

JavaScript

Agenda

1. About Syntax
2. About DOM & Event
3. About jQuery
4. About jQuery UI

1st Contents

1. About Syntax
2. About DOM & Event
3. About jQuery
4. About jQuery UI

と、その前に...

実行環境について
基本的には自由。お好きな環境でどうぞ。

...と言いたいところですが
少し制限させて下さい。

推奨環境 1

Firebugを使う



※ FireFoxのaddon

適当なディレクトリにindex.htmlを用意し、
FireFoxで開いてFirebugコンソールでJS

<http://getfirebug.com/>

推奨環境 1

Good.

- ・何も気にせず複数行のJSを書ける
- ・.jsファイルと併用することで表現度UP
- ・ページから多くの情報を検知可能
- ・どの道デバッガツールはJS書く上でマスト

Bad.

- ・機能が多い
- ・揮発性が強い

推奨環境 2

Google Chromeを使う



<http://www.google.co.jp/intl/ja/chrome/browser/>

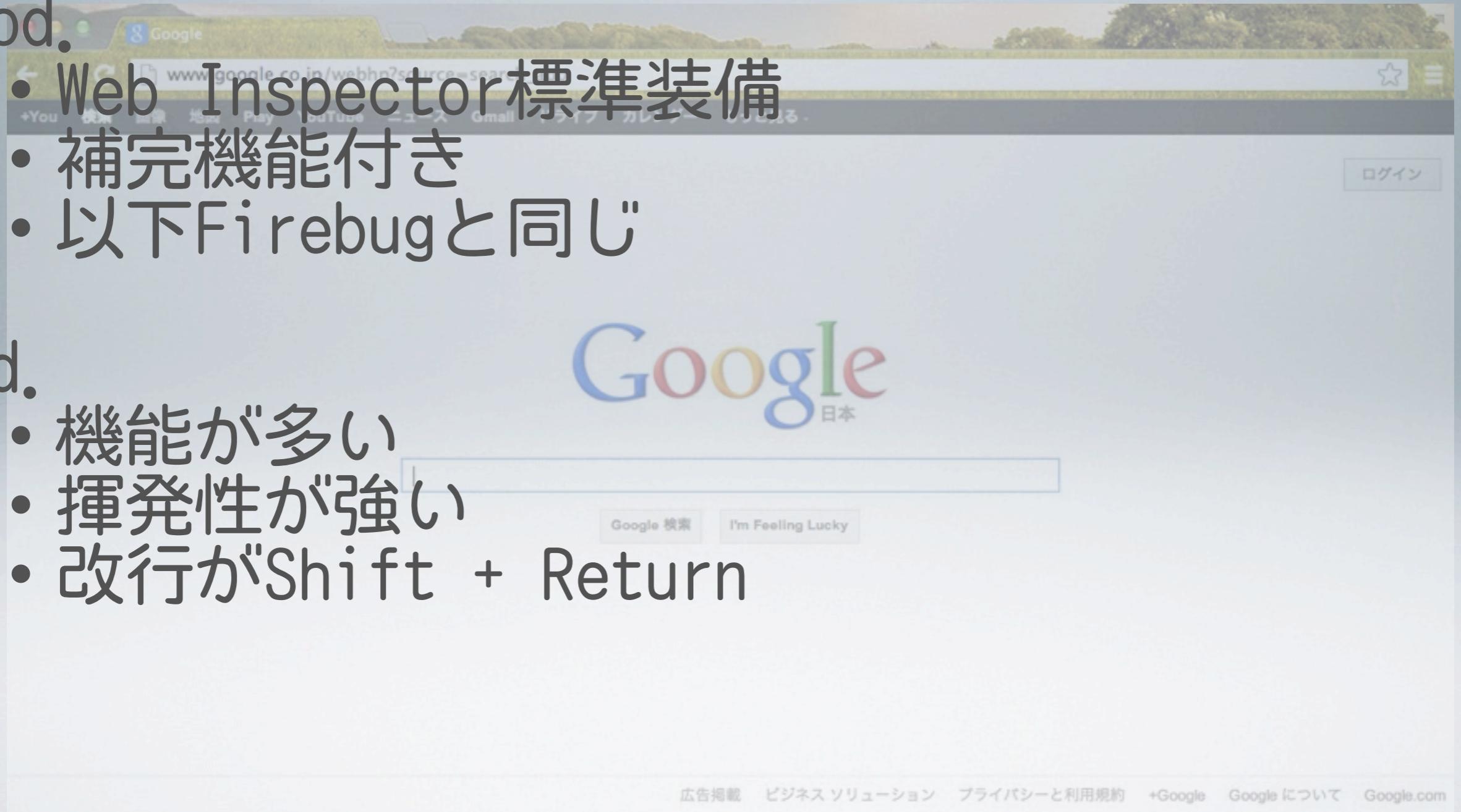
推奨環境 2

Good.

