

2nd Contents

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 ... というのは過去の話かも...

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. <**a** href="ここ">リンク</**a**>
2. <**script** type="text/javascript">ここ</**script**>
3. <**div** onclick="ここ">適当な文字</**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. <**a** href="ここ">リンク</**a**>
2. <**script** type="text/javascript">ここ</**script**>
3. <**div** onclick="ここ">適当な文字</**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. <**a** href="ここ">リンク</**a**>
2. <**script** type="text/javascript">ここ</**script**>
3. <**div** onclick="ここ">適当な文字</**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

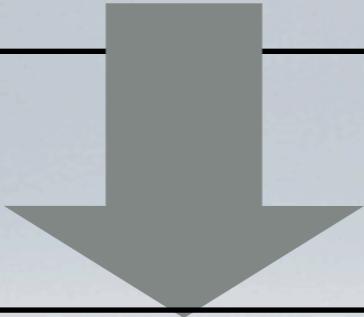
別ファイルに書いておけば使い回しも可能

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("切り離そう!");
  </script>
</body>
```



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

```
  alert("切り離そう!");
```

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: {
    ...
  },
  alert: function (data) {
    ...
  },
  ...
}
```

グローバル

JavaScriptではページ1枚につき、
一つの大きな**window**というオブジェクトが存在する

```
window = {  
  document: {  
    ...  
  },  
  alert: function (data) {  
    ...  
  },  
  ...  
}
```

グローバル

同じページにロードされる全てのJavaScript内の
変数や関数等の名前は、
`window`オブジェクトの要素となる

window	名前 1	値 1
	名前 2	値 2
	名前 3	値 3

グローバル

windowオブジェクトを共有しているので、
複数のJavaScriptファイルを読み込むと
困った現象が起きることもある

effect.htmlの中では下記の二つを順番にロードする

1. effect_1.js
2. effect_2.js

グローバル

effect.htmlの場合

effect_1.jsをロードした時

```
var effected_value = "I am in global.";
```

window

effected_value

"I am in global."

...

...

グローバル

effect.htmlの場合

effect_2.jsをロードした時

```
alert(effectected_value); // -> I am in global.
```

window

effectected_value

"I am in global."

グローバル

effect.htmlの場合

effect_2.jsをロードした時

```
alert(effectected_value); // -> I am in global.
```

html内でeffect_1.jsとeffect_2.jsを
順次ロードした結果、

中で使われる変数等は全て

windowオブジェクト内の要素になる

Window

effectected_value

"I am in global."

...

グローバル

effect2.htmlの場合

effect_1.jsをロードした時

```
var effected_value = "I am in global.";
```

window

effected_value

"I am in global."

...

...

グローバル

effect2.htmlの場合

effect_inserter.jsをロードした時

```
var effected_value = "effected !!";
```

window

effected_value

"I am in global."

"effected !!"



グローバル

effect2.htmlの場合

effect_2.jsをロードした時

```
alert(effectected_value); // -> effected !!
```

window

effectected_value

"effectected !!"

グローバル

effect2.htmlの場合

effect_2.jsをロードした時

```
alert(effectected_value); // -> effected !!
```

この様な現象を名前の衝突と呼ぶ。

名前の衝突は、

グローバル空間に名前を登録することで

起こりうる

window

effectected_value

"effected !!"

グローバル

グローバルを汚染せずにJavaScriptを書くためには

```
(function () {  
    // ここはグローバルを汚染しない  
}());
```

effect_inserter.jsを次のようにしてみよう

```
(function () {  
    var effected_value = "effected !!!";  
}());
```

無名関数 + 即時実行式

```
(function () {  
    // ここはグローバルを汚染しない  
}());
```

名前の無い関数なので**無名関数**

```
(function () {  
    // ここはグローバルを汚染しない  
}());
```

無名関数 + 即時実行式

```
(function () {  
    // ここはグローバルを汚染しない  
}());
```

```
(function () {  
    // ここはグローバルを汚染しない  
}());
```

関数を定義後すぐに呼び出すので即時実行式

無名関数 + 即時実行式

ただし注意が必要で...

```
function () {  
    // ここはグローバルを汚染しない  
}();
```

この様に書いても無名関数 + 即時実行式
になっているように見えるが、
function () {} までと、最後の () の部分が、
別物と解釈されてしまう。。。。

無名関数 + 即時実行式

なので

```
(function () {  
    // ここはグローバルを汚染しない  
}());
```

全体を()で囲うことで、
実行の()部分も含めて一つの文ですよ！
ということを明確にする必要がある

では、気を取り直してDOM

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

[fifth_lesson/1_1.html]

```
<!DOCTYPE html>
<html>
  <head>
    <title>JavaScript</title>
  </head>
  <body>
    <h1>Color List</h1>
    <div>
      <ul>
        <li>Red</li>
        <li>Green</li>
        <li>Blue</li>
      </ul>
    </div>
    <script type="text/javascript">
(function () {
  // この間に書いてね！
})();
  </script>
  </body>
</html>
```

要素アクセスAPI

JavaScriptでHTMLを操作するためには、
まず、操作したい要素をオブジェクトとして取得する
必要がある

```
var elem = document.querySelector("h1");
alert(elem.nodeName);

var rgb_list = document.querySelectorAll("div ul");
alert(rgb_list[0].nodeName);

alert(rgb_list[0].firstElementChild.textContent);
```

要素アクセスAPI

JavaScriptでHTMLを操作するためには、
まず、操作したい要素をオブジェクトとして取得する
必要がある

```
var elem = document.querySelector("h1");
alert(elem.nodeName);

var rgb_list = document.querySelectorAll("div ul");
alert(rgb_list[0].nodeName);

alert(rgb_list[0].firstElementChild.textContent);
```

[fifth_lesson/1_1.html]

```
<!DOCTYPE html>
<html>
  <head>
    <title>JavaScript</title>
  </head>
  <body>
    <h1>Color List</h1>
    <div>
      <ul>
        <li>Red</li>
        <li>Green</li>
        <li>Blue</li>
      </ul>
    </div>
    <script type="text/javascript">
(function () {
  // この間に書いてね !
} ());
    </script>
  </body>

var elem = document.querySelector("h1");
alert(elem.nodeName);

var rgb_list = document.querySelectorAll("div ul");
alert(rgb_list[0].nodeName);

alert(rgb_list[0].firstElementChild.textContent);
```

[fifth_lesson/1_1.html]

```
<!DOCTYPE html>
<html>
  <head>
    <title>JavaScript</title>
  </head>
  <body>
    <h1>Color List</h1>
    <div>
      <ul>
        <li>Red</li>
        <li>Green</li>
        <li>Blue</li>
      </ul>
    </div>
    <script type="text/javascript">
(function () {
  // この間に書いてね !
} ());
    </script>
  </body>
```

```
var elem = document.querySelector("h1");
alert(elem.nodeName);
```

```
var rgb_list = document.querySelectorAll("div ul");
alert(rgb_list[0].nodeName);
```

```
alert(rgb_list[0].firstElementChild.textContent);
```



