



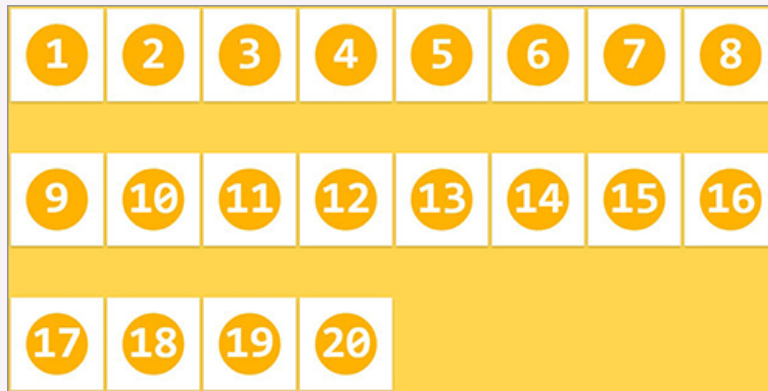
center



flex-end

# flexbox

- align-content クロス軸方向全体としての余白の使い方



space-between



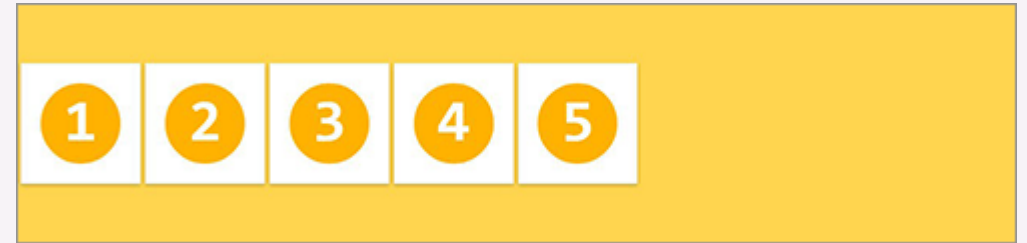
space-around

# flexbox

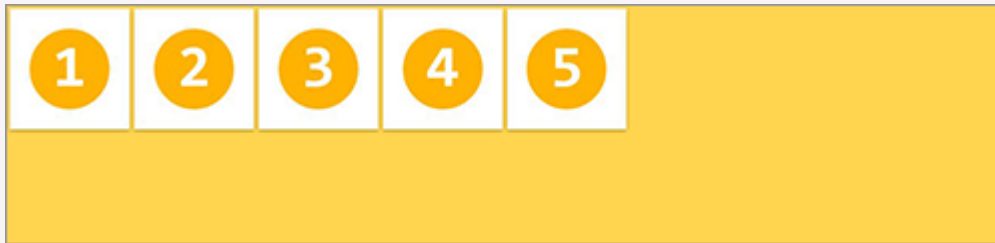
- align-items 各行でのクロス軸方向の余白の使い方



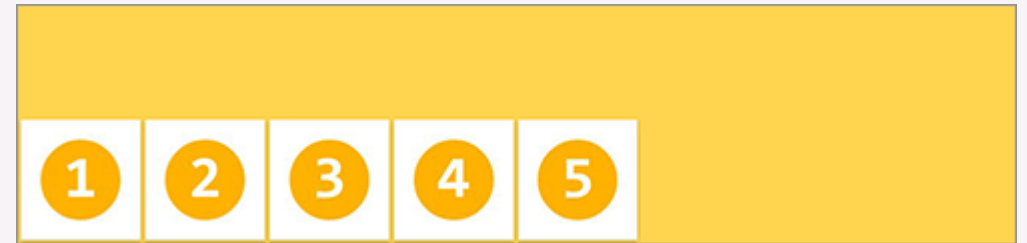
stretch(default)



center



flex-start



flex-end



baseline

# CSS

- 第3回

- body : デフォルトでmarginを持つ
- ブラウザ表示領域に対する単位
  - width: 100vw;
  - height: 100vh;
- marginの指定方法
  - margin: [上下左右];
  - margin: [上下] [左右];
  - margin: [上] [右] [下] [左];
  - padding等も同様

# DOM

- `document.getElementById(id)`
  - *id*属性が*id*の要素を得る
- `document.getElementsByTagName(tagName)`
  - タグ名が*tagName*の要素を配列で得る
- `document.getElementsByClassName(className)`
  - *className*クラスが設定されている要素を配列で得る
- `elem.querySelector(selector)`
  - *selector*にマッチする最初の要素を得る
- `elem.querySelectorAll(selector)`
  - *selector*にマッチする全ての要素を配列で得る

# DOM

- `elem.addEventListener(eventName, function)`
  - `elem`で`eventName`イベントが発生したら`function`を実行する
  - イベント監視
  - イベント例) "click", "mousedown", "mouseup", "keydown", "keyup", "keypress", "mousemove", "mouseover", "mouseout", "dragstart", "dblclick"
- `elem.textContent = "text";`
  - 要素の内容を `text` に変更する



# DOM

- `elem.classList.add(className)`
  - `elem` に `className` クラスを追加する
- `elem.classList.remove(className)`
  - `elem` の `className` クラスを削除する
- `elem.classList.toggle(className)`
  - `elem` の `className` クラスを反転(なければ追加, あれば削除)する
- `elem.classList.contains(className)`
  - `elem` が `className` クラスを持っているか(true/false)
- `elem.className = "";`
  - `elem` のクラスをクリアする

# DOM

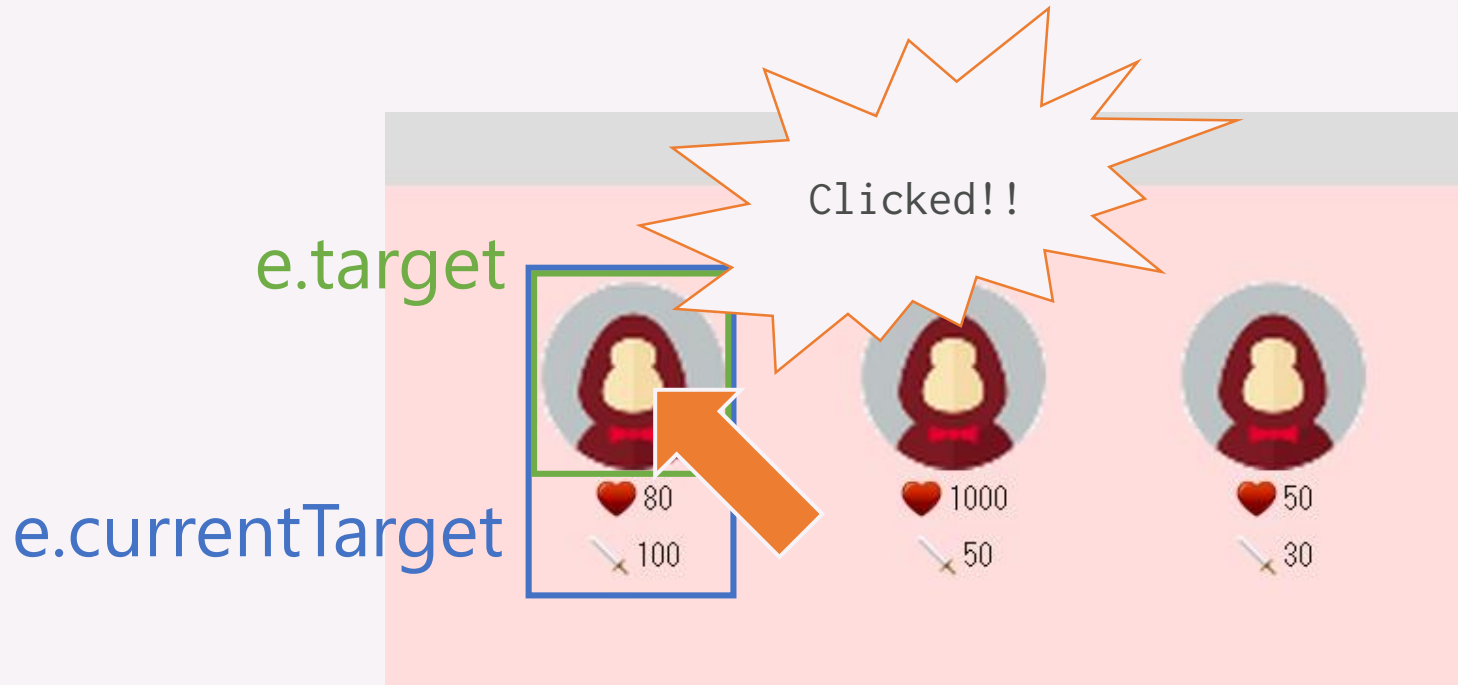
- `elem.children`
  - 子要素(配列)
- `elem.parentElement`
  - 親要素

# DOM

- `elem.dataset.dataName = value`
  - 要素にデータを設定できる(文字列のみ)
- `elem.dataset.dataName`
  - 要素のデータを参照できる