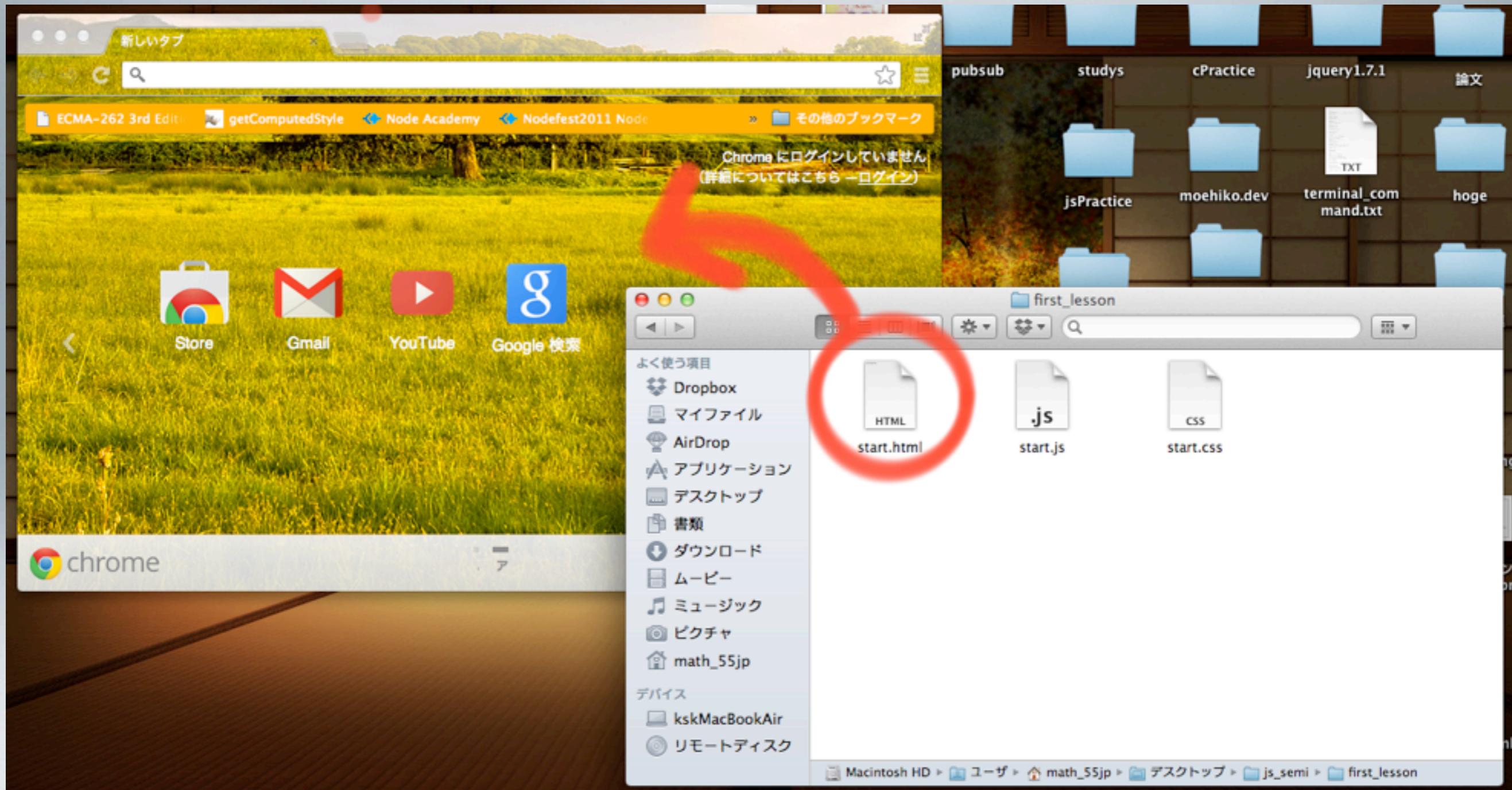
- Web Inspector標準装備
- 補完機能付き
- 以下Firebugと同じ

Bad.

- 機能が多い
- 揮発性が強い
- 改行がShift + Return



実行推奨環境 1



HTMLファイルをブラウザにドロップ

実行推奨環境 1

Good.

- 実際の挙動に限りなく近い
- オフラインでも動作確認できる
- ファイルを編集しながら確認できる

Bad.

- 特になし

実行推奨環境 2

The screenshot shows the jsFiddle.net interface. At the top, there's a navigation bar with a cloud icon, the text "JSFIDDLE", and buttons for "Run", "Save", "TidyUp", and "JSHint". On the right side of the bar is a "Login/Sign up" button. Below the navigation bar, there's a sidebar on the left containing several sections: "Frameworks & Extensions" (with dropdown menus for "No-Library (pure JS)" and "onLoad"), "Fiddle Options", "External Resources", "Languages", "Ajax Requests", "Examples", and "Legal, Credits and Links". At the bottom of the sidebar is a link to "Keyboard shortcuts". The main area consists of four large panes arranged in a grid. The top-left pane is labeled "HTML" and contains a code editor with the number "1" in the top-left corner. The top-right pane is labeled "CSS" and also contains a code editor with the number "1". The bottom-left pane is labeled "JavaScript" and contains a code editor with the number "1". The bottom-right pane is labeled "Result" and is currently empty. The overall layout is clean and organized, typical of a developer's tool for testing and sharing code snippets.

jsfiddle
<http://jsfiddle.net/>

実行推奨環境 2

The screenshot shows the JSFiddle interface with the title "Good." above the "Bad." section. The JSFiddle interface includes tabs for "JS", "CSS", and "Result". The "JS" tab contains the following code:

```
$(document).ready(function() {  
    $("p").text("Hello World");  
});
```

The "CSS" tab is empty. The "Result" tab shows the output: "Hello World".