[fifth_lesson/1_1.html]

```
<!DOCTYPE html>
<html>
  <head>
    <title>JavaScript</title>
  </head>
  <body>
    <h1>Color List</h1>
    <div>
      <ul>
        <li>Red</li>
        <li>Green</li>
        <li>Blue</li>
      </ul>
    </div>
    <script type="text/javascript">
(function () {
  // この間に書いてね !
} ());
    </script>
  </body>
```

```
var elem = document.querySelector("h1");
alert(elem.nodeName);
```

```
var rgb_list = document.querySelectorAll("div ul");
alert(rgb_list[0].nodeName);
```

```
alert(rgb_list[0].firstElementChild.textContent);
```

rgb_list[0].firstElementChild

[fifth_lesson/1_1.html]

```
<!DOCTYPE html>
<html>
  <head>
    <title>JavaScript</title>
  </head>
  <body>
    <h1>Color List</h1>
    <div>
      <ul>
        <li>Red</li>
        <li>Green</li>
        <li>Blue</li>
      </ul>
    </div>
    <script type="text/javascript">
(function () {
  // この間に書いてね !
} ());
    </script>
  </body>
```

```
var elem = document.querySelector("h1");
alert(elem.nodeName);
```

```
var rgb_list = document.querySelectorAll("div ul");
alert(rgb_list[0].nodeName);
```

```
alert(rgb_list[0].firstElementChild.textContent);
```

rgb_list[0].lastElementChild

[fifth_lesson/1_1.html]

```
<!DOCTYPE html>
<html>
  <head>
    <title>JavaScript</title>
  </head>
  <body>
    <h1>Color List</h1>
    <div>
      <ul>
        <li>Red</li>
        <li>Green</li>
        <li>Blue</li>
      </ul>
    </div>
    <script type="text/javascript">
(function () {
  // この間に書いてね !
} ());
    </script>
  </body>

```

var elem = document.querySelector("h1");
alert(elem.nodeName);

var rgb_list = document.querySelectorAll("div ul");
alert(rgb_list[0].nodeName);

alert(rgb_list[0].firstElementChild.textContent);



rgb_list[0].children

要素アクセスlesson

[fifth_lesson/1_1.html]

1. querySelectorAllを用いてli要素を配列で取得し、
for文を用いて各要素のテキストをコンソールに出力し
よう

2. querySelectorでbody要素を取得し、
firstElementChildやchildrenプロパティを使って、
GreenのGreenをコンソールに出力しよう

その他の要素プロパティ

createElement(), textContent, parentNode,
appendChild(), insertBefore(), innerHTML

```
(function () {
  var elem_red = document.querySelector("li"),
      elem_pink = document.createElement("li"),
      elem_orange = document.createElement("li");

  elem_pink.textContent = "PINK";
  elem_red.parentNode.appendChild(elem_pink);

  elem_orange.textContent = "ORANGE";
  elem_red.parentNode.insertBefore(elem_orange, elem_red);

  elem_red.innerHTML = '<li style="background-color: red;"> RED </li>';
})());
```

その他の要素プロパティ

createElement(), textContent, parentNode,
appendChild(), insertBefore(), innerHTML

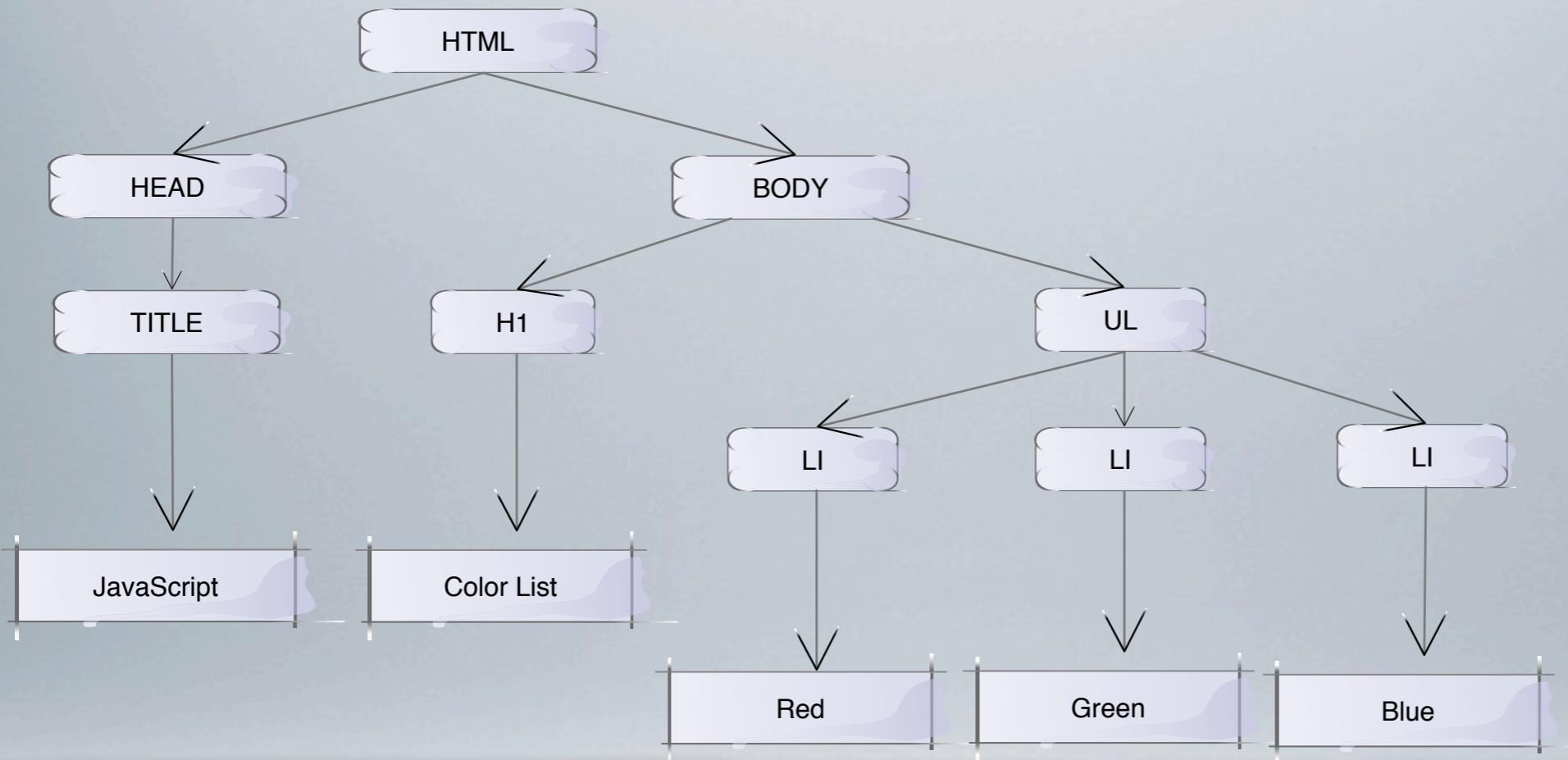
```
(function () {
  var elem_red = document.querySelector("li"),
      elem_pink = document.createElement("li"),
      elem_orange = document.createElement("li");

  elem_pink.textContent = "PINK";
  elem_red.parentNode.appendChild(elem_pink);

  elem_orange.textContent = "ORANGE";
  elem_red.parentNode.insertBefore(elem_orange, elem_red);

  elem_red.innerHTML = '<li style="background-color: red;"> RED </li>';
})());
```

その他の要素プロパティ



```
(function () {
  var elem_red = document.querySelector("li"),
      elem_pink = document.createElement("li"), → LI
      elem_orange = document.createElement("li"); → LI

  elem_pink.textContent = "PINK";
  elem_red.parentNode.appendChild(elem_pink);

  elem_orange.textContent = "ORANGE";
  elem_red.parentNode.insertBefore(elem_orange, elem_red);

  elem_red.innerHTML = '<li style="background-color: red;"> RED </li>';
})());
```

その他の要素プロパティ

createElement(), **textContent**, parentNode,
appendChild(), insertBefore(), innerHTML