# DOM

- `e.currentTarget`
  - そのイベントが設定された要素
- `e.target`
  - そのイベントが起きた一番深い要素





# 3. デバッグのいろは

# デバッグとは？

- プログラムのエラーやバグを特定して修正すること
- 複雑なバグは原因の特定が難しいので  
上手くデバッグする必要がある
- そのデバッグ手法についてここでは学ぶ
- まずはデバッグのための道具を紹介する

# console.log

- おなじみ

```
console.log("hoge");
```

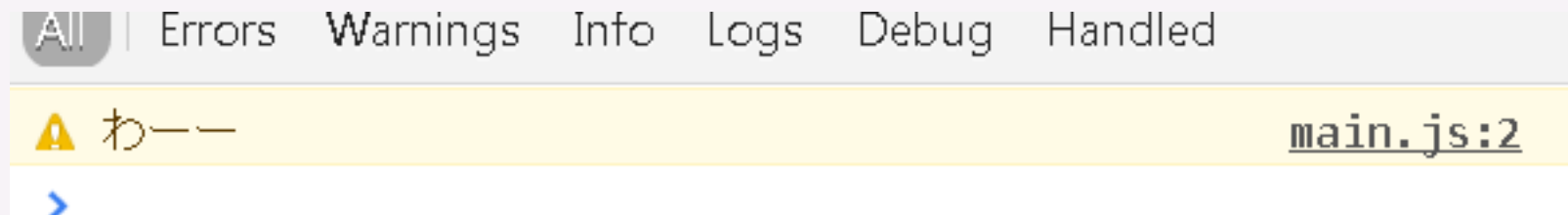
```
hoge
```

```
main.js:1
```

# console.warn

- 警告・注意 っぽく表示

```
console.warn("わー");
```

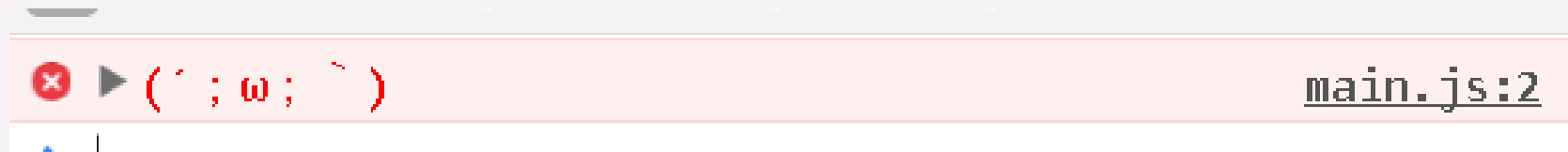




# console.error

- エラー っぽく 表示
  - プログラムの実行は中断されない

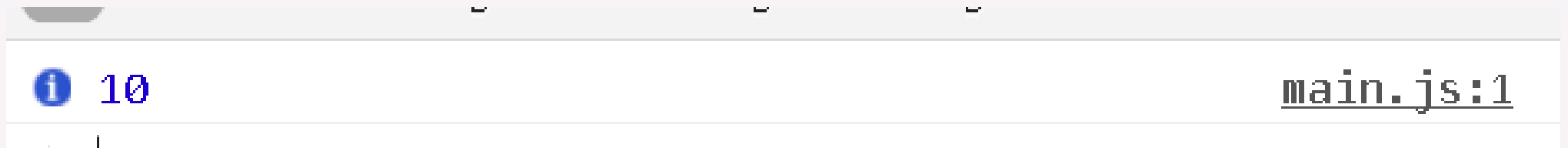
```
console.error("( ' ; ω ; ` )");
```



# console.info

- 情報 っぽく表示

```
console.info(10);
```



# console.clear

- コンソールをクリアする

# console.assert

- console.assert(*exp*)
  - *exp* が true であるはずだ！ と主張する
  - 万が一 false だった場合, エラーが出力される
  - エラーが出てもプログラムの実行は中断はされない

```
let dx = x2 - x1;  
console.assert(dx >= 0);
```



A screenshot of a web browser's developer console. The top bar is red, indicating an error. On the left, there is a red circle with a white 'x' icon, followed by a right-pointing triangle. The text 'Assertion failed:' is displayed in red. On the right side of the bar, the text 'main.js:4' is shown in black. Below the red bar, a white area contains a blue cursor icon and a vertical line.

Assertion failed: main.js:4

# console.count

- console.count()
  - その行が実行された回数を出力する

```
console.count(); // 1
for (let i = 0; i < 10; ++i) {
  console.count(); // 1~10
  // do something
}
console.count(); // 1
```

: 1	main.js:1
: 1	main.js:3
: 2	main.js:3
: 3	main.js:3
: 4	main.js:3
: 5	main.js:3
: 6	main.js:3
: 7	main.js:3
: 8	main.js:3
: 9	main.js:3
: 10	main.js:3
: 1	main.js:6

# console.count

- console.count(label)
  - 別の行でも label が同じであれば一緒にカウントする

```
console.count("outer"); // 1
for (let i = 0; i < 10; ++i) {
  console.count("inner"); // 1~10
  // do something
}
console.count("outer"); // 2
```

outer: 1	main.js:1
inner: 1	main.js:3
inner: 2	main.js:3
inner: 3	main.js:3
inner: 4	main.js:3
inner: 5	main.js:3
inner: 6	main.js:3
inner: 7	main.js:3
inner: 8	main.js:3
inner: 9	main.js:3
inner: 10	main.js:3
outer: 2	main.js:6

# console.count

- console.count(label)
  - 別の行でも label が同じであれば一緒にカウントする

```
console.count("outer"); // 1
for (let i = 0; i < 10; ++i) {
  console.count("inner"); // 1~10
  // do something
}
console.count("outer"); // 2
```

outer: 1	main.js:1
inner: 1	main.js:3
inner: 2	main.js:3
inner: 3	main.js:3
inner: 4	main.js:3
inner: 5	main.js:3
inner: 6	main.js:3
inner: 7	main.js:3
inner: 8	main.js:3
inner: 9	main.js:3
inner: 10	main.js:3
outer: 2	main.js:6

# console.dirxml

- HTML要素の構造を見るときに便利

```
> console.dirxml( document.body )
```

▼ <body> VM733:1

```
  <input type="text" id="left" value="0">
  "
  +
  "
  <input type="text" id="right" value="0">
  <button id="btnCalc"> = </button>
  <span id="result">0</span>
  <script src="main.js"></script>
</body>
```



# console.time ~ console.timeEnd

- 時間計測できる

```
console.time("timer1");  
for (let i = 0; i < 10000; ++i);  
console.timeEnd("timer1");
```

```
timer1: 0.743ms
```

```
main.js:3
```

# console.group ～ console.groupEnd

- グループ化される

```
console.info("情報");  
console.group();  
console.warn("警告");  
console.group();  
console.log("ろぐ");  
console.groupEnd();  
console.error("えらー");  
console.groupEnd();
```

情報	main.js:1
▼	main.js:2
警告	main.js:3
▼	main.js:4
ろぐ	main.js:5
えらー	main.js:7

# console.trace

- 呼びだされたのがわかる

```
const fact = (n) => {  
  console.trace();  
  if (n === 0) return 1;  
  return n * fact(n-1);  
}  
fact(3);
```

```
▼ console.trace() main.js:2  
  fact @ main.js:2  
  (anonymous function) @ main.js:6  
  
▼ console.trace() main.js:2  
  fact @ main.js:2  
  fact @ main.js:4  
  (anonymous function) @ main.js:6  
  
▼ console.trace() main.js:2  
  fact @ main.js:2  
  fact @ main.js:4  
  fact @ main.js:4  
  (anonymous function) @ main.js:6  
  
▼ console.trace() main.js:2  
  fact @ main.js:2  
  fact @ main.js:4  
  fact @ main.js:4  
  fact @ main.js:4  
  (anonymous function) @ main.js:6
```

# まとめ

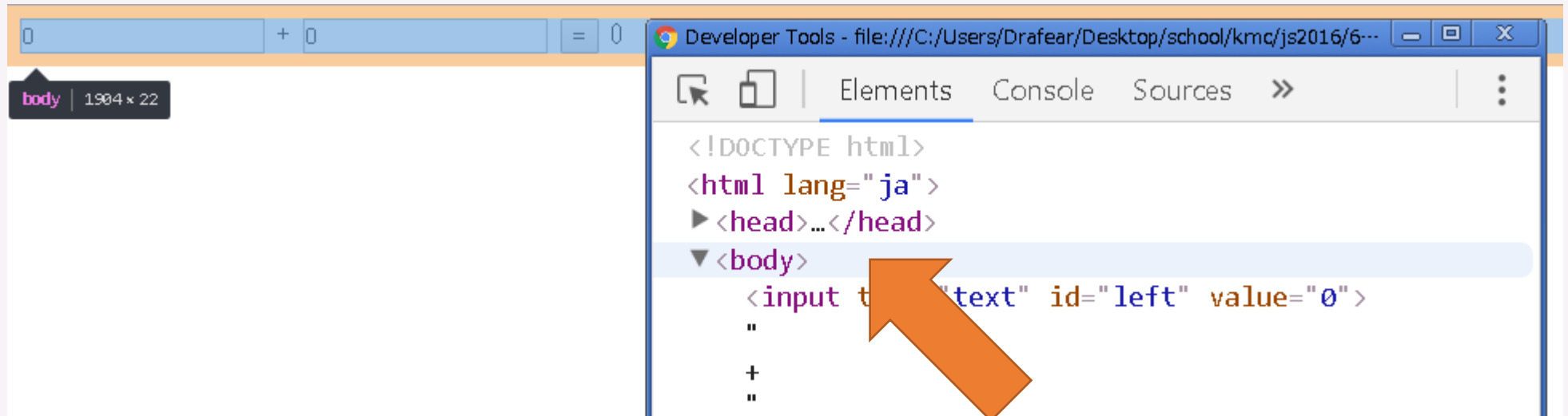
- `console.log`, `console.assert` で十分
- 必要に応じて見やすくしよう