Good.

- HTML, CSS, JavaScript, Viewが一望できる
- ライブラリ等も選択するだけで使用可能
- 簡単なバージョン管理が出来る
- 作って保存すればどこからでも再編集可能
- 人の書いたソースをフォーク出来る

Bad.

- 本当のブラウザ環境とは少なからず違う
- インデントとかが少し間抜け
- ページが重たい

1st Contents

1. About Syntax
2. About DOM & Event
3. About jQuery
4. About jQuery UI

Variables

```
var a = 10;  
alert(a); // -> 10
```

```
var b = "Hello";  
b = b + " World.";  
alert(b);  
/*  
 ->"Hello World."  
*/
```

```
var c = 5, d = 20, e;  
console.log(c + d); // -> 25
```

```
console.log(e); // -> undefined
```

Variables

```
var a = 10;  
alert(a); // -> 10
```

```
var b = "Hello";  
b = b + " World.";  
alert(b);  
/*  
->"Hello World."  
*/
```

```
var c = 5, d = 20, e;  
console.log(c + d); // -> 25
```

```
console.log(e); // -> undefined
```

Variables lesson

1. 変数「apple」に「"リンゴ"」を代入し、コンソールに出力させる
2. 変数「kozukai」に2500を足して、アラートボックスに出力させる

Variables lesson

1. 変数「apple」に「"リンゴ"」を代入し、コンソールに出力させる
 2. 変数「kozukai」に2500を足して、アラートボックスに出力させる
- ※ ちゃんと出力されましたか？
- ※ 変数の宣言で「var」を忘れてませんか？
- ※ セミコロン忘れてませんか？

Variables Quiz

```
console.log(null + 1); // -> ? (read P.4)
```

```
console.log(a); // -> ?  
var a = 10;
```

```
console.log(b); // -> ?
```

Variables Quiz2

```
(function () {
    var hoge = function () {
        var a = b = c = 20;
    }

    hoge();
    console.log(a); // -> ?
    console.log(b); // -> ?
    console.log(c); // -> ?
})()
```

Literals

P. 19

リテラル（定数）

1. NumericLiteral

e. g.) 1234

2. StringLiteral

e. g.) “hello”

3. BooleanLiteral

e. g.) true

4. NullLiteral

e. g.) null

5. RegularExpressionLiteral

e. g.) /^[Jj]ava[Ss]cript\$/

Array

```
var a = ["aa", "bb", "cc"];
```

```
console.log(a[0]); // -> "aa"
```

```
a[1] = 1;
```

```
console.log(a[1]); // -> 1
```

```
a.push("dd");
```

```
console.log(a[3]); // -> "dd"
```

```
var b = a.pop();
```

```
console.log(a[3], b); // -> undefined, "dd"
```

Array

P. 22

ちなみに、shiftの逆は
unshift()

```
var a = ["aa", "bb", "cc"];
var c = a.shift()
console.log(a[0], c, a[2]);
// ->"bb", "aa", undefined
console.log(a.length); // -> 2
```

Array Quiz

```
var a = [];
a[100] = 10;
```

```
a.push(20);
```

```
console.log(a.indexOf(20)); // -> ?
console.log(a.length); // -> ?
```

```
var b = [,,,];
console.log(b.length) // -> ?
```

Object

```
var obj = {  
  a: 1,  
  b: "hello."  
};
```

```
console.log(obj.a); // -> 1  
console.log(obj["b"]); // ->"hello."  
console.log(obj.c); // -> undefined  
obj.a = 20;  
console.log(obj.a); // -> 20
```

```
console.log("a" in obj); // -> true  
delete obj.a;  
console.log("a" in obj); // -> false
```

ES5からは末尾のプロ
パティにカンマを付けて
も仕様上問題なくなっ
た。ただしIE7以下では
NO. (p. 20)

Object

```
var obj = {  
  a: 1,  
  b: "hello."  
};
```

```
console.log(obj.a); // -> 1  
console.log(obj["b"]); // ->"hello."  
console.log(obj.c); // -> undefined  
obj.a = 20;  
console.log(obj.a); // -> 20
```

```
console.log("a" in obj); // -> true  
delete obj.a;  
console.log("a" in obj); // -> false
```

ES5からは末尾のプロ
パティにカンマを付けて
も仕様上問題なくなっ
た。ただしIE7以下では
NO. (p. 20)

