**JS基礎知識**

ブラウザ側で処理を行うための言語

　　Alert(‘Hello’)　⇒　ポップアップメッセージが表示される。

　　もともとブラウザが持っている機能を利用してメッセージを表示している。

【ファイル内にjavascriptを記述する】

    <script>

        alert("Hello")

    </script>

【外部ファイルとしてjsファイルを呼出する】

    <script src="main.js"></script>

※bodyタグの最後で読み込みを行う。

　DOMでセレクター参照を行った際に、最初に定義するとセレクター参照に失敗するから。

【デバッグ文の出力】

console.log("Hello")

consoleオブジェクトのlogメソッドを利用して、ブラウザ検証機能のコンソールに対して、引数の値を表示することができる。

※重要　コンソールの値を確認しながら処理を実装していくこととなる。

【変数、定数】

変数はlet、定数はconst

let let\_name = 'Tom';

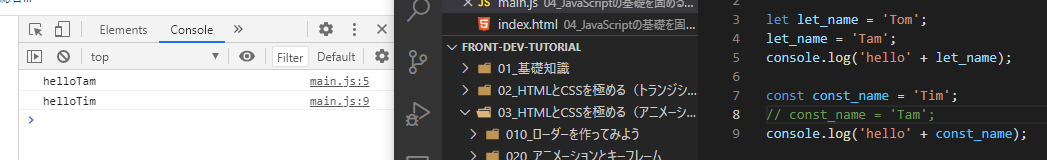
let\_name = 'Tam';

console.log('hello' + let\_name);

const const\_name = 'Tim';

const\_name = 'Tam'; //エラーになる。

console.log('hello' + const\_name);



【型について】

型指定をして変数を定義しない。

格納された値により変数の型が決定する。

1. 文字列を格納　⇒　２．数値を格納　の場合は、１時点では文字列型、２時点では数値型となる。

・　変数の現在型を確認　（typeof）

let variable = 'str';

variable = 12;

variable = true;

console.log(typeof variable);

※（コラム）TypeScriptはJavascriptに対して、型指定を厳密として拡張した言語

　　各種のフレームワークにおいては、TypeScriptが利用されていることが多い。

【関数定義】

Function 関数名(引数)で定義

Return で戻り値を指定

function hello (name) {

  console.log('hello' + name);

  return name ;

}

hello('code mafia');

const result = hello('code mafia2');

console.log(result);

無名関数

Function(引数){ 手続き }

関数のまとまりを変数として定義することができる。

const mumei = function (name) {

  console.log('hello' + name);

  return name ;

}

const result2 = mumei('code mafia3');

console.log(result2);

【配列】

const arry = new Array(1,2,3,4,5,6,'moji',false);

console.log(arry);

jsは複数の型が混ざっていても配列に格納することができる。

// 最初に追加

arry.unshift('first');

// 最後に追加

arry.push('last');

リストへ要素を追加

// 最初の要素を配列から削除　戻り値として取り出し

Let first = arry.shift();

// 最後の要素を配列から削除　戻り値として取り出し

Let last = arry.pop();

【オブジェクト】

// オブジェクト初期化

const person1 = {};

// オブジェクトのプロパティを設定

const person2 = {

  // プロパティとして配列を定義

  name: ['Code', 'Mafia'],

  age: 32,

  gender: 'male',

  // プロパティとしてオブジェクトを定義

  interests: {

    sports: 'soccer',

    music: 'piano'

  }

};

オブジェクト値を参照する

console.log(person2.interests);

キー情報が動的に変わる場合は変数で指定する

ディクショナリ的な使い方。

const agekey = 'age';

console.log(person2[agekey]);

オブジェクトメソッドの指定

  name: ['Code', 'Mafia'],

  //メソッドを定義 自オブジェクトのプロパティはthisで参照

  getFullName: function(){

    console.log(this.name[0] + this.name[1])

  }

person2.getFullName();

【ループ】

// for文

const arry = [1,2,3,4,5];

for(let i = 0; i < 5; i= i +1 ) {

  console.log(arry[i]);

};

配列に対してより良い記述

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

  console.log(arry[i]);

};

より簡略な反復処理の記載

// (iに配列のインデックスが格納)

const arry = [1,2,3,4,5];

for(let i in arry ) {

  console.log(arry[i]);

};

// for each文みたいなやつ(iに配列値が格納)

const arry2 = [1,2,3,4,5];

for(let i of arry2 ) {

  console.log(i);

};