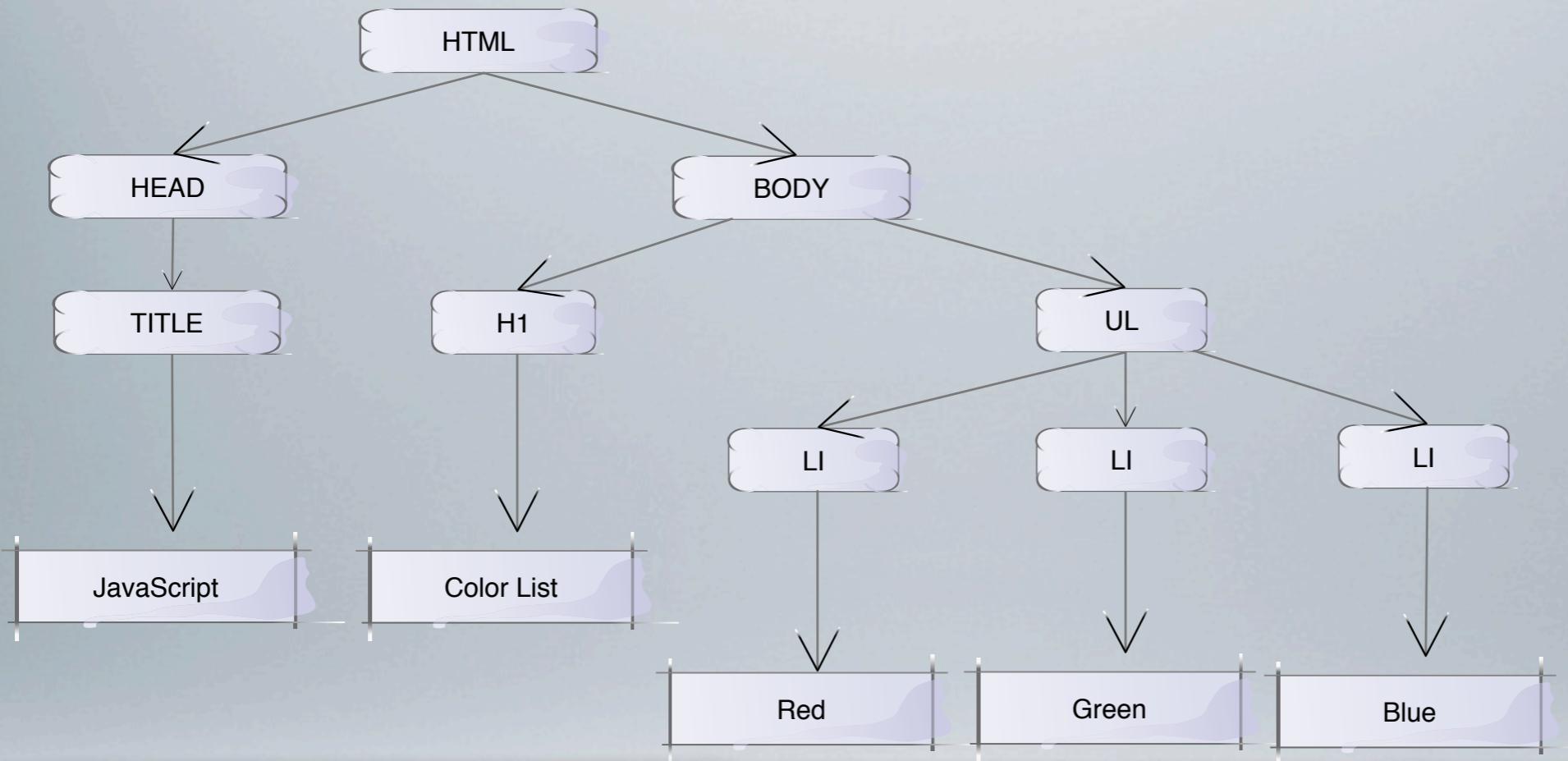
```
(function () {
  var elem_red = document.querySelector("li"),
      elem_pink = document.createElement("li"),
      elem_orange = document.createElement("li");

  elem_pink.textContent = "PINK";
  elem_red.parentNode.appendChild(elem_pink);

  elem_orange.textContent = "ORANGE";
  elem_red.parentNode.insertBefore(elem_orange, elem_red);

  elem_red.innerHTML = '<li style="background-color: red;"> RED </li>';
})());
```

その他の要素プロパティ

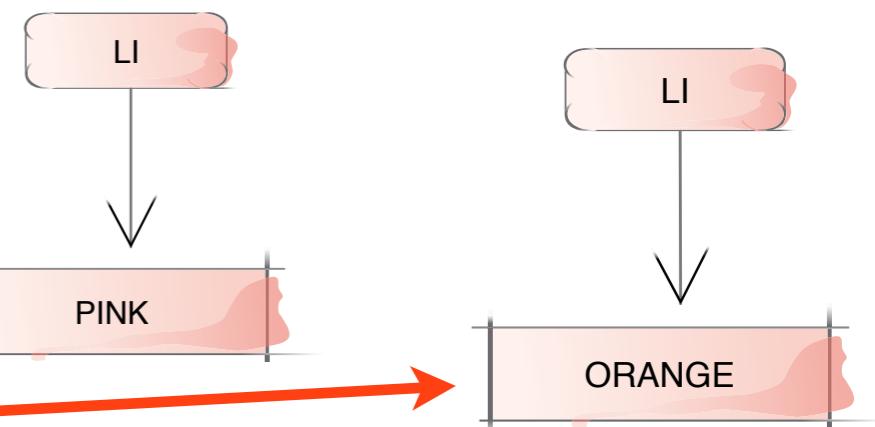


```
(function () {
  var elem_red = document.querySelector("li"),
      elem_pink = document.createElement("li"),
      elem_orange = document.createElement("li");

  elem_pink.textContent = "PINK"; elem_red.parentNode.appendChild(elem_pink);

  elem_orange.textContent = "ORANGE"; elem_red.parentNode.insertBefore(elem_orange, elem_red);

  elem_red.innerHTML = '<li style="background-color: red;"> RED </li>';
})();
```



その他の要素プロパティ

createElement(), textContent, parentNode,
appendChild(), insertBefore(), innerHTML