Operator 1

P. 23

1. Add

e. g.) $20 + 15; // \rightarrow 35$

2. Sub

e. g.) $20 - 15; // \rightarrow 5$

3. Mul

e. g.) $20 * 15; // \rightarrow 300$

4. Div

e. g.) $20 / 10; // \rightarrow 2$

5. Mod

e. g.) $20 \% 15; // \rightarrow 5$

6. increment

e. g.) $++1; // \rightarrow 2$

7. less than

e. g.) $20 < 15 // \rightarrow \text{false}$

Operator 2

8. greater than
e.g.) `20 > 15; // -> true`
9. less than or equal
e.g.) `20 <= 20; // -> true`
10. equality
e.g.) `20 == 15; // -> false`
11. identity
e.g.) `20 === "20" // -> false`
12. not equality
e.g.) `20 != 15; // -> true`
13. AND
e.g.) `true && false; // -> false`
14. OR
e.g.) `true || false // -> true`
15. NOT
e.g.) `!false // -> true`

Operator Quiz

```
var a = 10;  
a /= 5;  
console.log(a); // -> ?
```

```
console.log(1 + false == true); // -> ?
```

```
console.log(1 + true); // -> ?
```

if state

P.28

```
if ( 条件式1 ) {  
    // 条件式1がtrueの時の処理文  
} else {  
    // 条件式1がfalseの時の処理文  
}
```

```
if ( 20 < 100 ) {  
    console.log("TRUE");  
} else {  
    console.log("FALSE");  
}
```

※ すごく大切！よく使う

if state 2

```
if ( 条件式1 ) {  
    // 条件式1がtrueの時の処理文  
} else if (条件式2) {  
    /* 条件式1がfalseでかつ  
    条件式2がtrueの時の処理文 */  
}
```

```
var first_exp = false,  
second_exp = true;  
  
if ( first_exp ) {  
    console.log("first: true");  
} else if ( second_exp ) {  
    console.log("first: false && second:  
true");  
}
```

if state lesson

1. 変数 a が10以上かつ変数 b が20未満の時
コンソールに「now!」と出力する
2. 変数 a の値が奇数ならアラートで「奇数」と出力し、偶数なら「偶数」と出力する

if state lesson

1. 変数 a が"10以上かつ変数 b が"20未満の時
コンソールに「now!」と出力する
2. 変数 a の値が奇数ならアラートで「奇数」と出力し、偶数なら「偶数」と出力する

- ※ 奇数か偶数かは「%」を使って判断するよ
- ※ ちゃんと出力されましたか？
- ※ 変数の宣言で「var」を忘れてませんか？
- ※ セミコロン忘れてませんか？

if Quiz

```
(function () {
    var a = 10;
    if (a > 5) {
        a = 7;
    }
    console.log(a); // -> ?
})();
```

```
(function () {
    if (true) {
        var a = 5;
    }
    console.log(a); // -> ?
})();
```

switch state

P.30

```
switch ( 評価式 ) {  
    case 結果1:  
        // 評価式の結果が結果1になるときの処理  
        break;  
    case 結果2:  
        // 評価式の結果が結果2になるときの処理  
        break;  
    default:  
        // 評価式の結果が結果1, 2にならないときの処理  
}
```

※ あまり使われない・・・

while state

P.31

```
while ( 条件式 ) {  
    // 条件式がtrueの間、繰り返される処理  
}
```

```
var i = 0;  
  
while (i < 10) {  
    console.log(i);  
    i++;  
}
```

※ あまり使われない・・・

do-while state

P.32

```
do {  
    /* 条件式に関わらず、  
       少なくとも一度は行いたい処理 */  
} while ( 条件式 )
```

```
var i = 0;  
  
do {  
    console.log(i);  
    i++;  
} while (i < 0);
```

※ あまり使われない・・・

for state

P.32

```
for (初期化式; 条件式; 変化式) {  
    // 条件式がtrueの間、繰り返される処理  
}
```

```
for (var i = 0, j = 10; i < j; i++) {  
    console.log(i);  
}
```

※ すごく大切！よく使う

for-in state

P.32

```
for (引数 in オブジェクト) {  
    // オブジェクトが持っている  
    // プロパティ数分ループする  
}
```

```
var obj = {  
    a: 10,  
    b: 20,  
    c: 30  
};
```

```
for (var key in obj) {  
    console.log(key, obj[key]);  
}
```

※ たまに使う

Function

P. 25

```
function sum (a, b) {  
    return a + b;  
}  
  
var result = sum(10, 20);  
console.log(result); // -> 30;
```

Function

P. 25

```
function sum (a, b) {  
    return a + b;  
}  
  
var result = sum(10, 20);  
console.log(result); // -> 30;
```