# ここからが本題

- デバッグのいろはの本題です

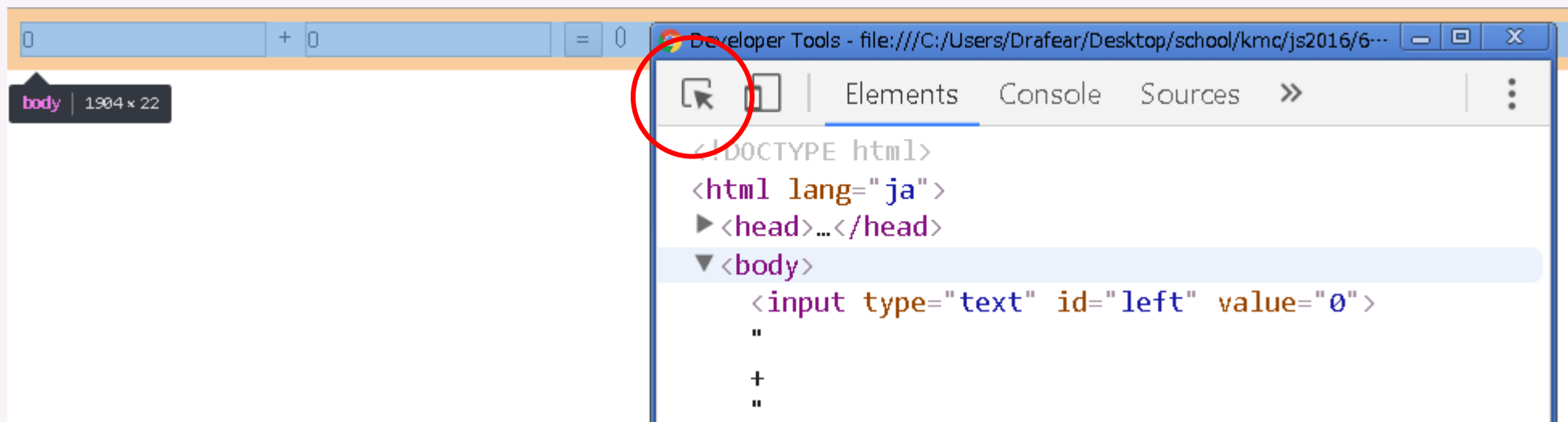
# HTML

- Elements
  - マウスカースールを合わせると, 対応する要素がハイライトされる



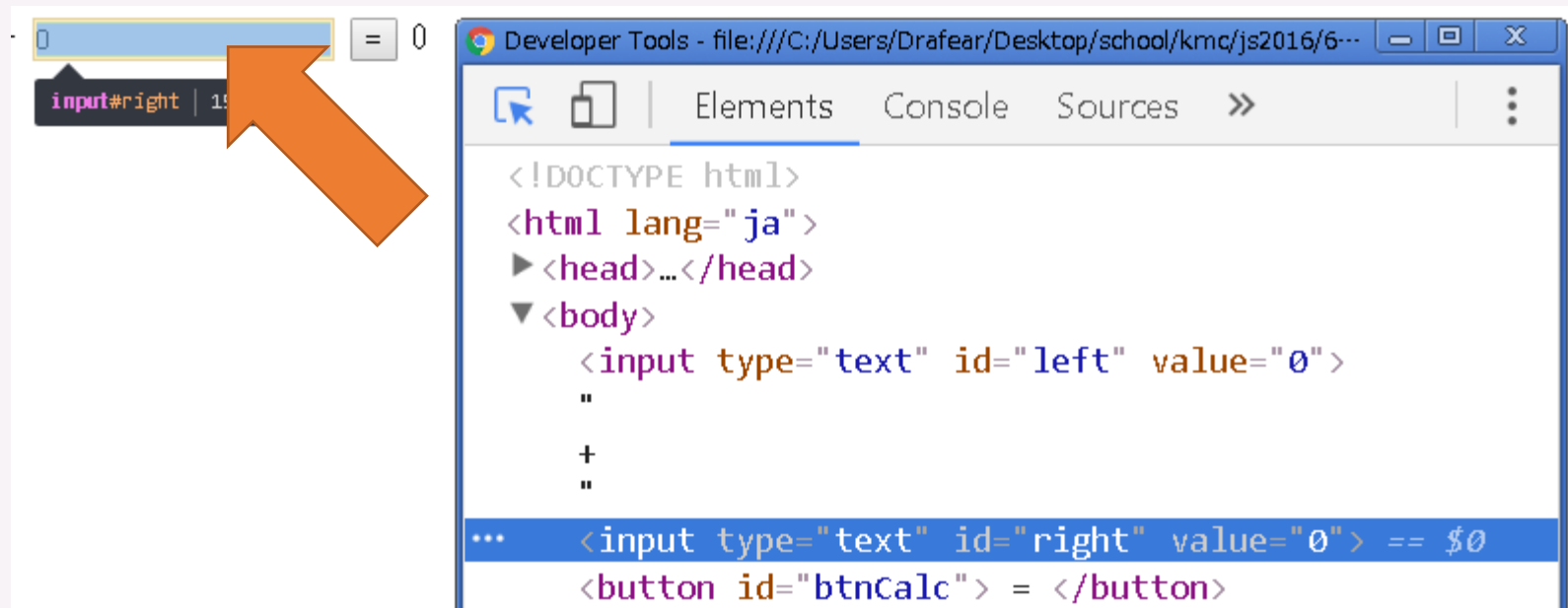
# HTML

- これを押すと...