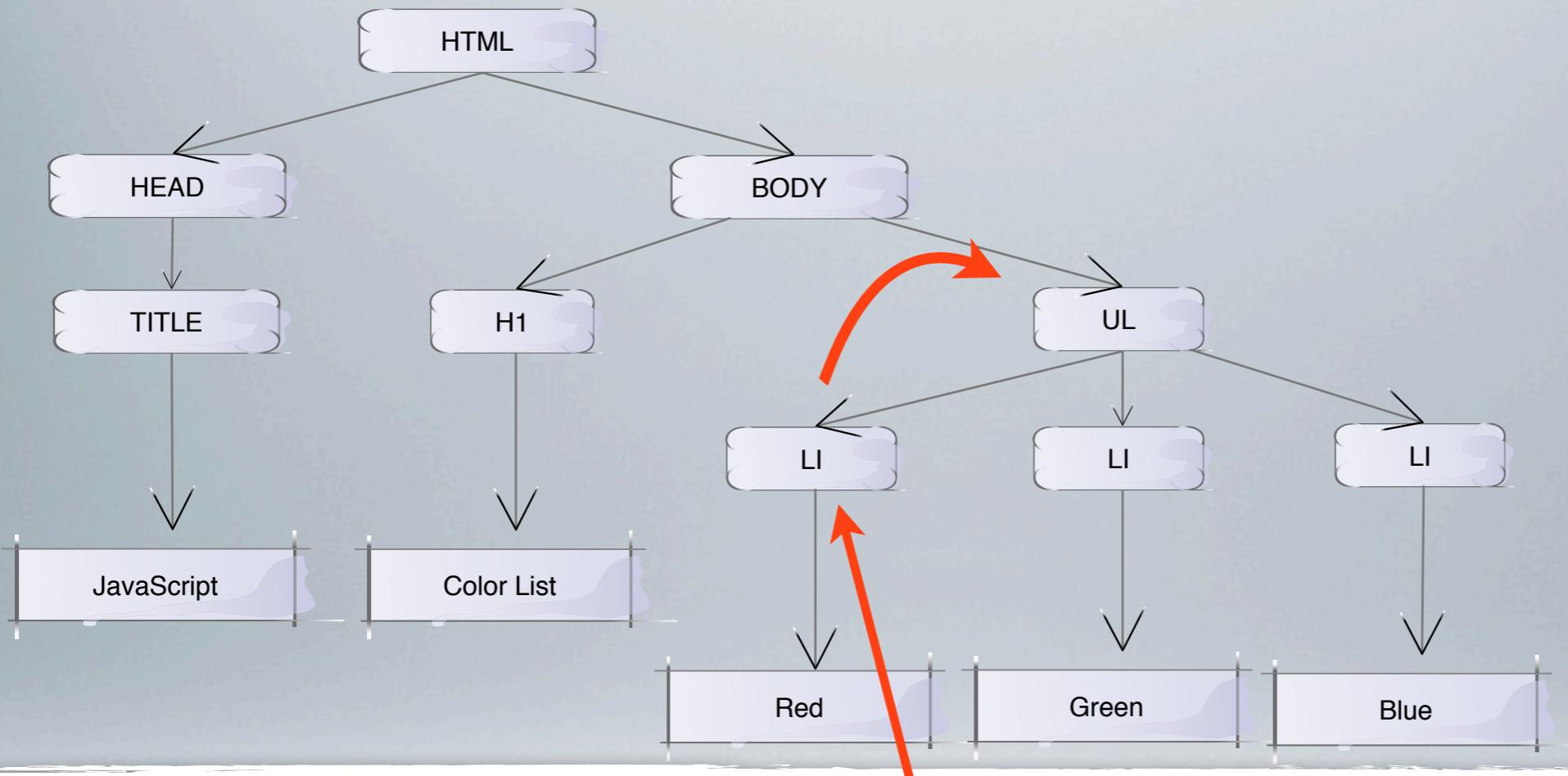
```
(function () {
  var elem_red = document.querySelector("li"),
      elem_pink = document.createElement("li"),
      elem_orange = document.createElement("li");

  elem_pink.textContent = "PINK";
  elem_red.parentNode.appendChild(elem_pink);

  elem_orange.textContent = "ORANGE";
  elem_red.parentNode.insertBefore(elem_orange, elem_red);

  elem_red.innerHTML = '<li style="background-color: red;"> RED </li>';
})());
```

その他の要素プロパティ

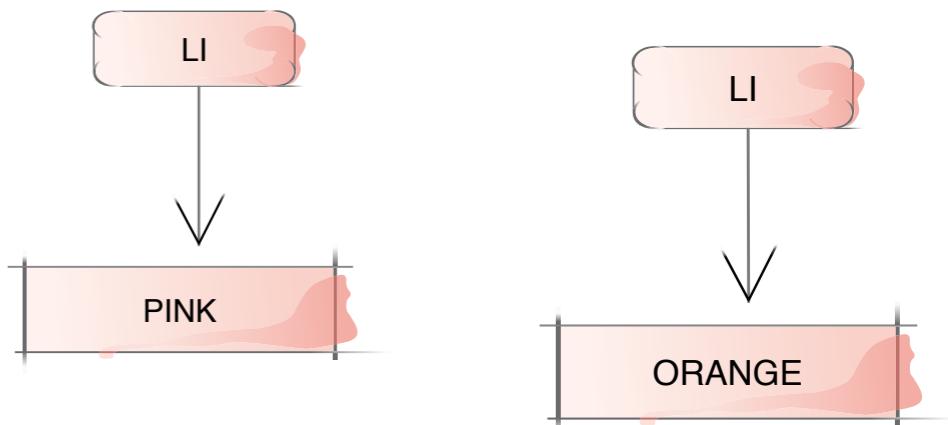


```
(function () {
    var elem_red = document.querySelector("li"),
        elem_pink = document.createElement("li"),
        elem_orange = document.createElement("li");

    elem_pink.textContent = "PINK";
    elem_red.parentNode.appendChild(elem_pink);

    elem_orange.textContent = "ORANGE";
    elem_red.parentNode.insertBefore(elem_orange, elem_red);

    elem_red.innerHTML = '<li style="background-color: red;"> RED </li>';
})());
```



その他の要素プロパティ

createElement(), textContent, parentNode,
appendChild(), insertBefore(), innerHTML

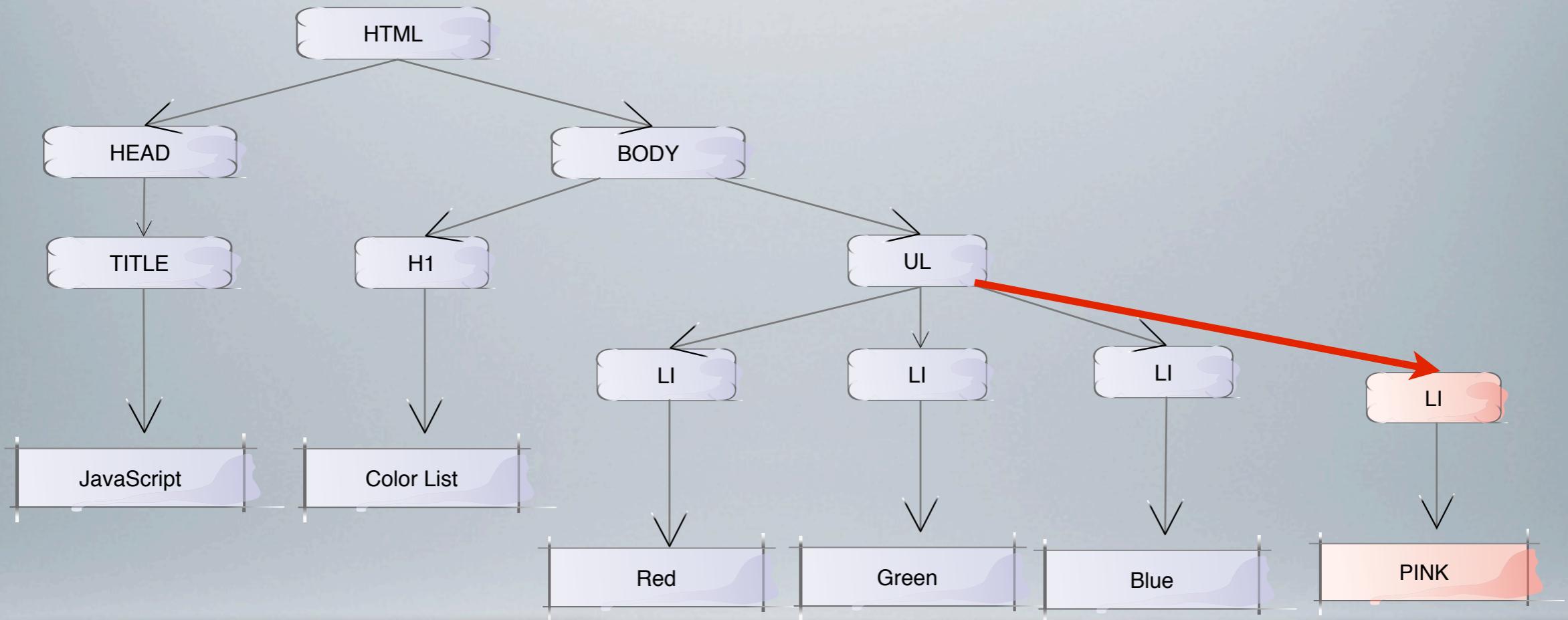
```
(function () {
  var elem_red = document.querySelector("li"),
      elem_pink = document.createElement("li"),
      elem_orange = document.createElement("li");

  elem_pink.textContent = "PINK";
  elem_red.parentNode.appendChild(elem_pink);

  elem_orange.textContent = "ORANGE";
  elem_red.parentNode.insertBefore(elem_orange, elem_red);

  elem_red.innerHTML = '<li style="background-color: red;"> RED </li>';
})());
```