Function lesson

1. 数値を一つ引数にとり、
+10して返す関数を作る
2. 関数の呼び出し結果を変数に入れず、
`console.log()`の中で直接関数を呼び出す

Function lesson

1. 数値を一つ引数にとり、
+10して返す関数を作る
2. 関数の呼び出し結果を変数に入れず、
`console.log()`の中で直接関数を呼び出す

- ※ ちゃんと出力されましたか？
- ※ 変数の宣言で「var」を忘れてませんか？
- ※ セミコロン忘れてませんか？

Function 2

```
function sum () {  
    return arguments[0] + arguments[1];  
}  
  
var result = sum(10, 20);  
console.log(result); // -> 30;
```

Function 2

```
function sum () {  
    return arguments[0] + arguments[1];  
}  
  
var result = sum(10, 20);  
console.log(result); // -> 30;
```

Function Quiz

```
console.log(sum(10, 20)); // -> ?  
console.log(sum2(15, 25)); // -> ?
```

```
function sum () {  
    return arguments[0] + arguments[1];  
}
```

```
var sum2 = function () {  
    return arguments[0] + arguments[1];  
}
```

Call by reference

```
var obj = {  
  a: 1  
};
```

```
var copy = obj;  
copy.a = 20;
```

```
console.log(obj.a); // -> 20
```

P.36

Call by reference 2

```
var obj_1 = {  
  a: 123  
};
```

```
var obj_2 = {  
  a: 123  
};
```

```
console.log(obj_1 == obj_2); // -> false
```

Call by reference 3

```
var arr = [1, 2, 3, 4, 5],  
    dummy = arr;  
  
function dummy_pop (dum) {  
    dum.pop();  
    dum.pop();  
    dum.pop();  
}  
  
dummy_pop(dummy);  
console.log(arr); // -> [1, 2]
```

Quiz

P.37

```
function replace ( input ) {  
  input = {a: 100};  
}
```

```
var obj = {a: 1};  
replace(obj);
```

```
console.log(obj.a) // -> ?
```

2nd Contents

1. About Syntax
2. About DOM & Event
3. About jQuery
4. About jQuery UI

と、その前に

Functionについて

JavaScriptにおいてFunctionは死ぬほど大切で
かつ、少し癖があって難しいポイント

なのでもう少し詳しく解説します。

Function

どちらも関数を定義しておいて呼び出している

```
function sum (a, b) {  
    return a + b;  
}
```

```
var result = sum(10, 20);  
console.log(result); // -> 30;
```

```
function sum () {  
    return arguments[0] + arguments[1];  
}
```

```
var result = sum(10, 20);  
console.log(result); // -> 30;
```

Function

どちらも関数を定義しておいて呼び出している

```
function sum (a, b) {  
    return a + b;  
}
```

```
var result = sum(10, 20);  
console.log(result); // -> 30;
```

```
function sum () {  
    return arguments[0] + arguments[1];  
}
```

```
var result = sum(10, 20);  
console.log(result); // -> 30;
```

Function

実はこんなことも可能

```
function sum (a, b) {  
    return a + b;  
}
```

```
var result = sum;  
console.log( result(10, 20) ); // -> 30;
```

```
var result = function (a, b) {  
    return a + b;  
}(10, 20);  
  
console.log(result); // -> 30;
```

Function

実はこんなことも可能

```
function sum (a, b) {  
    return a + b;  
}
```

```
var result = sum;  
console.log(result(10, 20)); // -> 30;
```

```
var result = function (a, b) {  
    return a + b;  
}(10, 20);
```

```
console.log(result); // -> 30;
```

First-class Function

JavaScriptのFunctionは第一級オブジェクトである

つまり、変数に格納ができたり、
引数として渡すことも出来る

```
var result = function (callback) {  
    return callback(10, 20);  
};  
  
console.log(  
    result(function (a, b) {  
        return a + b;  
    })  
) // -> 30
```

First-class Function

JavaScriptのFunctionは第一級オブジェクトである

つまり、変数に格納ができたり、
引数として渡すことも出来る

```
var result = function (callback) {  
    return callback(10, 20);  
};  
  
console.log(  
    result(function (a, b) {  
        return a + b;  
    })  
) // -> 30
```

First-class Function

当然こんな風に、
「呼び出されたら関数を返す」関数なんかも書ける

```
var x_plus_y = function (x) {  
    return function (y) {  
        console.log(x + y);  
    };  
};  
  
x_plus_y(10)(20); // -> 30
```

こと関数を返す関数に関しては、
面白い挙動を確認することが出来る

Closure

```
function counter () {  
  var i = 0;  
  return function () {  
    console.log(i);  
    i++;  
  }  
}
```

```
cou = counter();  
cou(); //-> 0  
cou(); //-> 1  
cou(); //-> 2  
cou(); //-> 3
```

Closure Quiz

```
(function () {
  for (var i = 0, j = 10; i < j; i++) {
    setTimeout(function () {
      console.log(i);
    }, 100);
  }
})();
```