# HTML

- 要素にカーソルを合わせるだけで情報が得られる！





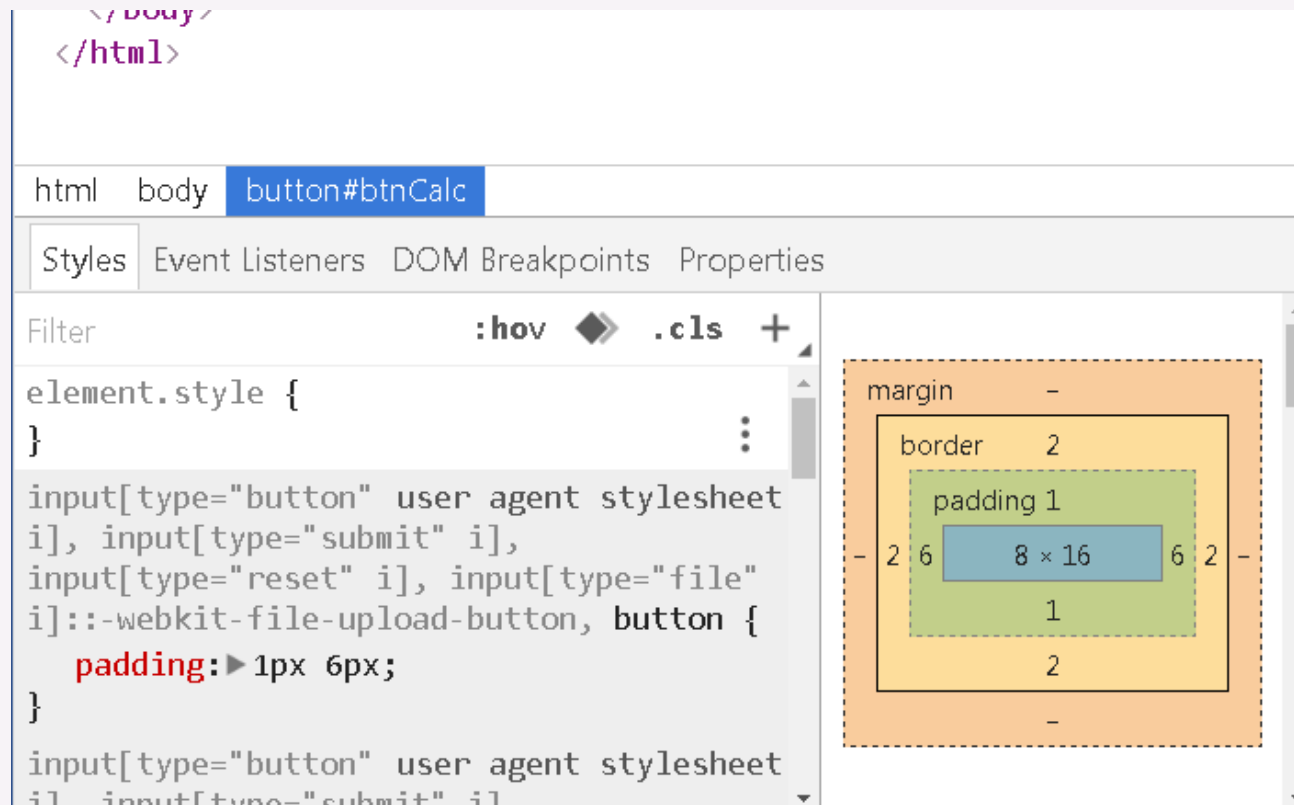
# HTML

- 右クリックから要素の検証でもおk

The image illustrates how to inspect an HTML element using the right-click context menu. On the left, a right-click context menu is shown over a text input field containing the value '0'. The menu includes standard editing actions like '取消(U)' (Undo), 'やり直し(R)' (Redo), '切り取り(T)' (Cut), 'コピー(C)' (Copy), '貼り付け(P)' (Paste), and 'すべて選択(A)' (Select All). At the bottom of the menu, the '検証(I)' (Inspect) option is highlighted, with the keyboard shortcut 'Ctrl+Shift+I' shown next to it. An orange arrow points from this option to the right, where the Chrome Developer Tools interface is shown. In the Developer Tools, the 'Elements' panel is active, displaying the HTML structure of the page. The selected element is an input field with the attributes `<input type="text" id="right" value="0">`. The 'Styles' panel on the right shows the default user agent styles for this input field, including margin, border, and padding. The 'Properties' panel on the far right shows the element's attributes, including `id="right"` and `value="0"`.

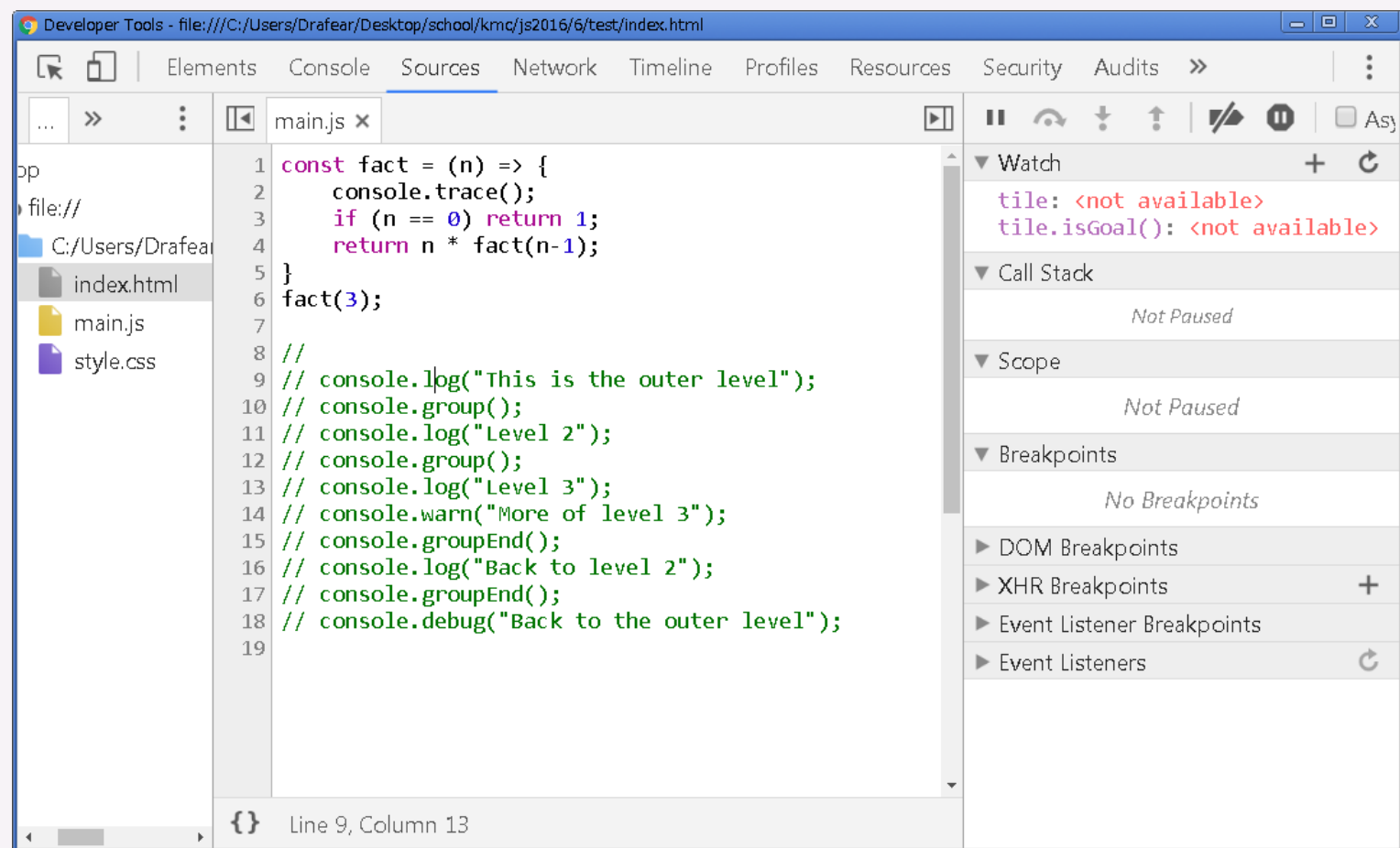
# HTML

- どのくらい情報が得られるのか
  - 現在設定されている属性やCSS, ボックス(margin, padding等)
  - レイアウト時や動的に属性を追加したりするときに便利！



# JavaScript

- Sources - JavaScriptのデバッグに便利！



# JavaScript

- このコードをデバッグしていく

```
const fact = (n) => {  
  if (n = 1) return 1;  
  return n * fact(n-1);  
};  
console.log( fact(3) );
```

# JavaScript

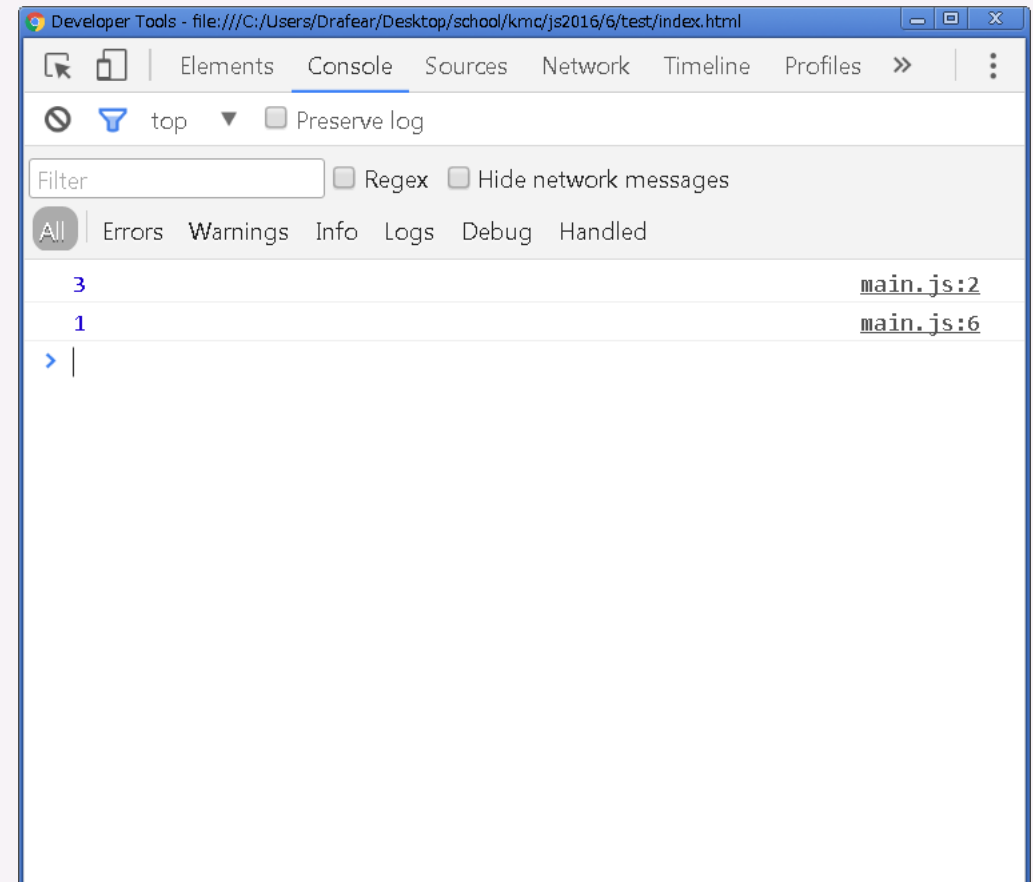
- fact(3) を実行すると 1 が返ってくる. なんだろう.
- その理由を考えて原因がわかったならそれでok.
- ちょっとでも時間がかかりそうなら,  
別の手法でデバッグしていこう.

```
const fact = (n) => {  
  if (n = 1) return 1;  
  return n * fact(n-1);  
};  
console.log( fact(3) );
```

