**JavaScript**

**HTMLでJavaScriptを読み込む**

<script type="text/javascript" src="js/sample.js"></script>

**<script></script>の記述場所**

①<head>タグ内に記述

②</body>タグ直前に記述

**JavaScriptの書き始め**

①即時関数

(function() {

// 処理

}());

jsファイルが読み込まれた瞬間に実行

全体を()で括るのはグローバル変数の汚染を防ぐため

②DOMContentLoaded

window.addEventListener('DOMContentLoaded', function() {

// 処理

});

HTMLの読み込みが完了したら実行

③load

window.addEventListener('load', function() {

// 処理

});

cssファイルや画像など全てのリソースの読み込みが完了したら実行

**JavaScriptとは**

|  |  |
| --- | --- |
| **ECMAScript** | どの実行環境でも共通な動作のみが定義されている |
| **ECMAScript 2015（ES2015）** | 大きくアップデートされた仕様 |
| **"use strict";** | strict mode  文の先頭に書く |

**変数と宣言**

|  |  |
| --- | --- |
| **const** | 値の再代入不可能な変数を宣言 |
| **let** | 値の再代入可能な変数を宣言 |

**値の評価と表示**

|  |  |
| --- | --- |
| **REPL** | Webコンソールにある機能 |
| 入力された値を評価し表示する |
| consoleの変数をクリア | F5でリロード |

**Scriptファイルを保存するときの文字コードと改行コード**

|  |  |
| --- | --- |
| 文字コード | **UTF-8** |
| 改行コード | **LF** |

|  |  |
| --- | --- |
| HTMLでJavascript読み込むタグ | **<script src=""></script>** |

**エラーついて**

|  |  |
| --- | --- |
| SyntaxError: missing ) after argument list[詳細] index.js:1:13 | ・)がない。  ・1行目13列目  ・行は1からカウント  ・列は0からカウント |

**データ型**

**・変数の型**はない。**値の型**はある

**プリミティブ型**

|  |  |
| --- | --- |
| **Boolean** | true、false |
| **Number** | 数値 |
| **String** | 文字列 |
| **undefined** | 未定義 |
| **null** | 値がない |
| **Symbol** | ES2015から追加された一意で不変な値のデータ型 |

**文字列リテラル**

|  |  |
| --- | --- |
| **'** |  |
| **"** |  |
| **`** |  |
| **改行ができる** |
| **${x}で変数を埋め込める** |

**オブジェクト**

|  |  |
| --- | --- |
| **リテラル** | **{}** |

|  |  |  |
| --- | --- | --- |
| **プロパティへのアクセス** | **ドット記法** | **object.key** |
| **ブラケット記法** | **object["key"]** |

**配列**

|  |  |
| --- | --- |
| **リテラル** | **[]** |
| **要素へのアクセス** | **array[index]** |

**正規表現**

|  |  |
| --- | --- |
| **リテラル** | **//** |
| **/で正規表現を囲む**  **例)数字**  **/\d/** |

|  |  |
| --- | --- |
| **===** | **厳密等価演算子** |
| **!==** | **厳密不等価演算子** |
| **==** | **等価演算子** |
| **!=** | **不等価演算子** |

・==では**暗黙的な型変換**を行ってから比較する。

console.log(1 == "1"); // => true

**型変換**

|  |  |  |
| --- | --- | --- |
| **Boolean()** | **true、false**に変換 | Boolean("string"); // => true  Boolean(0); // => false |

|  |  |
| --- | --- |
| **falsyな値** | false  undefined  null  0  NaN  "" |

|  |  |  |
| --- | --- | --- |
| **String()** | 文字列に変換 | String("str"); // => "str"  String(true); // => "true"  String(null); // => "null" |

※**プリミティブ**型のみ有効

|  |  |  |
| --- | --- | --- |
| **Number()** | 数値に変換 | Number("1"); // => 1 |

|  |  |  |
| --- | --- | --- |
| NaN | Not a Number |  |

|  |  |
| --- | --- |
| NaNかどうか判定 | Number.isNaN(x) |

**関数**

|  |  |
| --- | --- |
| **関数宣言** | **Function**キーワードを使う |
| **静**的 |
| **関数式** | **変数**に**匿名関数**を代入する |
| **動**的 |

|  |  |
| --- | --- |
| **arguments** | **関数ブロック**でのみ利用可能 |
| 与えられた全ての引数を保持する |
| **arrayライク**なオブジェクト |

function func1(a, b, c) {

console.log(arguments[0]); // expected output: 1

console.log(arguments[1]); // expected output: 2

console.log(arguments[2]); // expected output: 3

}

func1(1, 2, 3);