【オブジェクトを配列で扱う】

const todos = [

  //オブジェクトを格納

  {

    id: 1 ,

    title: 'Go to shopping' ,

    completed: true

  } ,

  {

    id: 2 ,

    title: 'Go to park' ,

    completed: true

  } ,

  {

    id: 3 ,

    title: 'Go to home' ,

    completed: true

  }

]

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

  console.log(i, todos[i].title);

}

【分岐処理】

// trueとみなされる

if(1 == '1') {

  console.log("this is true");

}

// falseとみなされる

if(1 === '1') {

  console.log("this is true");

} else {

  console.log("this is false");

}

== 値のみデータ型は比較しない。

=== 値とデータ型の比較をする。

論理否定

if(1 !== '1') { }

論理積

if(1 !== '1' && 1 === '1') { }

論理和

if(1 !== '1' || 1 === '1') { }

比較演算が設定されていない場合

const num1 = null;

const num2 = undefined;

const num3 = 0; // 0もfalseとみなされる。

if(num1 || num2) {

  console.log("this is true");

}

上記以外の値が設定されている場合はtrueとみなされる。

【関数のデフォルト値】

function hello(name = 'Tom') {

  console.log('hello ' + name);

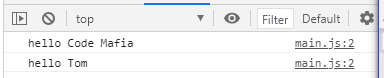
}

hello('Code Mafia');

hello();

引数の定義にデフォルト値を設定することで、

呼出元で引数を渡さずに呼出た場合にデフォルト値が設定される。



【アロー関数】

無名関数を定義する場合に、function句を省略するための記法

//アロー関数 、格納先変数：hello 、引数：name

const hello = name => {

  console.log('hello ' + name);

}

hello('Code Mafia');

1行で記述する場合は｛｝を省略することが出来る。ロジックの結果が戻り値として返却される。

let array = [1,2,3,4,5];

array.forEach(value => console.log(value));

配列の展開が容易に表現できる点も良い。

【コールバック関数】

関数の引数として関数を渡すことが可能。

関数を呼び出すときに()をつけずに、他関数に渡す。他関数で受け取った関数を引数として、()をつけて関数側で実行する仕組みとなる。

function hello(callback) {

  console.log('hello ' + callback());

}

function getname(){

  return 'Code Mafia';

}

hello(getname);

無名関数でもそうだが、変数に関数（オブジェクトとして）を格納することが出来ることがJSの特徴ともいえる。下記の記法が良く使用される。

function hello(callback) {

  console.log('hello ' + callback());

}

  hello(function() {

  return 'Code';

});

アロー関数に直すとすっきりする。

function hello(callback) {

  console.log('hello ' + callback());

}

  hello(() => 'Code');

コールバック関数に引数（lastname）を渡して、呼出元の関数を実行している。

下記のコードでは、アロー関数の引数nameにコールバック関数のlastnameの値が渡されている。Lastnameには、hello呼出時の第二引数Mafiaが渡されているため。

ログに表示されるのは、hello + Code + Mafia

function hello(callback, lastname) {

  console.log('hello ' + callback(lastname));

}

  hello(name => 'Code' + name, 'Mafia');

ForEach文との組み合わせ

const arry = [1, 2, 3, 4, 5];

function forEach(ary, callback) {

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

    callback(ary[i]);

  }

}

function log(val){

  console.log(val);

}

function double(val){

  val = val \* 2;

  log(val);

}

forEach(arry, double);

【配列とforEach】

const arry = [1, 2, 3, 4, 5];

arry.forEach(val => console.log(val))

第一引数：値、第二引数（省略可）：インデックス、第三引数（省略可）：配列自体

【配列とreduce】

// 文字列の加工等に有効

// accuは前回ループの戻り値を受け取る

// currは配列の値をそのままループする

const str = 'animation';

const strArry = str.split('');

const result = strArry.reduce((accu, curr) => {

  return accu + '<' + curr + '>'

}, '')

console.log(result);

reduce関数の第二引数には、初回ループ時にaccuに設定される初期値を設定する。

デフォルトだと配列の第一引数がaccuに格納されてしまうため、1文字目にたいして文字列の加工処理がスキップされることになる。

・・・コールバック関数と組み合わせた記法

const str = 'animation';

const strArry = str.split('');

function tag(accu, curr) {

  return accu + '<' + curr + '>';

}

function reduce(arry, callback, defaultValue) {

    let accu = defaultValue;

    arry.forEach(val => accu = callback(accu ,val));

    return accu;