# JavaScript

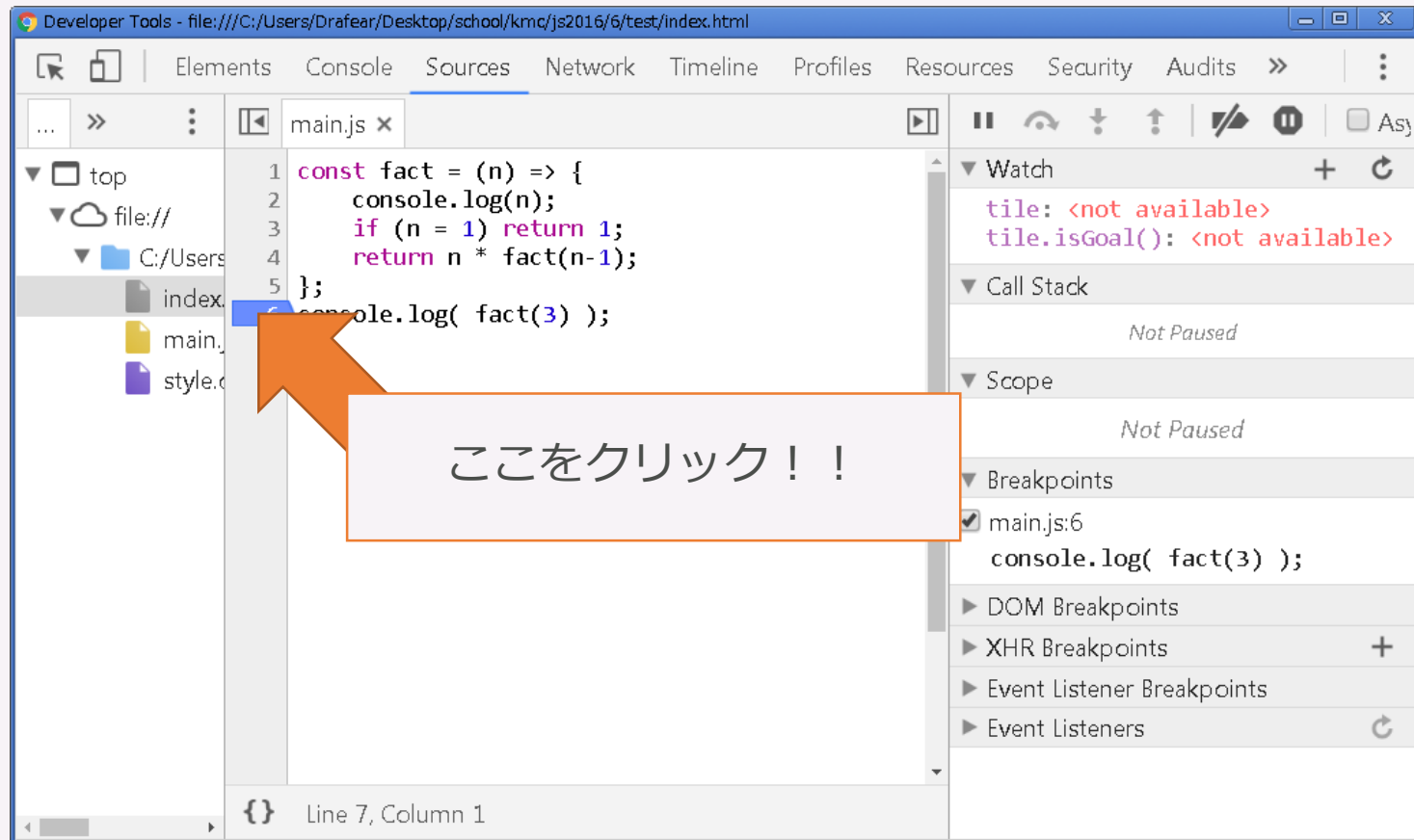
1. console.log を使う
  - 2行目の console.log が1回しか実行されていない

```
const fact = (n) => {  
  console.log(n);  
  if (n = 1) return 1;  
  return n * fact(n-1);  
};  
console.log( fact(3) );
```