上記の出力が0 1 2 3 4 ... となるよう、
setTimeoutの中を書き換える

ヒント: setTimeoutは非同期なので100ms待っている間にfor文のループが進んでしまって現状の出力になっている

2nd Contents

P.89

1. About Syntax
2. About DOM & Event
3. About jQuery
4. About jQuery UI

JavaScriptでしたい事

HTMLは文書と文書構造を定義する

CSSは文書の見せ方を定義する

JavaScriptは振る舞いを定義する

極論をいうと、JavaScriptはなくても良い

ないといけないページは作るべきでない

あくまで、ユーザビリティを向上するために

用いるのが吉

そもそもDOMとは？

DOM(Document Object Model)

XMLやHTMLを操作するためのAPI
仕様はW3Cにより策定されている

本来JavaScriptとは別物ですが、
ブラウザ上のJavaScript環境では
それらの基本的なAPI(関数)がサポートされている

要は、Webページの要素を
JavaScriptで扱えるように表現したのがDOMってこと

```
<html>
  <head>
    <title>Colors</title>
  </head>
  <body>
    <h1>Color List</h1>
    <div id="colorList">
      <ul class="richromatic">
        <li>Cyan</li>
        <li>Magenta</li>
        <li>Yellow</li>
      </ul>
      <ul class="RGB">
        <li>Red</li>
        <li>Green</li>
        <li>Blue</li>
      </ul>
    </div>
  </body>
</html>
```

要素アクセスAPI

ブラウザの大枠はWindowオブジェクト

その中にdocumentオブジェクトがあり、

documentオブジェクトには要素アクセスのための
APIが用意されている

```
var elem = document.getElementById("colorList");
alert(elem.nodeName); //-> 'DIV'
```

```
var rgb_list = document.getElementsByClassName("RGB");
alert(rgb_list[0].nodeName); //-> 'UL'
```

```
alert(rgb_list[0].firstElementChild.textContent); //-> 'Red'
```

要素アクセスAPI

ブラウザの大枠はWindowオブジェクト

その中にdocumentオブジェクトがあり、

documentオブジェクトには要素アクセスのための

APIが用意されている

```
var elem = document.getElementById("colorList");
alert(elem.nodeName); //-> 'DIV'
```

```
var rgb_list = document.getElementsByClassName("RGB");
alert(rgb_list[0].nodeName); //-> 'UL'
```

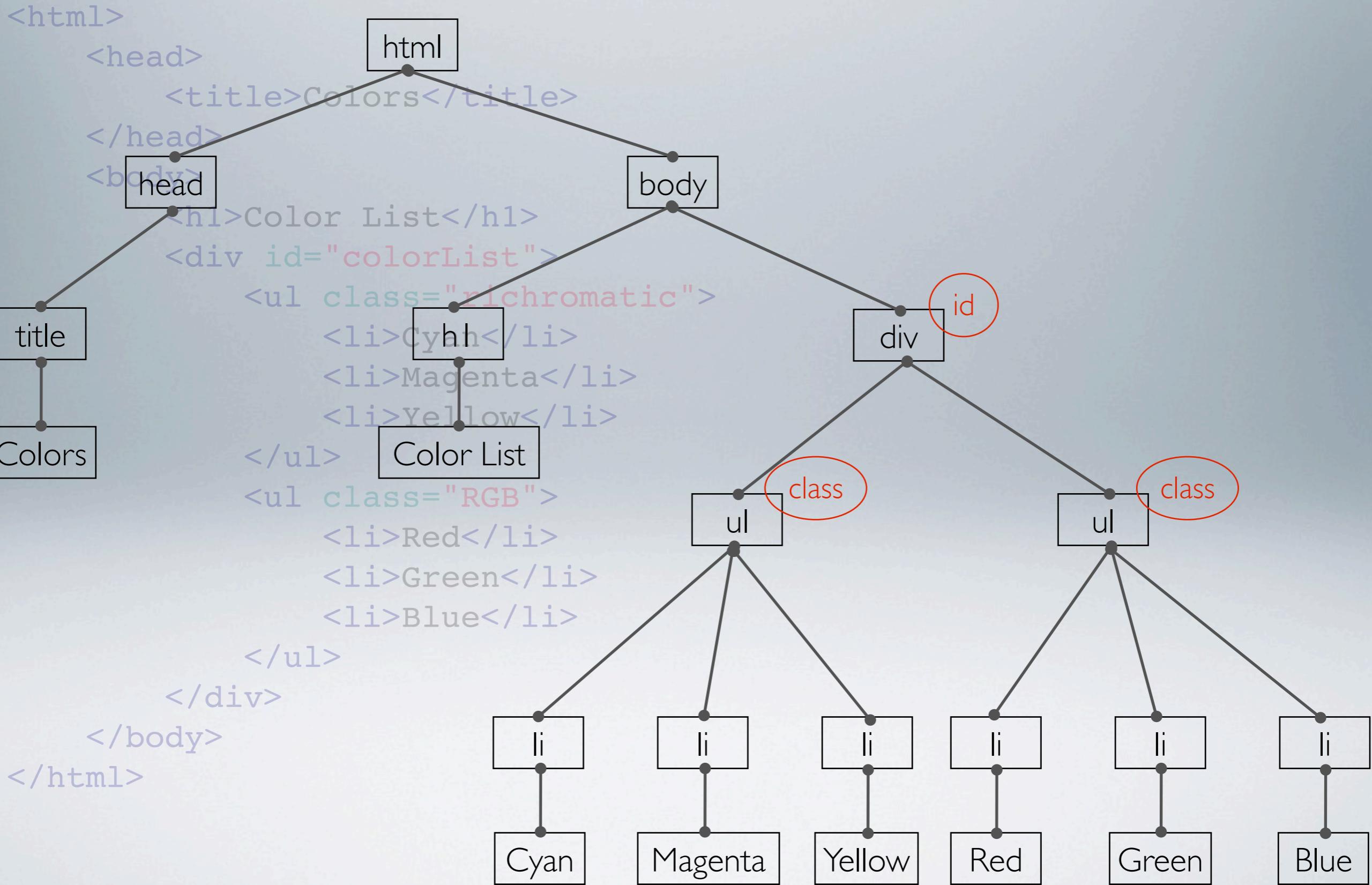
```
alert(rgb_list[0].firstElementChild.textContent); //-> 'Red'
```