}

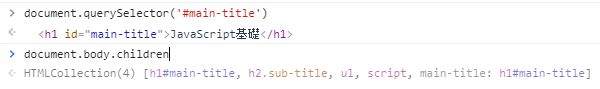
const result = reduce(strArry, tag, "");

console.log(result);

【DOM（Document Object Model）】

HTMLをJavaScriptから操作できるようにしたインターフェイス。

DOM APIを通じてHTML情報を取得・変更・イベント登録が可能になる。



ドキュメントの要素情報に対して参照する。

Childrenプロパティは階層構造で子要素の一覧を取得するため、DOMTREEとも呼ばれる。

・要素の取得　document.querySelector(‘#main-title’)

　　　　　　　　　　　※getElementByIdは古い記法

　　id要素については、#main-title　「#」をつける

　　class要素については、.sub-title　「.」をつける

　　タグ要素については、h1　「」何もつけない。

・要素内の値を取得

const h1 = document.querySelector("#main-title")

h1.innerHTML

⇒"JavaScript基礎"

・クラスの追加

　　h1.classList.add(‘underline’)

・クラスの削除

　　h1.classList.remove(‘underline’)

・リストに対する処理

　　document.querySelector　これは先頭の要素のみ。

　　Const li = document.querySelectorAll(‘li’)

li.forEach(node => node.style.color = ‘purple’);

リスト内の要素を配列として受け取っているため、それぞれの要素の文字色変更をforEachで

行っている。

【DOM　イベントリスナーの登録】

const btn = document.querySelector('#btn');

function hello() {

  alert('hello');

}

btn.addEventListener('click', hello);

btn.removeEventListener('click', hello);

addEventListever(第一引数：イベント, 第二引数：関数)

複数の関数を使用することが出来る。下記はどちらも呼び出される。

btn.addEventListener('click', hello);

btn.addEventListener('click', hello2);

【DOMContentLoaded と loadの違い】

ページの初期表示に関するイベントで、下記のように記述する。

document.addEventListener('DOMContentLoaded', hello);

window.addEventListener('load', hello);

DOMContentLoaded　DOMの要素が読み込まれた時点で画面に表示する。

Load　　　　　　　　　　　　　　　要素内のデータがすべて読み込まれた状態で画面に表示する。

Loadの場合は、画像等のデータサイズが大きい資源を利用する場合に画面表示を待機することとなる。基本的には画面の初期表示については、早く表示ができた方が良いとされるため、特別な理由で画面描画を待機したい要素以外については、DOMContentLoadedを選択することが好ましい。

【Classの実装】

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

 // インスタンス

const ta = new TextAnimation('.animate-title');

const ta2 = new TextAnimation('.animate-title-2');

setTimeout(() => {

    ta.animate();

    ta2.animate();

}, 10000);

});

class TextAnimation {

    //コンストラクタ

    constructor(el){

        this.el = document.querySelector(el);

        this.chars = this.el.innerHTML.trim().split("");

        this.el.innerHTML = this.\_splitText();

    }

    // プライベートメソッド PREFIXに \_ を付加

    \_splitText() {

        return this.chars.reduce((acc, curr) => {

            curr = curr.replace(/\s+/, '&nbsp;');

            return `${acc}<span class="char">${curr}</span>`;

        }, "");

    }

    // パブリックメソッド

    animate() {

        this.el.classList.toggle('inview');

    }

}

クラス自体は画面の要素が読み込まれた時点では、DOM要素として検出する必要がない。そのような要素が含まれることがないので、DOMContentLoadedの外に記載する。

自身のプロパティ値を参照する際はthisで参照する。

子に該当するスコープのthisを親スコープのthisで参照したい場合。（Thisはオブジェクト）

１．(子オブジェクト).bind(this);

２．変数に親オブジェクトのthisを格納して、子オブジェクトから親オブジェクトを格納した変数を参照する。

addEventLisner(‘click’, ta.animate);のようにイベントハンドラによりメソッドを呼び出す場合は、clickイベントが発生した元の要素（buttom等）がオブジェクトとして渡される。

クラスのパブリックメソッド内でthisを使用して渡したオブジェクトのプロパティ値を参照する場合は、どのオブジェクトの要素を渡すかを意識する必要がある。

addEventLisner(‘click’, ta.animate.bind(ta)); // ボタンオブジェクトではなく、インスタンスオブジェクトを明示的にbindしてクラスメソッドを参照している。

【クラス継承】

// 継承

class TextAnimation2 extends TextAnimation {

    //コンストラクタ

    constructor(el){

        super(el);

    }

    // オーバーライド

    animate() {

        this.el.classList.toggle('inview');

        console.log('aaa');

    }

}