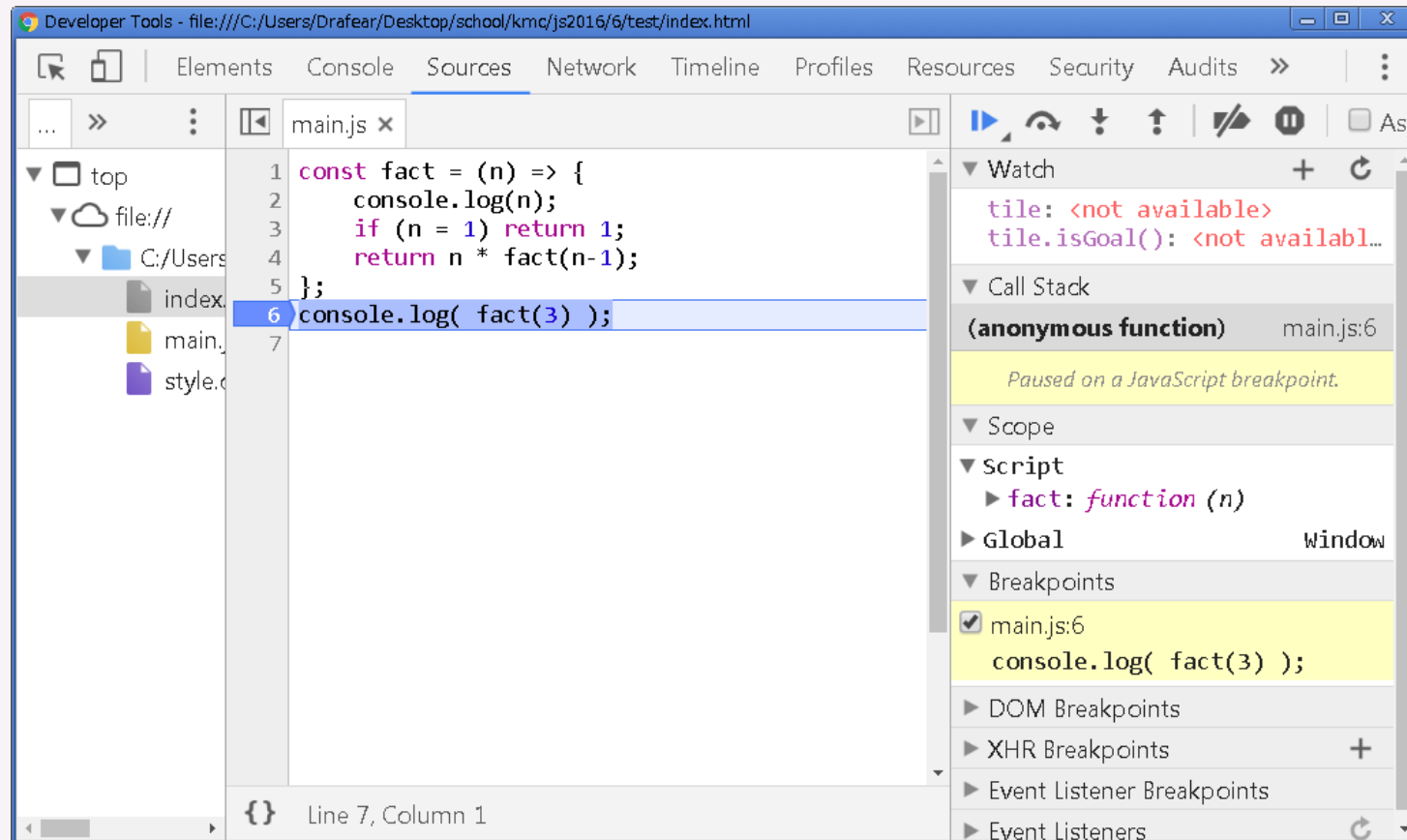
# JavaScript

## 2. break point を設定してステップ実行する



# JavaScript

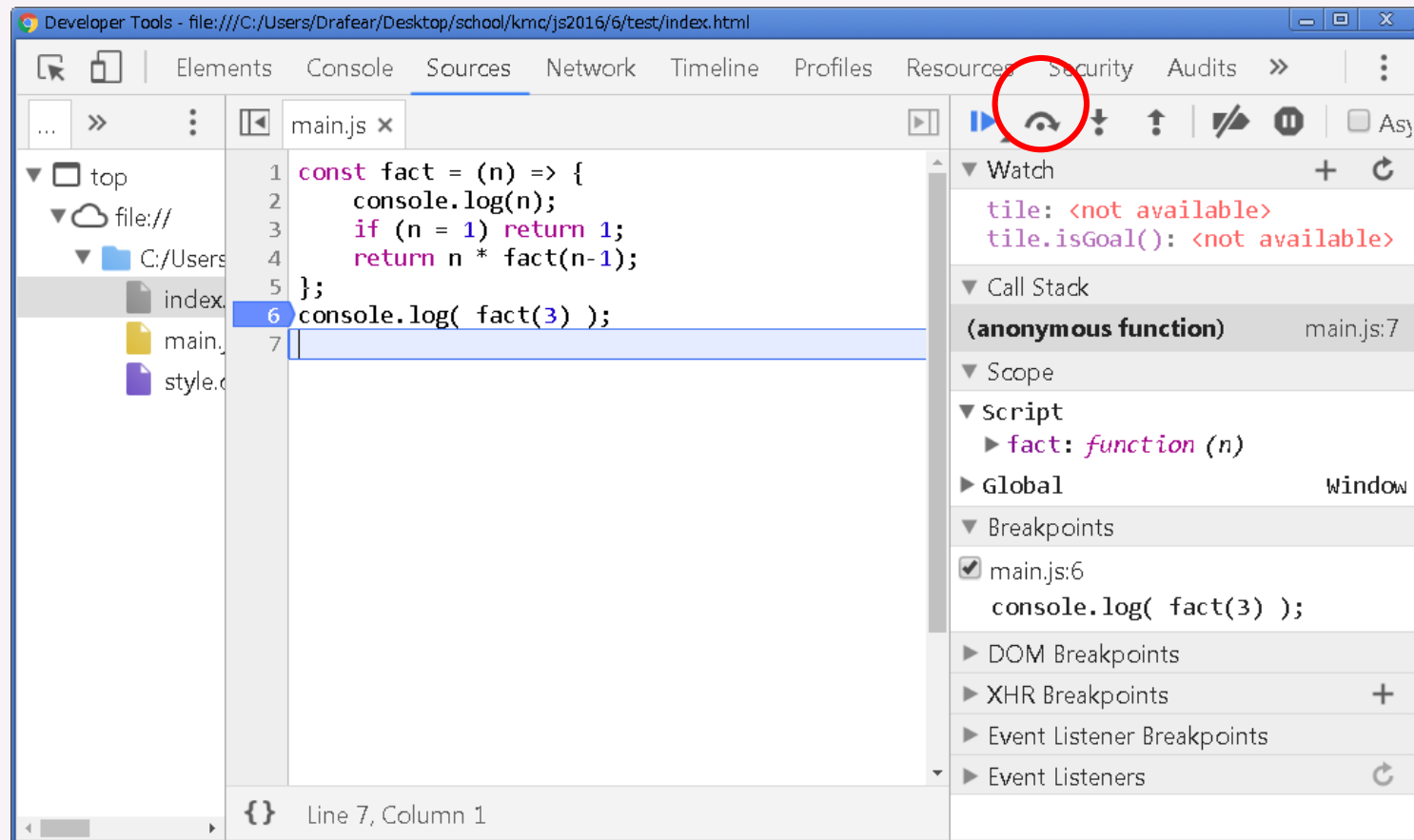
2. Ctrl + R または F5 で再読み込みしよう
  - プログラムの動作が6行目で止まるはず





