JavaScript 01

JavaScript / 高度なWEBサイト制作

Re start

JavaScriptとは

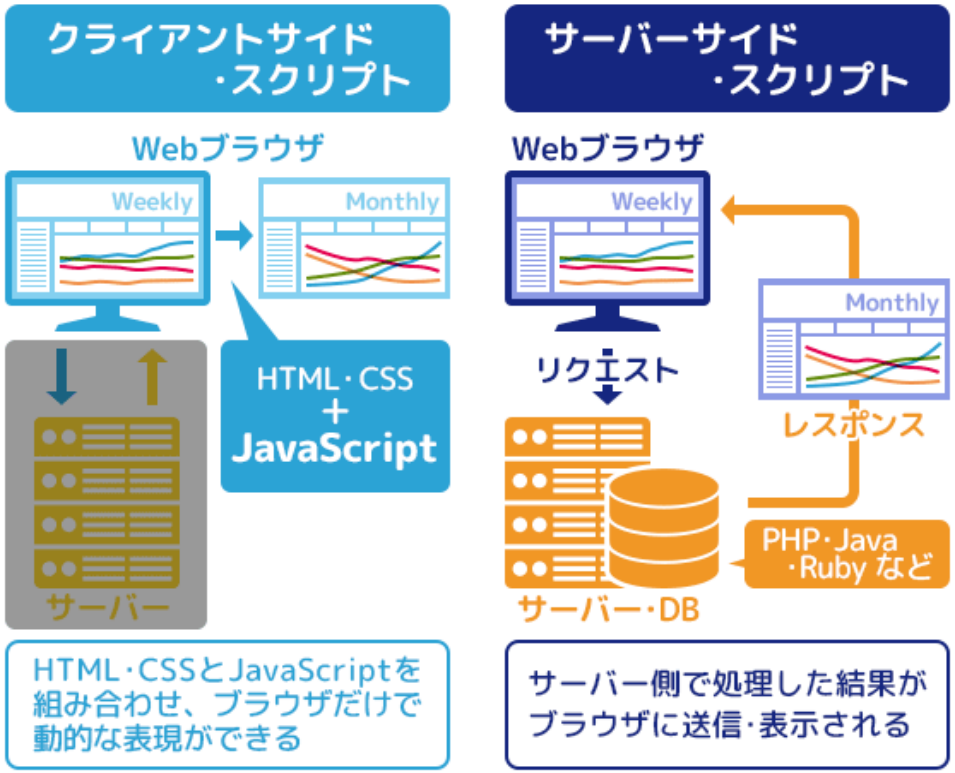
JavaScriptとは、プログラミング言語のうちの一つです。

WEBページに動きをつけるための言語として有名ですが、システムやゲームの開発などで、
幅広く使われている言語です。

近年はバックエンドの開発も
JavaScriptで行う事が可能になり、

- WEBサイトのアニメーション
- システムのバックエンド
- iOSアプリ(iphoneアプリ)
- Androidアプリ
- ゲーム

上のようなものもJavaScriptで
開発できるようになりました。



JavaScriptとJava

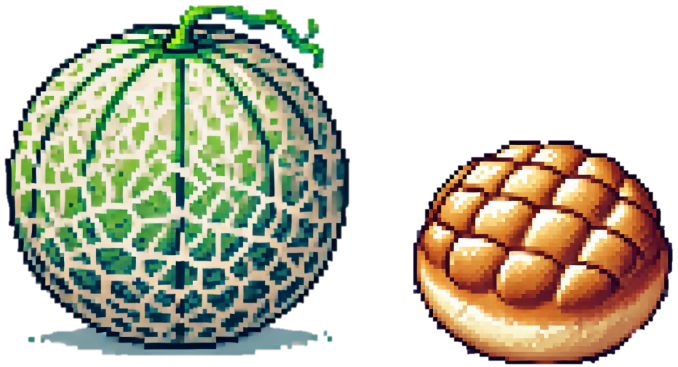
JavaScriptとよく似た名前のJavaという言語があります。
このJavaとJavaScriptは名前は似ているものの、違う言語です。
メロンとメロンパンくらい違います。

Javaは現在、世界で最も使われている言語といわれており、様々なシステムなどで
使われています。
Pythonなどの後発の人気言語が開発された今でも根強い人気があり、
開発言語として高いシェアを誇っています。