要素アクセスlesson

1. 適当なタグにid属性を付与し、getElementByIdを使って要素にアクセスしてみよう！
2. getElementsByTagNameを使って、divタグを全て取得してみよう！

要素アクセスquiz

1. `document.evaluate()`を使ってXPathで要素にアクセスしてみよう！



要素アクセスAPI

getElementByIdやgetElementsByClassName、
getElementsByTagNameなんかがあるけど、

モダンブラウザでは
querySelector、querySelectorAllが使える
からそちらが簡単です
指定するのはCSSセレクタ

```
var elem = document.querySelector("#colorList");
alert(elem.nodeName); //-> 'DIV'

var rgb_list = document.querySelectorAll(".RGB");
alert(rgb_list[0].nodeName); //-> 'UL'

alert(rgb_list[0].firstElementChild.textContent); //-> 'Red'
```

要素アクセスAPI

getElementByIdやgetElementsByClassName、
getElementsByTagNameなんかがあるけど、

モダンブラウザでは
querySelector、querySelectorAllが使える
からそちらが簡単です
指定するのはCSSセレクタ

```
var elem = document.querySelector("#colorList");
alert(elem.nodeName); //-> 'DIV'

var rgb_list = document.querySelectorAll(".RGB");
alert(rgb_list[0].nodeName); //-> 'UL'

alert(rgb_list[0].firstElementChild.textContent); //-> 'Red'
```

要素アクセスlesson2

1. colorlistのhtmlで、querySelectorを用いて、
div(class="RGB")の子要素の内、一番始めに出てくる
li要素のテキストノードを取得してみよう！
2. colorlistのhtmlで、querySelectorAllを用いて、全
li要素を取得してみよう！

便利なデバッグ

JavaScriptを書く上で、
現在変数の中にはどんな値が入っているのか？や、
返ってきたオブジェクトはどんなプロパティ、メソッド
を持っているのか等を確認するには、
`console.log`や`console.dir`が便利！

確認はデバッガで行える

```
var elem = document.querySelector("#colorList");
console.log(elem.nodeName); //-> 'DIV'

var rgb_list = document.querySelectorAll(".RGB");
console.dir(rgb_list[0]);
```

便利なデバッグ

JavaScriptを書く上で、
現在変数の中にはどんな値が入っているのか？や、
返ってきたオブジェクトはどんなプロパティ、メソッド
を持っているのか等を確認するには、
console.logやconsole.dirが便利！

確認はデバッガで行える

```
var elem = document.querySelector("#colorList");
console.log(elem.nodeName); //-> 'DIV'

var rgb_list = document.querySelectorAll(".RGB");
console.dir(rgb_list[0]);
```

JavaScriptを書く場所

ところで、JavaScriptはどこに書くと良いでしょう？

1. リンク
2. <script type=”text/javascript”>ここ</script>
3. <div onclick=”ここ”>適当な文字列</div>

JavaScriptを書く場所

ところで、JavaScriptはどこに書くと良いでしょう？

1. リンク
2. <script type=”text/javascript”>ここ</script>
3. <div onclick=”ここ”>適当な文字列</div>

どこに書いても動くけど。。。。

控えめなJavaScript

JavaScriptはHTMLとは別ファイルに書きましょう！

なぜなら、

HTMLは文書構造を書くものであって、
振る舞いを書くべきではないから

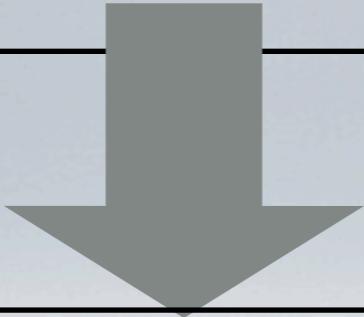
別ファイルに書いておけば使い回しも可能

HTMLとCSSを書く人とJSを書く人で分業も可能

JSのテストも書き易くなる

切り離してみよう

```
<body>
  <script type="text/javascript">
    alert("hogehoge");
  </script>
</body>
```



