

【JS】基礎

基礎文法編

■JavaScriptとHTMLファイルとの連携方法

- ▶ ☆ JavaScriptとHTMLファイルとの連携方法

■初歩的注意

- ▶ ※ 大文字と小文字を区別する言語である。
- ▶ ※ 文字列は"か""どちらで囲んでもいい。
- ▶ ※ 中身を書き得られたくない変数は**すべて大文字**で書こう（慣習） ※TAX_RATEなど
- ▶ ※ ブロック{}内で定義した定数,変数はそのブロック内でだけ有効
- ▶ ※ コード全体をブロック{}で囲んでおいたほうがいい

■データ型の種類

プリミティブ型（基本型）

- string 'hello' "世界"
- number 5 4.3 -20 -1.8
- boolean true false
- undefined undefined ※宣言したが値を代入していない変数など
- null null

オブジェクト（複合型）

- object [2, 5, 8, 9, 11] {a:3, b:5}
- function

■基礎

- ▶ コメントのしかた
- ▶ エラーCheck容易に
- ▶ VBAのDebug.Print
- ▶ 定数を定義
- ▶ 変数を定義
- ▶ データ型を調べる

【JS】基礎

基礎文法編

■JavaScriptとHTMLファイルとの連携方法

- ▶ ☆ JavaScriptとHTMLファイルとの連携方法

■初歩的注意

- ▶ ※ 大文字と小文字を区別する言語である。
- ▶ ※ 文字列は"か""どちらで囲んでもいい。
- ▶ ※ 中身を書き得られたくない変数は**すべて大文字**で書こう（慣習） ※TAX_RATEなど
- ▶ ※ ブロック{}内で定義した定数,変数はそのブロック内でだけ有効
- ▶ ※ コード全体をブロック{}で囲んでおいたほうがいい

■データ型の種類

プリミティブ型（基本型）

- string 'hello' "世界"
- number 5 4.3 -20 -1.8
- boolean true false
- undefined undefined ※宣言したが値を代入していない変数など
- null null


オブジェクト（複合型）

- object [2, 5, 8, 9, 11] {a:3, b:5}
- function

■基礎

- ▶ コメントのしかた // で行末まで。 /* */ で囲めば改行可能。
- ▶ エラーCheck容易に 'use strict';
- ▶ VBAのDebug.Print console.log('Hello, World!');
- ▶ 定数を定義 const hoge = 値;
- ▶ 変数を定義 let hoge = 値;
- ▶ データ型を調べる console.log(typeof 式);

▶ boolean以外を評価	
▶ VBAのMsgBox	
■条件分岐	
▶ 条件分岐	
▶ 比較演算子	
▶ 論理演算子	
▶ 2 股分岐の略記	
▶ boolean以外も評価	
▶ switch文	
■繰り返し処理	
▶ for文	
▶ while文	
▶ do-while文	
▶ foreach文	
▶ foreach文 (index付)	
▶ 中断	
▶ 関数を定期的にリピ	
▶ 低負荷の定期的リピ	
■関数	
▶ 関数を定義	
▶ 名付けずに関数定義	
▶ 即席関数定義	
▶ 値返すだけの "	
▶ 関数の呼び出し	
▶ 配列の全要素を渡す	
■例外処理	
▶ わざと例外を投げる	
▶ 例外受け取って処理	

▶ boolean以外を評価	console.log(Boolean(式));
▶ VBAのMsgBox	alert(文字列); confirm(文字列);
■条件分岐	
▶ 条件分岐	if else if else
▶ 比較演算子	> < >= <= === !==
▶ 論理演算子	&& !()
▶ 2 股分岐の略記	条件式 ? 真での処理 : 偽での処理; や 条件式 ? 真での値 : 偽での値
▶ boolean以外も評価	false 0 " null undefined
▶ switch文	switch (式) { case 値: 処理; break; default: 処理; break; }
■繰り返し処理	
▶ for文	for (let i = 1; i <= 10; i++) { 処理 }
▶ while文	while (条件式) { 処理; 条件に関する処理; }
▶ do-while文	do { 処理; 条件の処理; } while (条件式); ※一度は必ず実行
▶ foreach文	配列.forEach((好きな変数) => { 処理 });
▶ foreach文 (index付)	配列.forEach((好きな変数, index用の変数) => { 処理 });
▶ 中断	continue; break;
▶ 関数を定期的にリピ	const intervalId = setInterval(関数名カッコなし, 間隔 _{ミリ} 秒); （条件分岐節やイベントドリブン中の） clearInterval(intervalId);
▶ 低負荷の定期的リピ	const timeoutId = setTimeout(関数名カッコなし, 間隔 _{ミリ} 秒); （条件分岐節やイベントドリブン中の） clearTimeout(timeoutId);
■関数	
▶ 関数を定義	function helloWorld() { … return 値; }
▶ 名付けずに関数定義	const 定数名 = function(p1, …) { … return 値; };
▶ 即席関数定義	() => { 処理 }; や (p1, p2, p3) => { 処理 }; や p => { 処理 };
▶ 値返すだけの "	(a, b, c) => a + b + c や a => a * 2 ※ 1 行で書ける
▶ 関数の呼び出し	hoge(<i>arg1</i> , …) ※ 末尾に  が必要なことも当然ある
▶ 配列の全要素を渡す	関数(…, …a, …)
■例外処理	
▶ わざと例外を投げる	try { … throw new Error("エラーメッセージ"); … }
▶ 例外受け取って処理	catch (e) { 処理 }