・returnが省略された場合**undefined**を返す

・javaScriptは引数の個数をチェックしない

・varがない変数は全てグローバル変数となる

**ES2015**

|  |  |
| --- | --- |
| **アロー関数** | **変数**に**匿名関数**を代入する |
| **動**的 |

const materials = [

'Hydrogen',

'Helium',

'Lithium',

'Beryllium'

];

console.log(materials.map(material => material.length));

// expected output: Array [8, 6, 7, 9]

|  |  |
| --- | --- |
| **可変長引数** | **...**を仮引数の前につける |
| numsは完璧な**配列** |

|  |  |
| --- | --- |
| **引数のデフォルト値** | **仮引数=x**でできる |

|  |  |
| --- | --- |
| **コールバック関数** | 引数として渡される関数 |
| **高階関数** | 引数がコールバック関数な関数 |

例）

setTimeout(function(){

console.log("Hello")

}, 2000);

**ビルトインオブジェクト**

**Array**

|  |  |
| --- | --- |
| **length** | 配列の長さ |
| **Array.isArray(object)** | 配列かどうか判定 |
| **push()** | 配列の末尾に要素を追加 |
| **pop()** | 配列の末尾の要素を取得 |
| **shift()** | 配列の先頭に要素を追加 |
| **unshift()** | 配列の先頭の要素を取得 |
| **sort()** | 配列を昇順にソート |
| **reverse()** | 配列をリバースする |
| **indexOf()** | 要素のインデックスを |
| **concat()** | 配列を取り込む |
| **join()** | 配列の要素をdelimiterで区切った**文字列**に変換  delimiterを空文字にすると区切られず連結される |
| **Array.of()** | 引数に渡された値から配列を生成 |
| **includes()** | 引数に渡された値を含むかどうか判定 |

|  |  |
| --- | --- |
| **forEach()** | 要素を順に処理 |
| **map()** | 要素を順に加工 |
| **filter()** | 要素をフィルタリング |

・**\***が付いているメソッドは元の配列の値が変わる。

・ついていないメソッドは元の配列の値に影響しない。

Array.of(7); // [7]

Array.of(1, 2, 3); // [1, 2, 3]

Array.of(undefined); // [undefined]

Array(7); // array of 7 empty slots 7つの空のスロットの配列の生成

Array(1, 2, 3); // [1, 2, 3]

const letters = ['a', 'b', 'c'];

const numbers = [1, 2, 3];

letters.concat(numbers); // result in ['a', 'b', 'c', 1, 2, 3]

const beasts = ['ant', 'bison', 'camel', 'duck', 'bison'];

console.log(beasts.indexOf('bison')); // expected output: 1

console.log(beasts.indexOf('bison', 2)); // start from index 2expected output: 4

console.log(beasts.indexOf('giraffe')); // expected output: -1

const array1 = [1, 2, 3];

console.log(array1.includes(2)); // expected output: true

const pets = ['cat', 'dog', 'bat'];

console.log(pets.includes('cat')); // expected output: true

console.log(pets.includes('at')); // expected output: false

var a = ['風', '水', '火'];

a.join(); // '風,水,火'

a.join(', '); // '風, 水, 火'

a.join(' + '); // '風 + 水 + 火'

a.join(''); // '風水火'

const items = ['item1', 'item2', 'item3'];

const copy = [];

for (let i=0; i<items.length; i++) {

copy.push(items[i]);

}

items.forEach(function(item){

copy.push(item);

});

**オブジェクトのプロパティの列挙**

|  |  |
| --- | --- |
| **Object.keys(obj)** | ・objectの列挙可能なプロパティを取得 |
| **Object.getOwnPropertyNames(obj)** | ・objectの列挙可能、列挙不可能なプロパティを取得 |
| **for(prop in obj)** | ・objectの列挙可能なプロパティを取得 |
| ・プロトタイプチェーンを辿って取得 |

・取得できるのはプロパティのみ。Functionは取得できない



**オブジェクト**

・オブジェクトのKeyには**String**か**Symbol**のみ利用できる

・オブジェクトのKeyは **"** や **'** を省略できる

・オブジェクトのValueには任意の値や**変数**などが利用できる

・ドット記法でKeyを**変数**としてアクセスする

・ブラケット記法ではKeyを**文字列**としてアクセスする

|  |  |
| --- | --- |
| **delete** | オブジェクトのプロパティを削除する |
| **instanceof** | オブジェクトの型を判定 |