その他の要素プロパティ

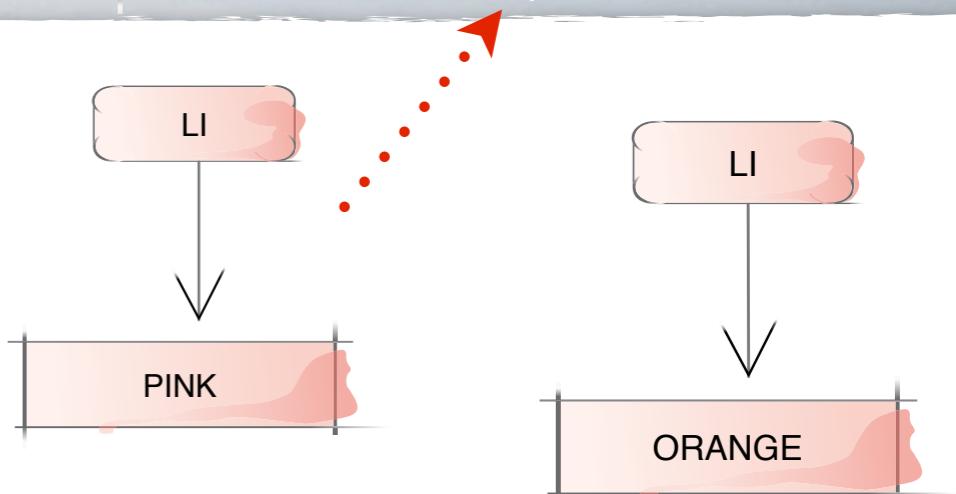


```
(function () {
  var elem_red = document.querySelector("li"),
      elem_pink = document.createElement("li"),
      elem_orange = document.createElement("li");

  elem_pink.textContent = "PINK";
  elem_red.parentNode.appendChild(elem_pink);

  elem_orange.textContent = "ORANGE";
  elem_red.parentNode.insertBefore(elem_orange, elem_red);

  elem_red.innerHTML = '<li style="background-color: red;"> RED </li>';
})());
```



その他の要素プロパティ

createElement(), textContent, parentNode,
appendChild(), insertBefore(), innerHTML

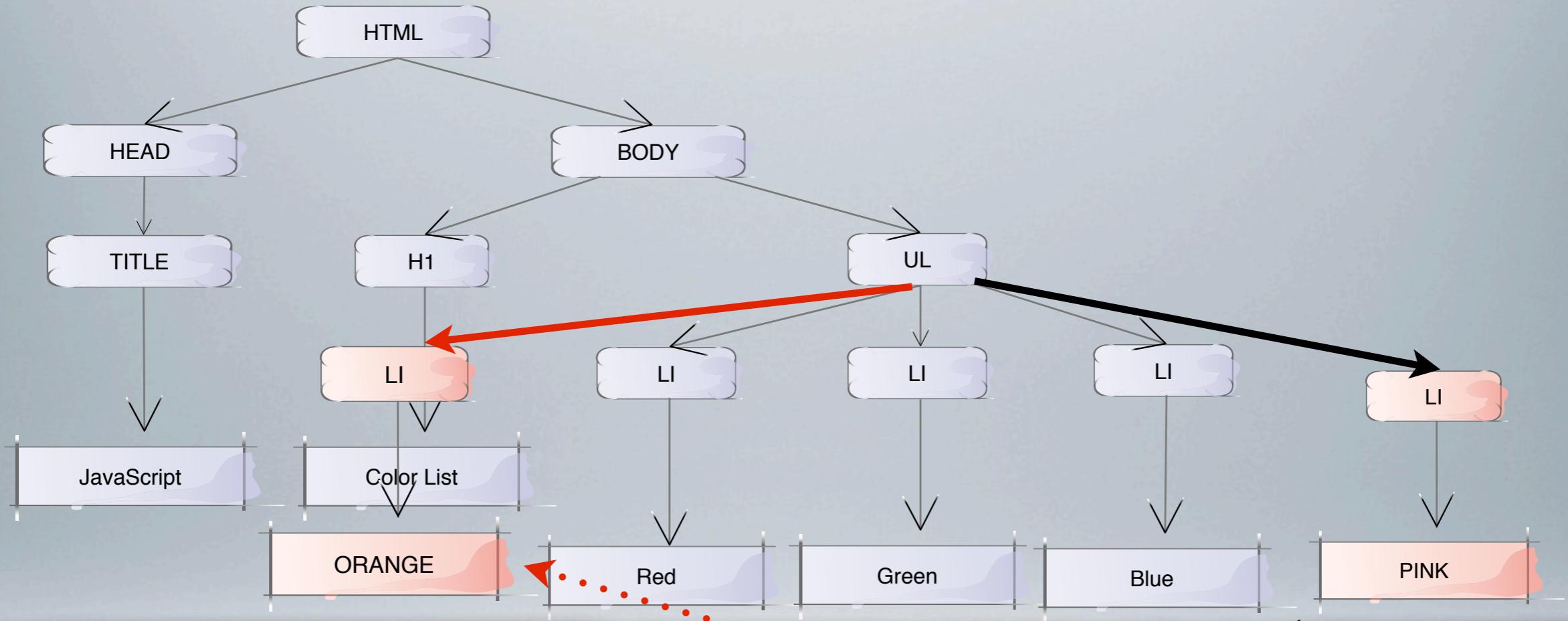
```
(function () {
  var elem_red = document.querySelector("li"),
      elem_pink = document.createElement("li"),
      elem_orange = document.createElement("li");

  elem_pink.textContent = "PINK";
  elem_red.parentNode.appendChild(elem_pink);

  elem_orange.textContent = "ORANGE";
  elem_red.parentNode.insertBefore(elem_orange, elem_red);

  elem_red.innerHTML = '<li style="background-color: red;"> RED </li>';
})());
```

その他の要素プロパティ

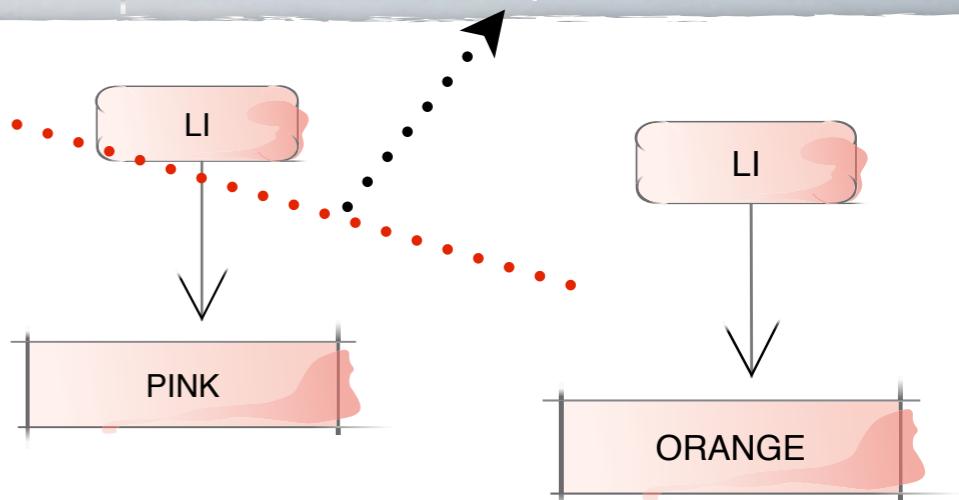


```
(function () {
  var elem_red = document.querySelector("li"),
      elem_pink = document.createElement("li"),
      elem_orange = document.createElement("li");

  elem_pink.textContent = "PINK";
  elem_red.parentNode.appendChild(elem_pink);

  elem_orange.textContent = "ORANGE";
  elem_red.parentNode.insertBefore(elem_orange, elem_red);

  elem_red.innerHTML = '<li style="background-color: red;"> RED </li>';
})());
```