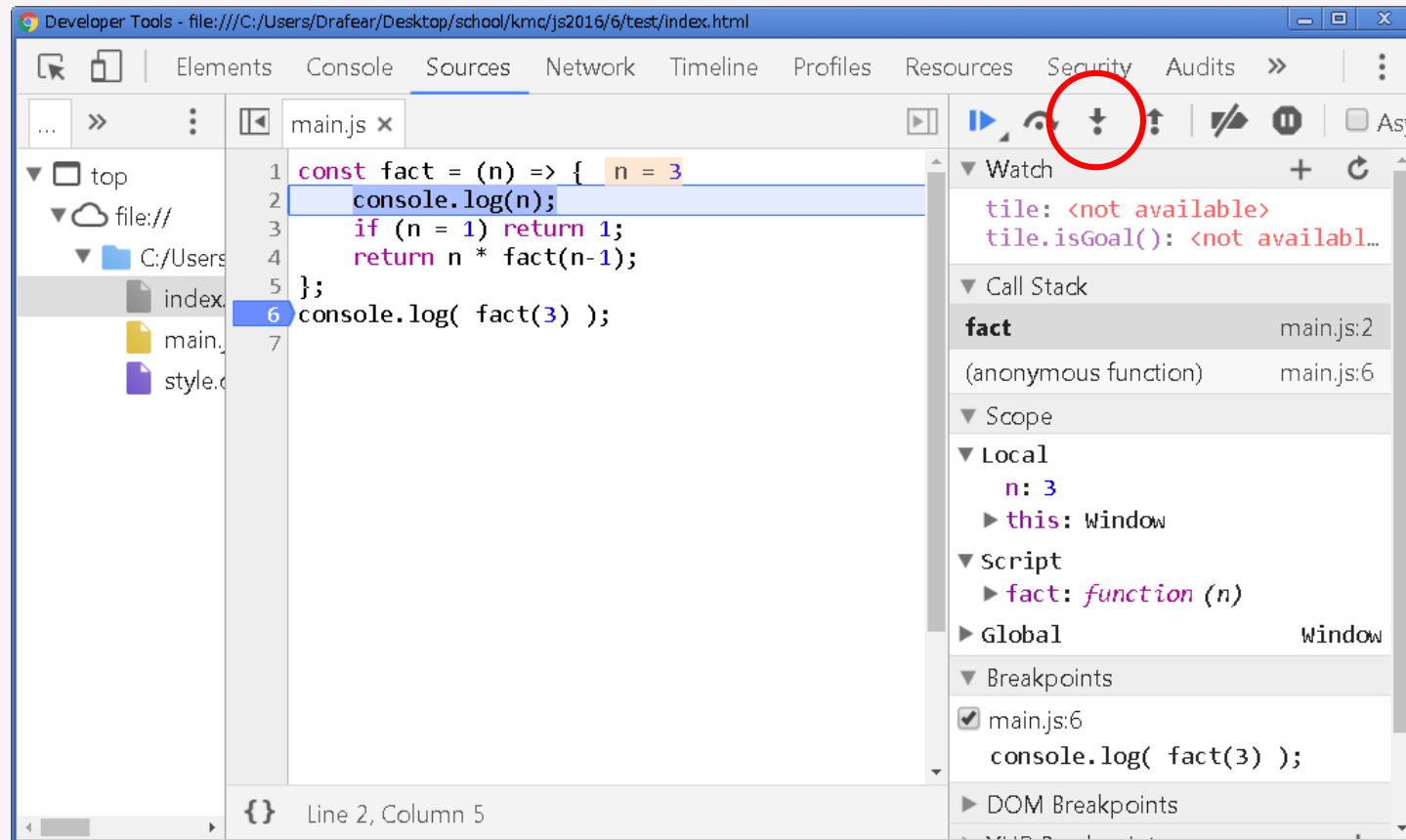
# JavaScript

下の赤丸のボタンか, F10 で 1行ずつ実行できる (ステップオーバー)  
... fact終わっちゃったw



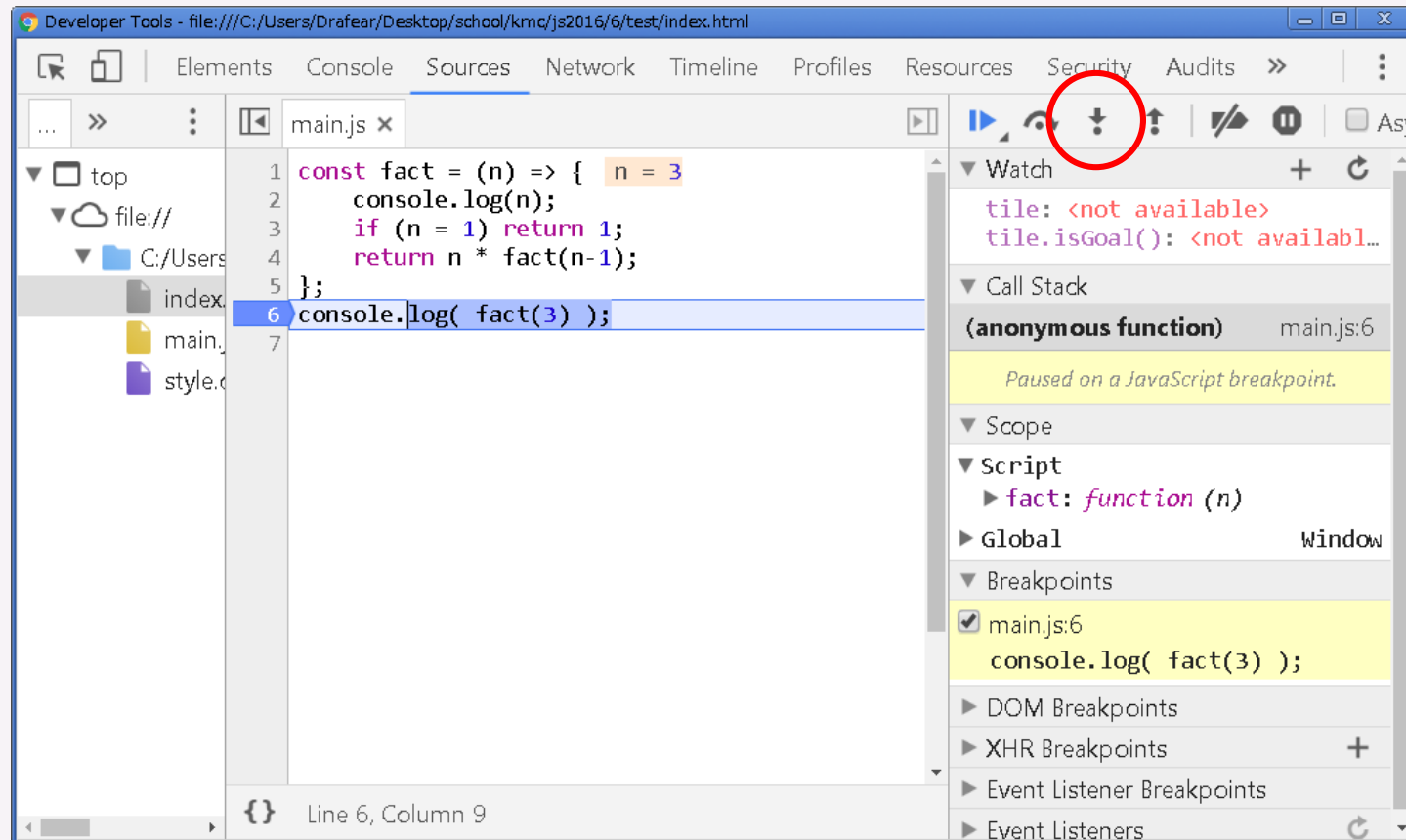
# JavaScript

下の赤丸のボタンか, F11 で 関数の中に入る (ステップイン)



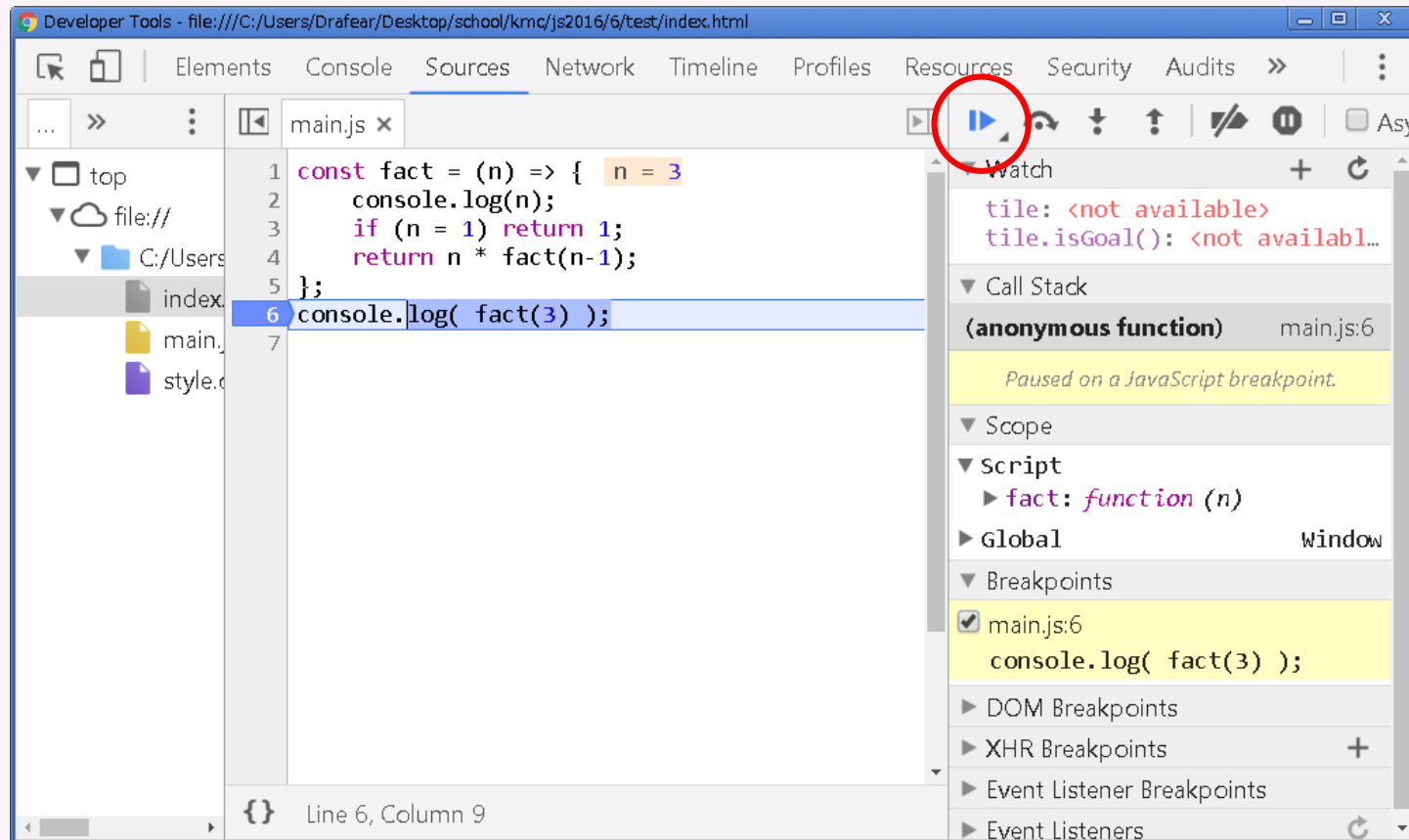
# JavaScript

関数をもし出るときは下の赤丸のボタンか Shift+F11 (ステップアウト)