JavaScriptとJavaは用途も特徴も目的も全く違う、別の言語です。

JavaScriptは元々LiveScriptという名前でしたが、Javaのあまりの人気に便乗するために
名前をJavaScriptに変更しました。

現在はJavaとの混同を避けるために、
Jsと呼ばれることが多いです。

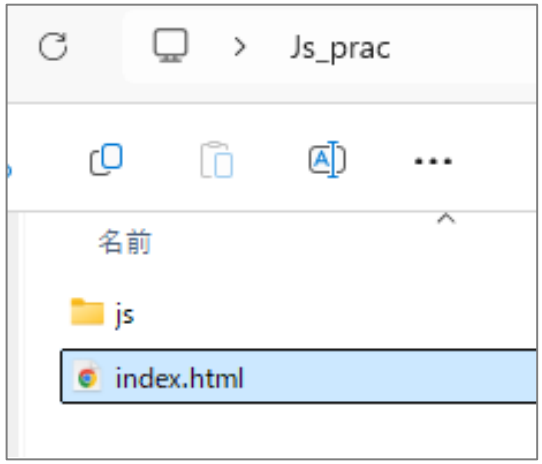


JavaScriptの実行

JavaScriptはWEBブラウザ上で実行できます。
実行するための環境を準備してみましょう。

① HTMLファイルを作る。

js_pracというフォルダを作り、
直下にindex.htmlファイルを作ります。
jsフォルダも作っておきましょう。



② index.htmlにコードを記述

index.htmlをvscodeで開きます。
!と打つとテンプレートが記述されます。
bodyの中に以下を記述しましょう。

```
<script>
  console.log("jsはじめました。");
</script>
```

③ 検証ツールを確認

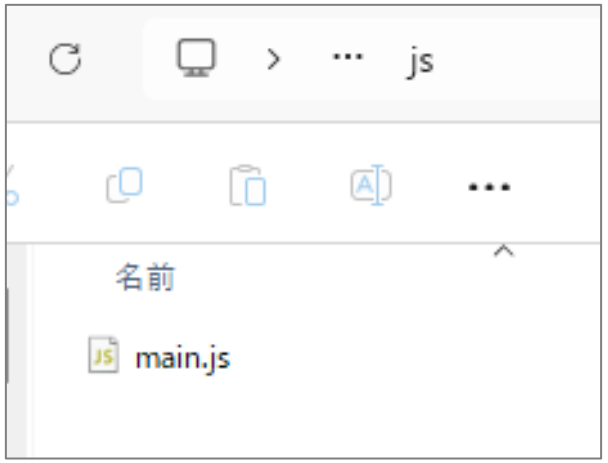
検証ツールを開いてコンソールを確認。
「jsはじめました。」
と表示されていれば成功です。

外部のJavaScriptを実行する

**JavaScriptはHTML内のscriptタグで実行することもできますが、
cssファイルと同じように外部ファイルを読み込むのが一般的です。
外部ファイルから読み込んでみましょう。**

① jsファイルを作る

jsフォルダ内に main.js をつくります。
.jsはJavaScriptファイルの拡張子です。



② htmlファイルに記述

main.jsを読み込む為のコードを
index.htmlに記述します。
bodyの中に以下を記述しましょう。

```
<script src="js/main.js"></script>
```

③ main.jsに記述

main.jsに処理を記述します。
検証ツールのコンソールを確認して
テキストが表示されれば成功です。

```
console.log("jsはじめました。");
```


ReStart

JavaScriptはどこに書く？

JavaScriptはhtml内のどこにでも書くことができます。
目的にもよりますが、bodyタグの一番下の終了タグの上に記述するのが一般的です。
なぜ、一番下に記述するのでしょうか？
ディスカッションしてみましょう。

※ヒント　プログラミングの処理は上から順番に実行されます。

変数とは？

変数とは、データを入れる箱のようなものです。
プログラミングではこの変数を使って、データの操作を行います。

まずは変数を作ってみましょう。

```
let text;
```

「text」という名前の変数を作ることができました。
このように、変数は自由に名前をつけて作ることができます。

これを、「**変数の宣言**」といいます。

変数にデータを入れる

作った変数にデータを入れてみましょう。

```
let text;
text = "jsはじめました。";
console.log(text);
```

記述ができたらコンソールを確認してみましょう。

このように、作った変数にはデータを入れることができ、再利用することができます。
これを「代入」といいます。

変数の宣言と代入は、以下のように省略することができます。

```
let text = "jsはじめました。";
console.log(text);
```

データの操作

変数に入れたデータはそのまま使うだけではなく、操作することができます。

```
let num = 122;
num = 20;
console.log(num);
```