その他の要素プロパティ

createElement(), textContent, parentNode,
appendChild(), insertBefore(), innerHTML

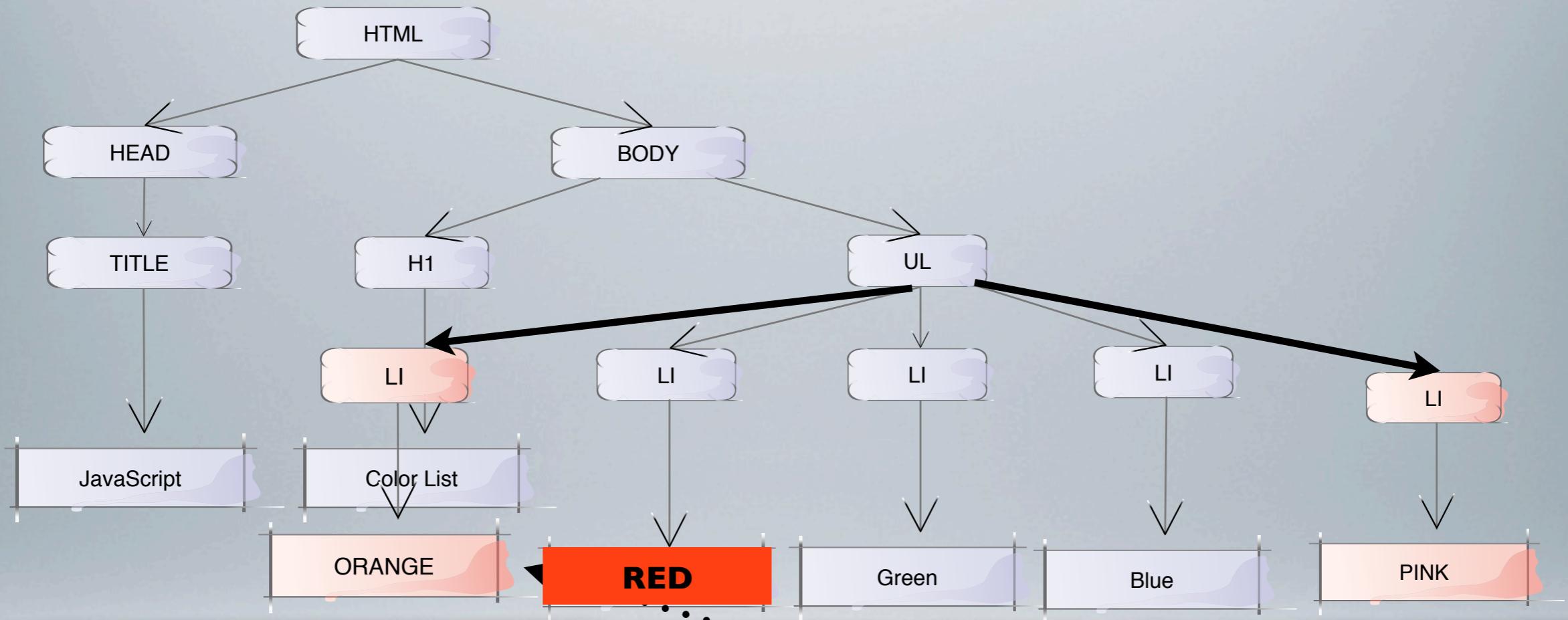
```
(function () {
  var elem_red = document.querySelector("li"),
      elem_pink = document.createElement("li"),
      elem_orange = document.createElement("li");

  elem_pink.textContent = "PINK";
  elem_red.parentNode.appendChild(elem_pink);

  elem_orange.textContent = "ORANGE";
  elem_red.parentNode.insertBefore(elem_orange, elem_red);

  elem_red.innerHTML = '<li style="background-color: red;"> RED </li>';
})());
```