▶ 例外発生しても処理

オブジェクト編

■文字列

▶ 特殊な文字を表現	
▶ 文字列の結合	
▶ 変数展開	
▶ 数値への変換	
▶ 長さを取得	
▶ ExcelのMID関数	
▶ n 番目の文字を取得	
▶ 配列要素をJoinする	
▶ VBAのSplit()	
▶ 大文字にする	

■数値

▶ (算術)演算子	
▶ 複合代入演算子	
▶ ☆ 合計	
▶ ☆ 平均	
▶ 端数処理	
▶ $0 \leq \text{乱数} < 1$ の生成	
▶ $\text{min} \leq \text{乱数} < \text{max} \in \mathbb{N}$	
▶ TEXT(数, "000")	

■配列

▶ 配列を定義	
▶ 要素を取得	
▶ 要素の値を変更	
▶ 要素数を取得	
▶ 頭尾に要素を追加	

▶ 例外発生しても処理 finally { 処理 }

オブジェクト編

■文字列

▶ 特殊な文字を表現	<code>\' \' \'n \'t</code>
▶ 文字列の結合	<code>+</code>
▶ 変数展開	<code>`Hello, \${name}`</code> ※バッククオート
▶ 数値への変換	自然と変換 または <code>parseInt(文字列, 10)</code>
▶ 長さを取得	<code>文字列.length</code>
▶ ExcelのMID関数	<code>文字列.substring(開始位置, 終了位置)</code> ※第2引数省略可
▶ n 番目の文字を取得	<code>文字列[n]</code>
▶ 配列要素をJoinする	<code>文字列.join(区切り文字)</code>
▶ VBAのSplit()	<code>文字列.split(区切り文字)</code>
▶ 大文字にする	<code>文字列.toUpperCase()</code>

■数値

▶ (算術)演算子	<code>+</code> <code>-</code> <code>*</code> <code>/</code> <code>%</code> <code>**</code>
▶ 複合代入演算子	<code>price++;</code> <code>price--;</code>
▶ ☆ 合計	
▶ ☆ 平均	
▶ 端数処理	<code>Math.floor(数値)</code> <code>.ceil(数値)</code> <code>.round(数値)</code> <code>数値.toFixed(〇)</code>
▶ $0 \leq \text{乱数} < 1$ の生成	<code>Math.random()</code>
▶ $\text{min} \leq \text{乱数} < \text{max} \in \mathbb{N}$	<code>Math.floor(Math.random() * (max - min + 1)) + min</code>
▶ TEXT(数, "000")	<code>String(数値).padStart(桁数, '0')</code>

■配列

▶ 配列を定義	<code>const a = [80, 90, 40];</code> ※letでもOK
▶ 要素を取得	<code>a[n]</code>
▶ 要素の値を変更	<code>a[n] = 値;</code>
▶ 要素数を取得	<code>a.length</code>
▶ 頭尾に要素を追加	<code>a.unshift(追加要素, ...)</code> <code>a.push(追加要素, ...)</code>

▶ 頭尾の要素を削除	
▶ 途中要素を追加削除	
▶ 別配列から奪う	
▶ 連結	
▶ 破壊的関数に渡す	
▶ 各要素を楽に格納	
▶ 値を交換	
▶ 各要素に同処理	
▶ 条件にあう要素抽出	
▶ ☆ シャッフル	

■オブジェクト

▶ オブジェクトを定義	
▶ 値を取得	
▶ 値を上書き	
▶ プロを追加	
▶ プロを削除	
▶ 別オブに各プロ追加	
▶ 各プロを楽に格納	
▶ キーの配列を取得	

▶ ※ 基本型のデータを変数に格納すると、変数に値がそのまま格納される。
 いっぽう複合型のデータを変数に格納すると、変数に**参照先の住所**だけが格納される。