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

```
alert("hogehoge");
```

Window Object

ブラウザで動くJavaScriptを書く上で、無視する事が出来ないグローバルがwindow

※ 下記はあくまでイメージ

```
window = {
  document: {
    getElementById: function (str) {
      return element;
    },
    getElementsByClass: function (str) {
      return elements_arr;
    },
    ...
  },
  alert: function (obj) {
    ...
  },
  ...
}
```

```
alert === window.alert; //--> true
```

Event

ブラウザ上で動かすJavaScriptはイベント駆動であることを意識します。

要は、「何」が起こった時に「どう」動くのか。

```
document.onclick = function () {  
    alert("hogehoge");  
}
```

「クリック」というイベントが発生した時に、
「hogehoge」という文字列をアラートする

Event

ブラウザ上で動かすJavaScriptはイベント駆動であることを意識します。

要は、「何」が起こった時に「どう」動くのか。

```
document.onclick = function () {  
    alert("hogehoge");  
}
```

「クリック」というイベントが発生した時に、
「hogehoge」という文字列をアラートする

Event lesson

1. 用意されたHTMLファイルをブラウザで表示し、表示されているdiv要素（青い四角）の要素を取得し、タグの名前を出力しよう！
2. 上記で取得した要素に対し、クリックされたらタグの名前をalertで出力するようにイベントをはってみよう！

Event quiz

1. 用意されたHTMLの全div要素にクリックイベントをはってみよう！
2. オレンジのdiv要素がクリックされた時に親divのイベントが発火するのを防ごう！

hint: イベント伝播とstopPropagationメソッド

Event

ところで、ある要素に対して2つのイベント処理をひも付けるとどうなるでしょう？

```
var first = document.querySelector("#first_box");
first.onclick = function () {
  alert("first event.");
};

first.onclick = function () {
  alert("second event.");
};
```

Event

そうなんです
イベント処理が上書きされてしまって、
一つ目の処理が実行されないんです

```
var first = document.querySelector("#first_box");
first.onclick = function () {
    alert("first event."); // -> 実行されない (ToT)
};

first.onclick = function () {
    alert("second event.");
};
```

Event

そんな時有効なのが

addEventListener(“event”, 处理, false);

```
var first = document.querySelector("#first_box");
first.addEventListener("click", function (evt) {
    alert("first event."); // -> 実行される
}, false);

first.addEventListener("click", function (evt) {
    alert("second event.");
}, false);
```

これを使えばいくらでも要素に対して
イベント処理を追加できる！

Event

そんな時有効なのが

addEventListener(“event”, 处理, false);

```
var first = document.querySelector("#first_box");
first.addEventListener("click", function (evt) {
    alert("first event."); // -> 実行される
}, false);

first.addEventListener("click", function (evt) {
    alert("second event.");
}, false);
```

これを使えばいくらでも要素に対して
イベント処理を追加できる！

Event lesson

1. colorlistのhtmlで、li要素すべてにクリックイベントを張り、クリックされた要素の中身（テキストノード）の文字列を出力するようにしてみよう！
2. 1を一つ一つのliに対して実装した人は、全li要素をquerySelectorAllで取得し、for文で回しながらイベントを張ってみよう！
3. addEventListenerの第二引数の処理関数で、引数にわたってくるevtの中身をconsole.dir()で確認してみよう！

setTimeout

setTimeout(function () {/* 处理 */}, 数値ms);

数値(ミリ秒)後、処理を行う

返り値はタイマーID

```
var first = document.querySelector("#first_box");
first.addEventListener("click", function (evt) {
    setTimeout(function () {
        alert("2秒たったよ!");
    }, 2000); // 2秒後に上記の処理
}, false);
```

setTimeout

```
setTimeout(function () {/* 处理 */}, 数値ms);
```

数値(ミリ秒)後、処理を行う

返り値はタイマーID

```
var first = document.querySelector("#first_box");
first.addEventListener("click", function (evt) {
    setTimeout(function () {
        alert("2秒たったよ!");
    }, 2000); // 2秒後に上記の処理
}, false);
```

clearTimeout

clearTimeout(タイマーID);

setTimeoutで予定された処理を中断する

返り値はなし

```
var first = document.querySelector("#first_box");
first.addEventListener("click", function (evt) {
  var timer_id = setTimeout(function () {
    alert("3秒たったよ!");
  }, 3000);
  setTimeout(function () {
    clearTimeout(timer_id);
    console.log("2秒たったよ!");
  }, 2000); // 2秒後に上記の処理
}, false);
```

clearTimeout

clearTimeout(タイマーID);

setTimeoutで予定された処理を中断する

返り値はなし

```
var first = document.querySelector("#first_box");
first.addEventListener("click", function (evt) {
  var timer_id = setTimeout(function () {
    alert("3秒たったよ!");
  }, 3000);
  setTimeout(function () {
    clearTimeout(timer_id);
    console.log("2秒たったよ!");
  }, 2000); // 2秒後に上記の処理
}, false);
```