その他の要素プロパティ

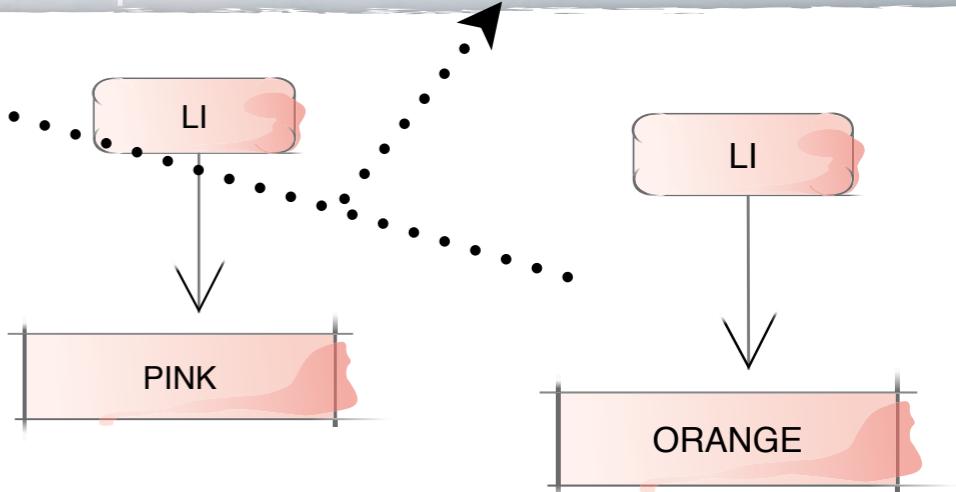


```
(function () {
  var elem_red = document.querySelector("li"),
      elem_pink = document.createElement("li"),
      elem_orange = document.createElement("li");

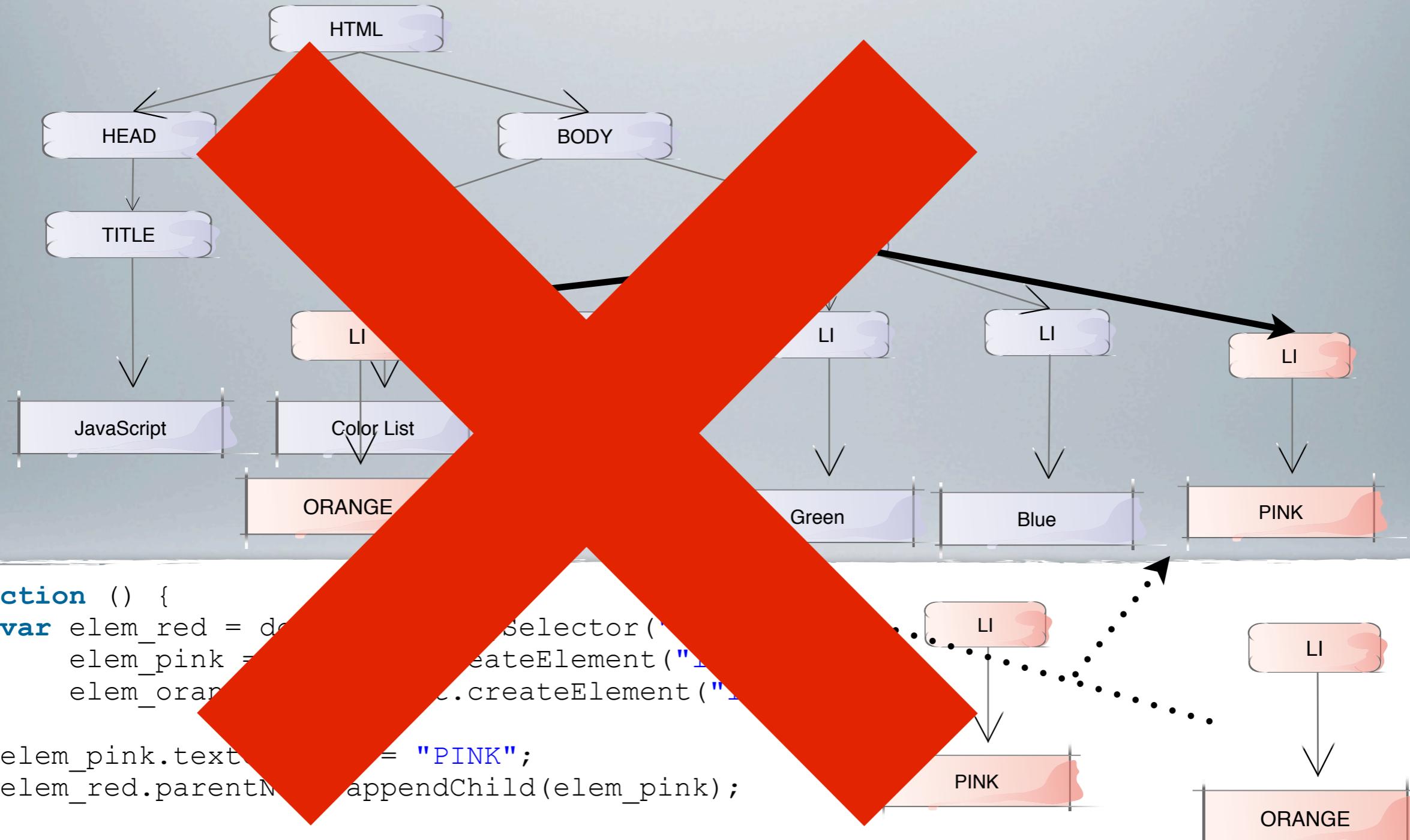
  elem_pink.textContent = "PINK";
  elem_red.parentNode.appendChild(elem_pink);

  elem_orange.textContent = "ORANGE";
  elem_red.parentNode.insertBefore(elem_orange, elem_red);

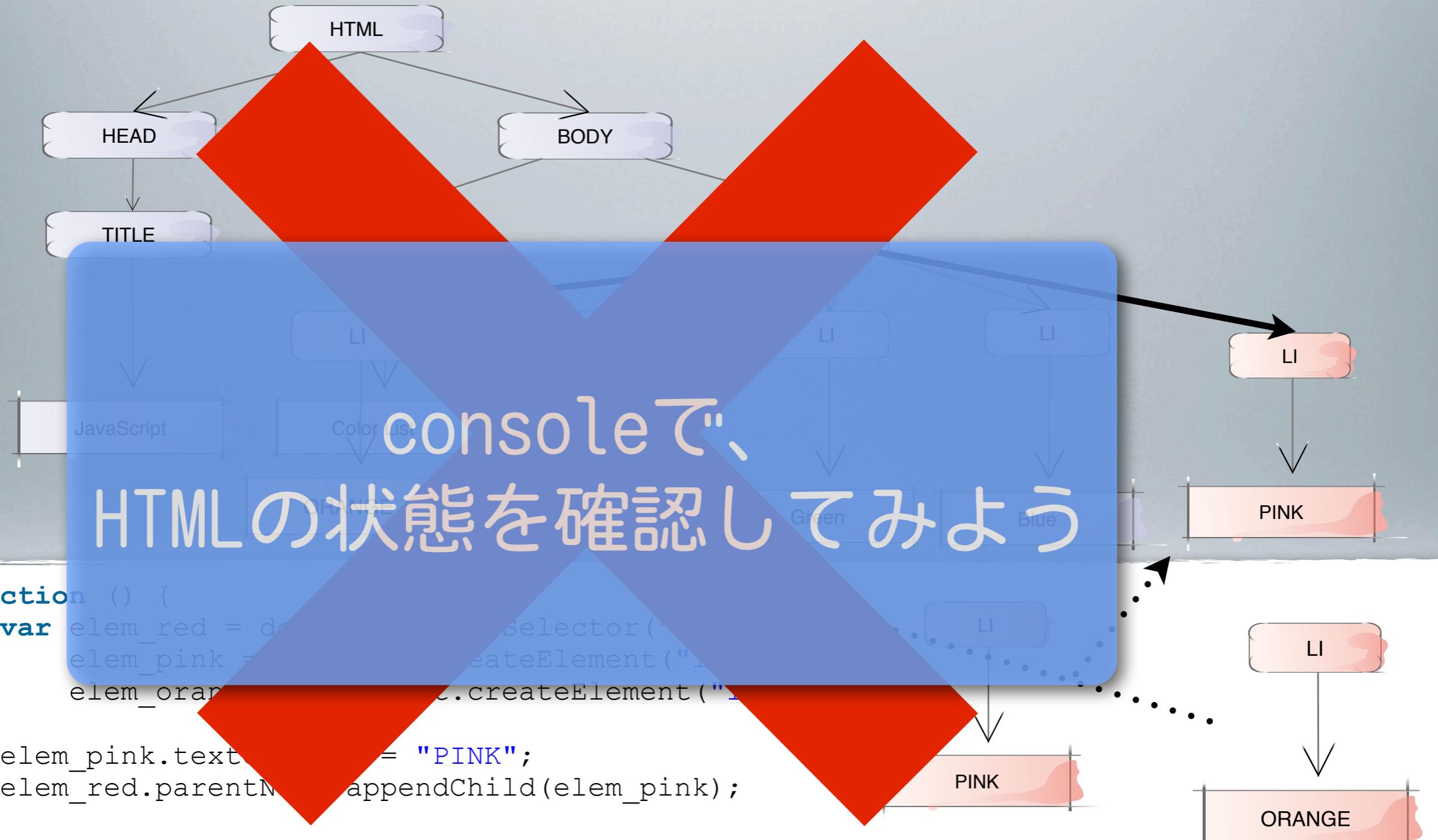
  elem_red.innerHTML = '<li style="background-color: red;"> RED </li>';
})();
```