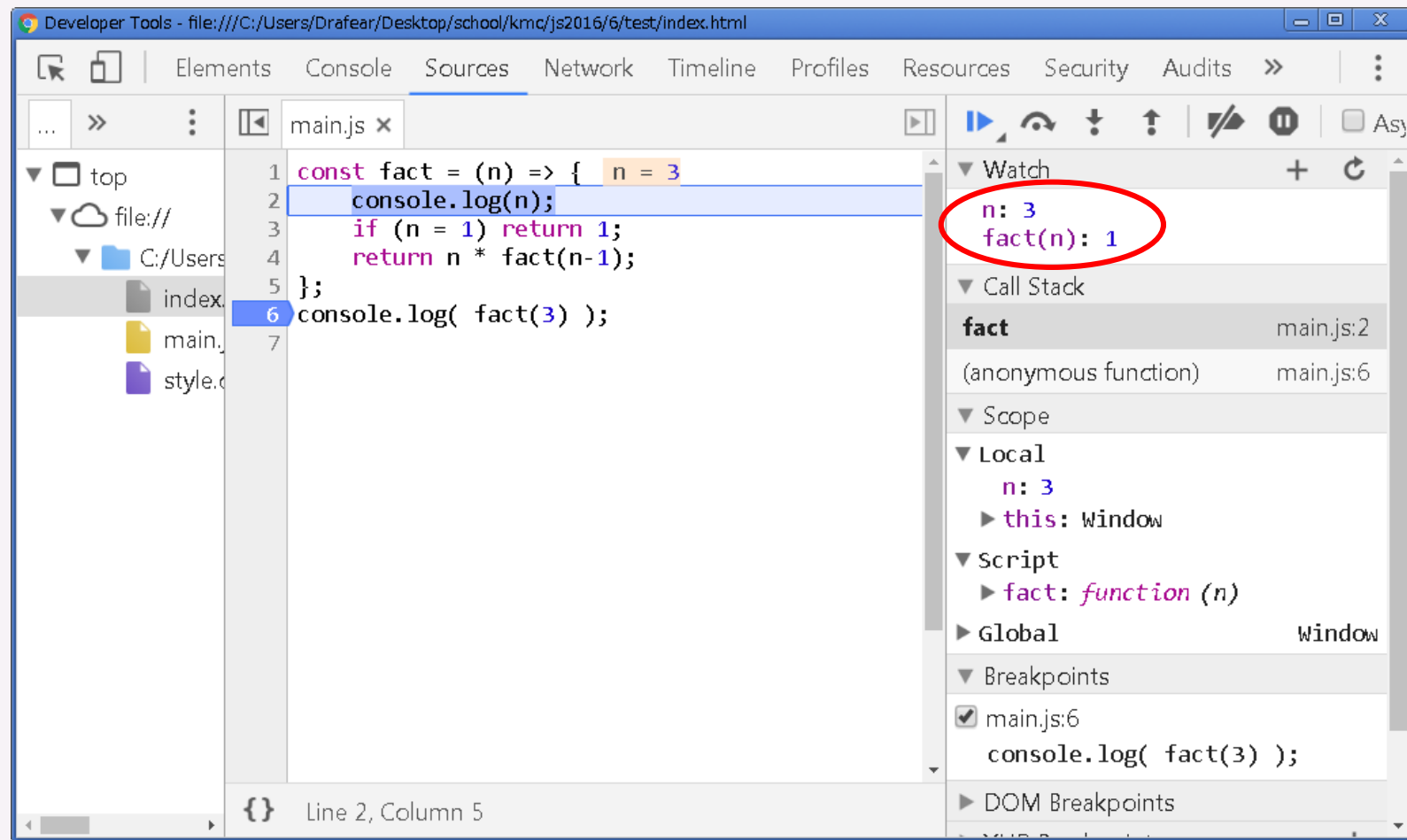
# JavaScript

下の赤丸のボタンが F8 で  
次の break point または プログラムの最後まで実行する



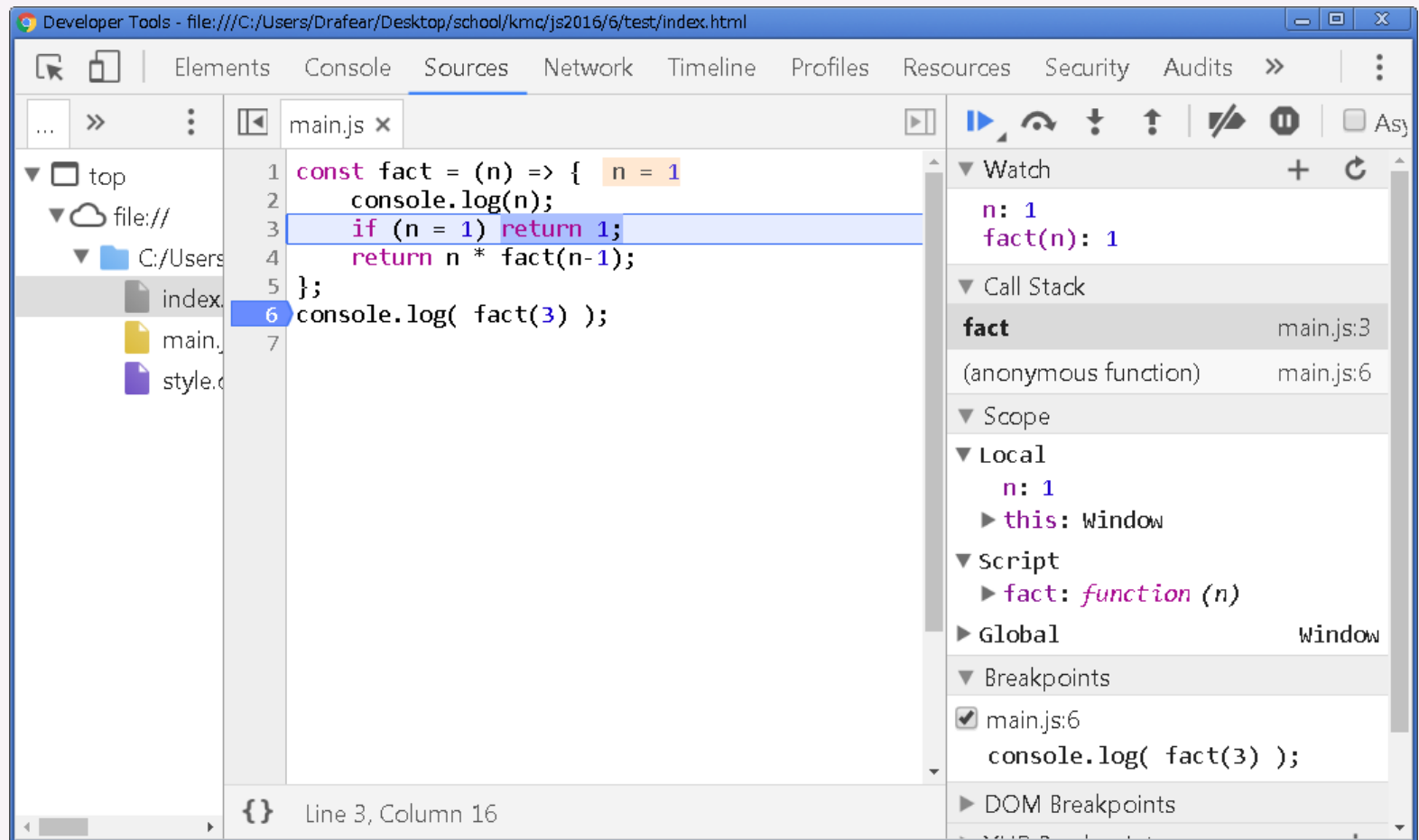
# JavaScript

Watch 欄に見たい変数や式を書けば値が見られる



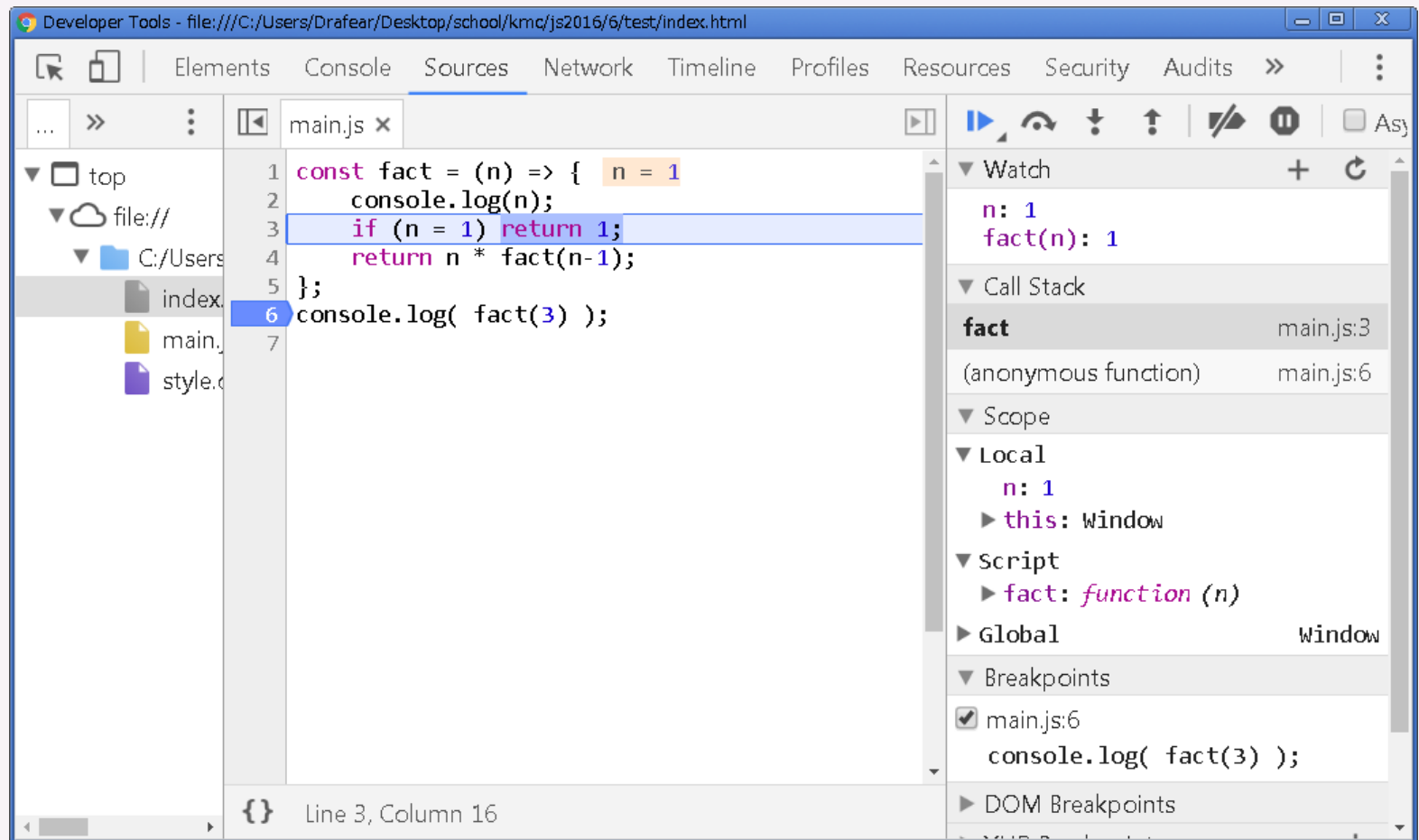
# JavaScript

結局, `fact(3)`なのに, ここで `return` されることがわかる



# JavaScript

if (n = 1) ではなく if (n === 1) でした。





# 4. 何かを作ろう



# めんせきくん

- 計算ボタンを押すと円の面積を計算してくれるめんせきくん



# めんせきくん

- 計算ボタンを押すと円の面積を計算してくれるめんせきくん

`<input type="text" value="114514">`



width: 100px; height: 100px;  
border-radius: 50px;

# 簡単なゲーム・ツールを作る

- ゲーム
  - 宝探しゲーム
  - Hit and Brow
  - High or Low
  - ○×クイズ
- ツール
  - 計算ツール
  - カウンタ(+1, -1), ポケモンの努力値カウンタ
  - 西暦 ⇔ 平成 変換器

# 今後の予定

- 6/5(日) 13:00 ~ 16:00
  - クラスを学ぶ
  - 避けゲーを作る
- 6/12(日) 13:00 ~ 16:00
  - JavaScriptの細かな便利関数の紹介
  - CSS3による簡単なデザインの紹介
- 6/19(日) 13:00 ~ 16:00
  - 文字列処理と正規表現, Ajax
  - Web API いろいろ