・どちらも演算子

・instanceofはプリミティブ型か参照型かどうかの判定のみにしか使えない

**オブジェクトにプロパティがあるか判定**

|  |  |
| --- | --- |
| **in 演算子** | ・オブジェクトにプロパティがあるか判定  ・プロトタイプチェーンを辿る |
| **object.hasOwnProperty(prop)** | ・オブジェクトにプロパティがあるか判定  ・プロトタイプチェーンを辿らない |

**String**

|  |  |
| --- | --- |
| split | 配列に変換 |
| 引数なし | そのまま配列に変換 |
| 引数が空文字 | 1文字ずつ区切られる |
| 引数がカンマ | カンマで区切られる |

var str = 'あいうえお';

str.split(); // ["あいうえお"]

str.split("") // ["あ","い","う","え","お"]

**プロトタイプ指向**

**コンストラクタ**

・function Member(){}

・var Member = function(){};

先頭文字が大文字の関数

**インスタンス化**

|  |
| --- |
| var mem = new Member(); |

newすると以下2つがコンストラクタに加えられる

・var this = {};

・return this;

**プロパティ・メソッド・静的プロパティ・静的メソッドの定義先**

|  |  |
| --- | --- |
| プロパティの定義 | **コンストラクタ** |
| メソッドの定義 | **プロトタイプ** |
| 静的プロパティ・静的メソッドの定義 | **コンストラクタ** |

プロトタイプにメソッドを記述することで参照のみ持たせる

// コンストラクタ

var Member = function(firstName, lastName){

this.firstName = firstName;

this.lastName = lastName;

};

// プロトタイプ

Member.prototype = {

getName: function(){

},

toString: function(){

}

};

// 静的プロパティ・静的メソッド

Member.charge = '10000yen';

Member.showVersion = function(){

console.log("ver 1.0");

};

・ valueが関数のプロパティがメソッド

・プロトタイプはオブジェクトリテラルで記述するのが良い

・同一のコンストラクタからインスタンス化されたものでもプロパティが同じとは限らない

・インスタンス生成後でもprototypeにプロパティを記述すれば参照できる

・インスタンスプロパティはプロトタイププロパティを隠蔽する

・インスタンスから静的プロパティ・メソッドは呼び出せない

・オブジェクトを生成するとprototype、\_\_proto\_\_ が自動で生成される

**クラス指向**

class Player {

// コンストラクタ

constructor(name, score) {

this.name = name;

this.score = score;

}

// メソッド

showInfo() {

console.log(`name: ${this.name}, score: ${this.score}`);

}

// 静的メソッド

static showVersion() {

console.log("ver 1.0");

}

}

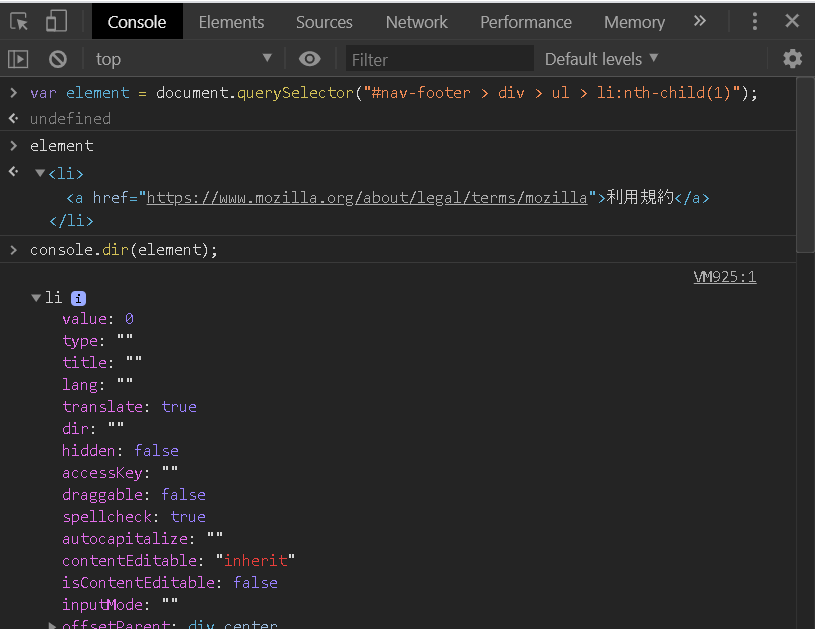
・class直下で let name; のようなインスタンスフィールドは定義できない

・継承の方法はjavaと同じ

**DOM**

**HTMLオブジェクトのプロパティを表示**

console.dir(element)