その他の要素プロパティ

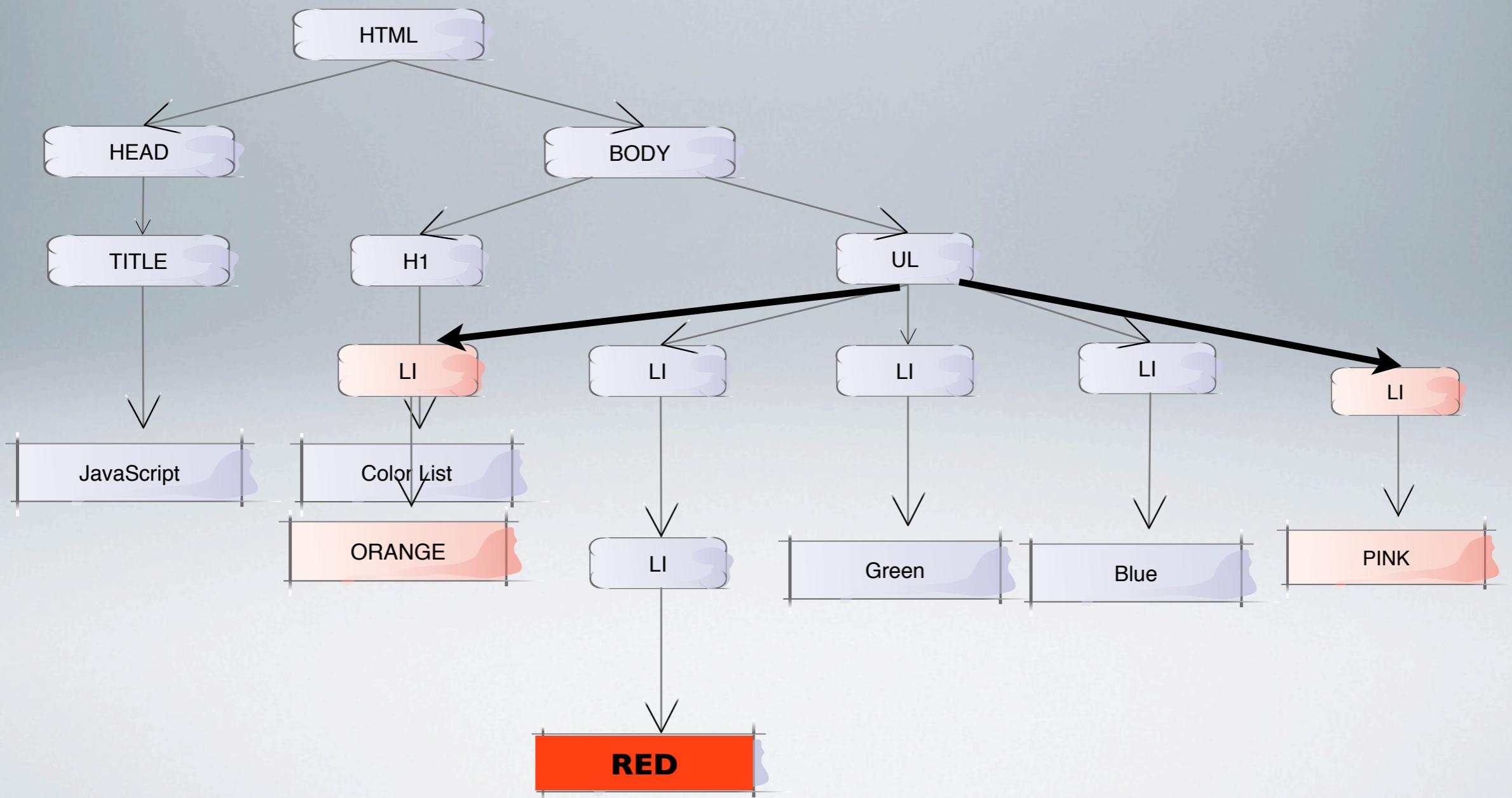


その他の要素プロパティ

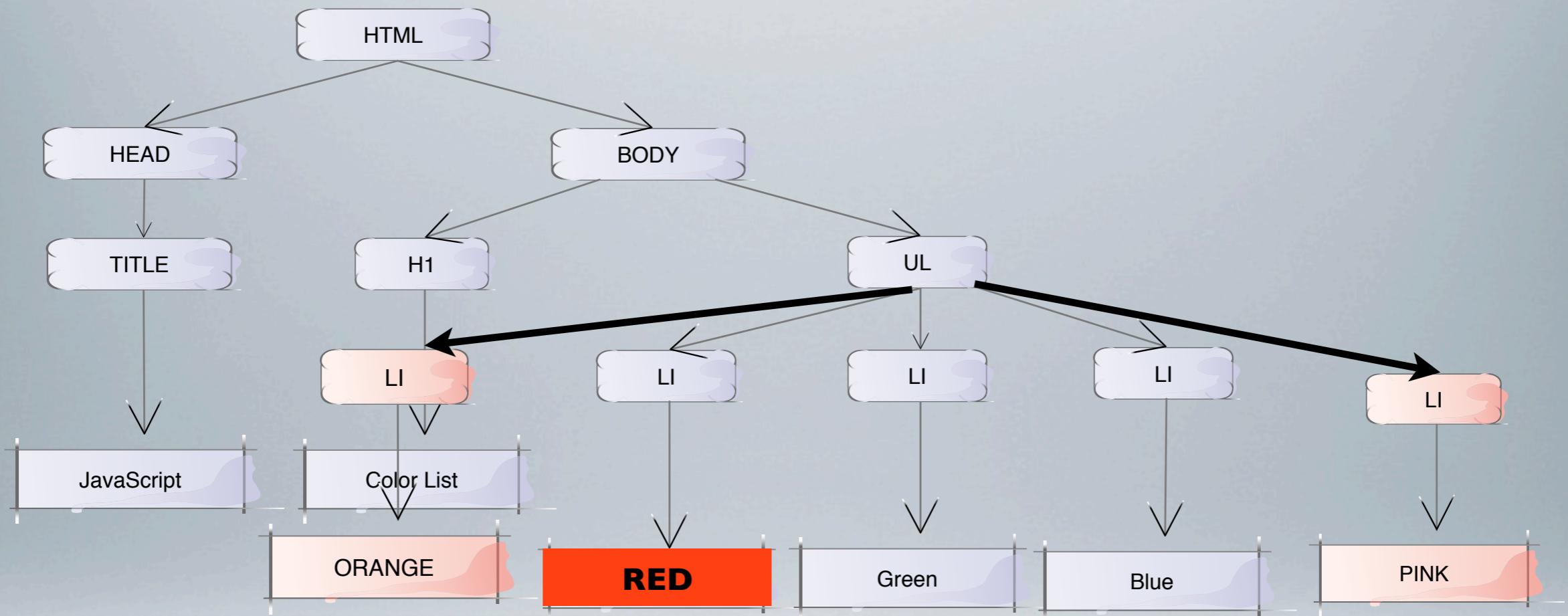


その他の要素プロパティ

こうなってますよね？



こっちが正解



```
(function () {
  var elem_red = document.querySelector("li"),
      elem_pink = document.createElement("li"),
      elem_orange = document.createElement("li");

  elem_pink.textContent = "PINK";
  elem_red.parentNode.appendChild(elem_pink);

  elem_orange.textContent = "ORANGE";
  elem_red.parentNode.insertBefore(elem_orange, elem_red);

  elem_red.outerHTML = '<li style="background-color: red;"> RED </li>';
})());
```

要素のプロパティ lesson

[fifth_lesson/1_1.html]

1. h1要素のテキストを変えてみよう

2. li要素の背景色を変えてみよう

3. li要素を作つてul要素の下に追加する

4. for文でli要素を10個追加する

[fifth_lesson/checker.html]

5. for文で要素を追加しながら白黒のチェックフラッグを作成する

※ CSSは、チェックフラッグがtable要素で作成されることを想定して書いています。li要素等で作成する方はお手数ですが、各自よしなにお願いします。

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("hello");  
}
```

「クリック」というイベントが発生した時に、
「hello」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("hello");  
}
```

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

Event lesson

[fifth_lesson/clickEvent.html]

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？

fifth_lesson/clickEvent.jsに書いてみよう！

```
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 () {
    alert("first event."); // -> 実行される
}, false);

first.addEventListener("click", function () {
    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”, 処理, false);

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

first.addEventListener("click", function () {
    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

[fifth_lesson/addText.html]

input要素に何か文字列が入れられ、
その状態でAddボタン要素をクリックしたら、
h2要素の下にinput要素内の文字列が追加されるように
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);
```