▶ ※ 複合型でも、値をそのまま格納したいとき → スプレッド演算子 **...**

■日時

▶ 世界標準の現在日時	
▶ Dateオブを作る	
▶ 年月日曜日を数値で	
▶ 時分秒 _ミ 秒を数値で	
▶ 世界標準の時刻に	
▶ 日時を特定のものに	
▶ ○日後, ○秒後, ...に	

▶ 頭尾の要素を削除	a.shift()	a.pop()
▶ 途中要素を追加削除	a.splice(変更位置, 削除数, 追加要素, 追加要素, ...)	
▶ 別配列から奪う	a.push(a2.splice(奪う要素のIndex, 1)[0]); ※ [0] 忘れない	
▶ 連結	[...a1, ...a2]	
▶ 破壊的関数に渡す	関数(..., [...a], ...)	
▶ 各要素を楽に格納	const [変数1, 変数2, ...] = a;	: 分割代入 ※レスト構文も
▶ 値を交換	[x, y] = [y, x]	: 分割代入
▶ 各要素に同処理	const a = a2.map(変数 => { ... return 式; });	
▶ 条件にあう要素抽出	const a = a2.filter(変数 => { " }); ※trueを返す時だけ採用	
▶ ☆ シャッフル		

■オブジェクト

▶ オブジェクトを定義	const point = {x: 100, y: 180};	
▶ 値を取得	オブ.キー	オブ['キー']
▶ 値を上書き	オブ.キー = 新しい値;	
▶ プロを追加	オブ.新しいキー = その値;	
▶ プロを削除	delete オブ.キー;	
▶ 別オブに各プロ追加	...オブ	: スプレッド構文
▶ 各プロを楽に格納	const {キー1, キー2, ...} = オブ;	: 分割代入 ※レスト構文も
▶ キーの配列を取得	Object.keys(オブ)	

▶ ※ 基本型のデータを変数に格納すると、変数に値がそのまま格納される。
 いっぽう複合型のデータを変数に格納すると、変数に**参照先の住所**だけが格納される。

▶ ※ 複合型でも、値をそのまま格納したいとき → スプレッド演算子 **...**

■日時

▶ 世界標準の現在日時	Date.now()	※UTCなる数値で取得
▶ Dateオブを作る	new Date(?????)	
▶ 年月日曜日を数値で	Dateオブ.getFullYear()	.getMonth() .getDate() .getDay()
▶ 時分秒 _ミ 秒を数値で	Dateオブ.getHours()	.getMinutes() .getSeconds() .getMilliseconds()
▶ 世界標準の時刻に	Dateオブ.getTime	※UTCなる数値で取得
▶ 日時を特定のものに	Dateオブ.setHours(時 , 分, 秒, _ミ 秒) など	
▶ ○日後, ○秒後, ...に	Dateオブ.setDate(そのDateオブ.getDate() + ○); など	

■クラス

- ▶ ☆ クラスを作る流れ
- ▶ ※ クラスとはずばり工具箱である。
- ▶ ※ constructor() は "new" で新しいインスタンスをつくったときに必ず実行される特殊なメソッド。
- ▶ ※ プロパティは外からイジるなかれ！（カプセル化せよ）
- ▶ プロパティの値を取得
- ▶ プロパティの値を設定（変更）
- ▶ ※ あるカテゴリーにおいて有用なメソッドを作りたい → クラスの仕組みを利用
- ▶ ☆ クラスを継承する流れ

DOM編

■DOMについて

- ▶ ☆ ブラウザがしていることの流れ
- ▶ ☆ DOMツリーとは

■要素の指定

- ▶ セレクタに合う初の要素
- ▶ セレクタに合う要素すべて
- ▶ あるid属性をもつ要素
- ▶ あるタグの要素
- ▶ あるクラスをもつ要素
- ▶ ☆ 子ノード、子要素ノード
- ▶ ☆ 親ノード、兄弟ノード、兄弟要素ノード

■イベントドリブン

- ▶ 〇秒後（復習）
- ▶ クリック
- ▶ ダブルクリック
- ▶ focusされた時
- ▶ focusが外された時

■クラス

- ▶ ☆ クラスを作る流れ
- ▶ ※ クラスとはずばり工具箱である。
- ▶ ※ constructor() は "new" で新しいインスタンスをつくったときに必ず実行される特殊なメソッド。
- ▶ ※ プロパティは外からイジるなかれ！（カプセル化せよ）
- ▶ プロパティの値を取得
- ▶ プロパティの値を設定（変更）
- ▶ ※ あるカテゴリーにおいて有用なメソッドを作りたい → クラスの仕組みを利用
- ▶ ☆ クラスを継承する流れ