**eventオブジェクト**

|  |  |
| --- | --- |
| target | **イベントが発生した要素** |
| currentTarget | **イベントリスナがバインドされた要素** |
| type | **イベントのタイプ** |

<ul>

<li>リスト1</li>

<li>リスト2</li>

</ul>

<script>

var ul = document.querySelector('ul');

ul.addEventListener('click', function(event){

console.log("currentTarget: " + event.currentTarget);

console.log("target.innerHtml: " + event.target.innerHTML);

})

</script>

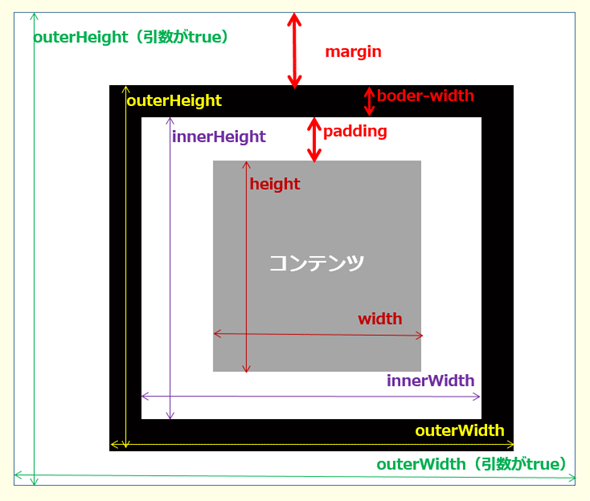
**画面のサイズ**

|  |  |
| --- | --- |
| screen.width | **ディスプレイの幅** |
| screen.height | **ディスプレイの高さ** |
| window.innerWidth | **ブラウザ内の横の表示域(スクロールバーを含む)** |
| window.innerHeight | **ブラウザ内の縦の表示域(スクロールバーを含む)** |

**jQueryで要素のサイズを取得**

|  |  |
| --- | --- |
| width() | **コンテンツの幅** |
| innerWidth() | **コンテンツの幅 + padding** |
| outerWidth() | **コンテンツの幅 + padding + border** |
| outerWidth(true) | **コンテンツの幅 + padding + border + margin** |

cssのbox-sizingは関係ない

****

**window.open**

<https://qiita.com/yun_bow/items/356f21fc376133037d84>

<https://web-designer.cman.jp/javascript_ref/window/open/>

**this**

|  |  |
| --- | --- |
| **グローバルコンテキスト** | グローバルオブジェクト |
| **関数** | グローバルオブジェクト |
| **メソッド** | メソッドを呼び出したオブジェクト |
| **コンストラクタ** | 生成されたオブジェクト |
| **イベントハンドラ** | イベントハンドラがattachされた要素 |
| **インラインイベントハンドラ** | イベントハンドラがattachされた要素 |

console.log(this === window); // true

a = 37;

console.log(window.a); // 37

var obj = {a: 'Custom'};

var a = 'Global';

function whatsThis() {

return this.a;

}

whatsThis(); // 'Global'

whatsThis.call(obj); // 'Custom'

whatsThis.apply(obj); // 'Custom'

var o = {

prop: 37,

f: function() {

return this.prop;

}

};

console.log(o.f()); // 37

var Member = function(firstName, lastName){

this.firstName = firstName;

this.lastName = lastName;

};

var obj = new Member('TORU','IIDA');

console.log(obj.firstName) // 'TORU'

<ul>

<li>リスト1</li>

<li>リスト2</li>

</ul>

<script>

var ul = document.querySelector('ul');

ul.addEventListener('click', function(){

console.log(this); // <ul>...</ul>

console.log(this === event.currentTarget); // true

})

</script>

<button onclick="alert(this.tagName.toLowerCase());">Show this</button>

**jQueryの$(this)とthis**

<ul>

<li>リスト1</li>

<li>リスト2</li>

</ul>

<h2>居住地域</h2>

<ol></ol>

<script>

$('li').click(function(){

console.log($(this).text());

});

array = ["関東", "関西", "それ以外のエリア"];

$.each(array, function() {

$('ol').append("<li>" + this + "</li>");

});

</script>

**アロー関数のthis**

param = 'global param';

function printParam1(){

console.log(this.param);

}

let printParam2 = () => {

console.log(this.param);

}

let object1 = {

param: 'object1 param',

func: printParam1

}

let object2 = {

param: 'object2 param',

func: printParam2

}

object1.func(); // 'object1 param'

object2.func(); // 'global param'

・アロー関数のthisは宣言時に参照が決まってる

・アロー関数以外のthisは実行時に参照が決まる

**css**

**user-select**

**letter-spacing**

**疑似クラス**