DOM編

■DOMについて

- ▶ ☆ ブラウザがしていることの流れ
- ▶ ☆ DOMツリーとは

■要素の指定

- ▶ セレクタに合う初の要素
- ▶ セレクタに合う要素すべて
- ▶ あるid属性をもつ要素
- ▶ あるタグの要素
- ▶ あるクラスをもつ要素
- ▶ ☆ 子ノード、子要素ノード
- ▶ ☆ 親ノード、兄弟ノード、兄弟要素ノード

■イベントドリブン

- ▶ 〇秒後（復習）
- ▶ クリック
- ▶ ダブルクリック
- ▶ focusされた時
- ▶ focusが外された時

- ▶ テキストの内容が少しでも変更された時
- ▶ テキストの内容の変更が確定した時
- ▶ 送信された時
- ▶ マウスを動かした時
- ▶ キーが押された時
- ▶ イベントに関する情報を受け取る
- ▶ イベント時に自動的になされてしまう処理を解除する
- ▶ ☆ イベントの伝播を利用し、addEventListener() を設定するノードを少なくする

■イベントドリブンをわざと発動

- ▶ クリックしたことに

■処理

- ▶ テキストを変更
- ▶ クラス属性の値（クラス）を変更
- ▶ クラスを追加
- ▶ クラスを除外
- ▶ 要素に特定のクラスがついてるか
- ▶ 特定のクラスをつけ外し
- ▶ ある属性の値を変更
- ▶ スタイルを変更（やむを得ず）
- ▶ 新値や新テキストをHTML内に保持
- ▶ 要素を作成
- ▶ 要素を複製
- ▶ 作成要素を末尾に追加
- ▶ 作成要素を指定要素の前に挿入
- ▶ 要素を削除
- ▶ あるフォーム要素を無効にする
- ▶ ある要素にフォーカスする
- ▶ テキストボックスの内容
- ▶ セレクトボックスで現在選択されている内容
- ▶ セレクトボックスで選択中の Index の番号

- ▶ テキストの内容が少しでも変更された時 要素指定.addEventListener('input', 関数);
- ▶ テキストの内容の変更が確定した時 要素指定.addEventListener('change', 関数);
- ▶ 送信された時 form要素指定.addEventListener('submit', 関数);
- ▶ マウスを動かした時 document.addEventListener('mousemove', 関数);
- ▶ キーが押された時 document.addEventListener('keydown', 関数);
- ▶ イベントに関する情報を受け取る addEventListener() に渡す関数に第1引数を設定
- ▶ イベント時に自動的になされてしまう処理を解除する Eventオブジェクト.preventDefault();
- ▶ ☆ イベントの伝播を利用し、addEventListener() を設定するノードを少なくする

■イベントドリブンをわざと発動

- ▶ クリックしたことに 要素指定.click (ex)

■処理

- ▶ テキストを変更 要素指定.textContent = '新テキスト';
- ▶ クラス属性の値（クラス）を変更 要素指定.className = '新値';
- ▶ クラスを追加 要素指定.classList.add('追加値');
- ▶ クラスを除外 要素指定.classList.remove('除外値');
- ▶ 要素に特定のクラスがついてるか 要素指定.classList.contains('値'); ※返: Boolean
- ▶ 特定のクラスをつけ外し 要素指定.classList.toggle('値');
- ▶ ある属性の値を変更 要素指定.属性名※ = '新値';
- ▶ スタイルを変更（やむを得ず） 要素指定.style.CSSプロパティ※ = '新値'
- ▶ 新値や新テキストをHTML内に保持 カスタムデータ属性を利用
- ▶ 要素を作成 const 定数 = document.createElement('h1やp');
- ▶ 要素を複製 const 定数 = 要素指定.cloneNode(true);
- ▶ 作成要素を末尾に追加 親要素指定.appendChild(作成要素);
- ▶ 作成要素を指定要素の前に挿入 親要素指定.insertBefore(作成要素, 指定要素);
- ▶ 要素を削除 要素指定.remove();
- ▶ あるフォーム要素を無効にする 要素指定.disabled = true;
- ▶ ある要素にフォーカスする 要素指定.focus();
- ▶ テキストボックスの内容 要素指定.value (ex)
- ▶ セレクトボックスで現在選択されている内容 要素指定.value
- ▶ セレクトボックスで選択中の Index の番号 要素指定.selectedIndex (ex)

▶ ラジオボタンで現在選択されているか	
▶ チェックボックスで現在選択されているか	

■注意

- ▶ ※ 要素指定の部分は定数にしてしまうことが多い

▶ ラジオボタンで現在選択されているか	要素指定.checked
▶ チェックボックスで現在選択されているか	要素指定.checked

■注意

- ▶ ※ 要素指定の部分は定数にしてしまうことが多い