コンソールを確認すると、 20 と表示されていますね。
このように、一度代入された変数にもう一度代入すると、上書きすることができます。

この性質を利用して、以下のようなこともできます。

```
let num = 122;
num = num + 20;
console.log(num);
```

コンソールに 元々代入されていた120に20が加算された数 144 が表示されていますね。
このように計算した結果を代入して利用することができます。

半角英数字で記述する

jsは基本的に半角英数字で記述しますが、以下のように書いても動きます。

```
let はこ = 122;
はこ = 20;
console.log(はこ);
```

このように書いても動きますが、非常に読みにくくなるのでやめましょう。

ただし、以下のように日本語のような全角文字を記述する必要がある場合は、全角を使ってOKです。

```
let text = "jsはじめました。";
console.log(text);
```

命名規則

jsでは変数の宣言のように自分で名前をつける場面が多くあります。
この名前の付け方にもルールがあります。

① 英語で名前をつける

変数の名前を付ける際は、ローマ字ではなく英語でつけましょう。
ローマ字で命名すると、とても読みにくいです。

```
let okyakusannokaishanonamae = "A社";
```

② 単語の区切りは大文字にする

2つ以上の単語をつなげて命名する場合は、区切りを大文字にします。
この記述の方法をキャメルケースといいます。

```
let customerCompanyName = "A社";
```

コメントアウトを使う

jsはHTMLやCSSと比べて、処理が複雑です。
どのような処理を行っているか、コメントアウトで説明するようにしましょう。

```
// テキストを出力
let num = "jsはじめました。";
console.log(text);
```

このように、// と記述するとその行をコメントアウトすることができます。

複数の行をコメントアウトするには、

```
/*
テキストを出力
制作者：山田
*/
let num = "jsはじめました。";
console.log(text);
```

このように、開始を /* 終了を */ と記述することでコメントアウトできます。

インデントを徹底する

jsにも入れ子構造は存在します。
処理を記述する時に混乱を招かないように、インデントを徹底しましょう。

```
if (num % 2 == 0) {
    if (num >= 10) {
        console.log("numは偶数で10以上ですよ");
    } else {
        console.log("numは偶数ですよ");
    }
} else {
    console.log("numは奇数ですよ");
}
```

処理の区切り

ひとつの処理の区切りには、必ず ; を入力します。
区切りがなくても動きますが、意図した動きにならない可能性があります。

```
if (num % 2 == 0) {
    if (num >= 10) {
        console.log("numは偶数で10以上ですよ");
    } else {
        console.log("numは偶数ですよ");
    }
} else {
    console.log("numは奇数ですよ");
}
```

データ型

jsに限らず、プログラミングでは、データの種類によって取り扱い方が変わります。
この分類のことをデータ型といいます。

jsで取り扱う代表的なデータ型には、以下のような種類があります。

データ型	説明	使用例
整数(number)	1 ,18 のような整数 -2 のように負の数値も含む	10, 08, 0
小数(number)	小数を含む数値	0.02, -1.543, 5.000
文字列(string)	複数の文字からなるデータ	“こんにちは”, ”Hello”, “あ”
ブール(boolean)	真偽値 真か偽を示す	true, false

この他にもデータ型の種類は複数あります。
jsは取り扱っているデータの型が把握しづらい言語です。
データ型を理解することにより、よりスムーズにプログラミングを進める事ができます。

数値（number）

数値には 33 のような整数と、0.0032 のような小数の値があります。
算数と同じように足したり引いたりなどの計算ができます。

また、とても大きい数(9007199254740991よりも大きい数)を取り扱う場合は
BigInt という型を使います。

値が数値かどうかを判別するには、typeof　を使います。
typeof の後に値を記述すると、型を判別することができます。

```
let num = 10;
console.log(typeof num); //number
```

文字列（string）

文字として取り扱うデータ型を文字列といいます。
””（ダブルクォート）か’’（シングルクォート）で囲んだものを文字列として取り扱います。

```
let num = 10;
let str = "10";
console.log(typeof num); //number
console.log(typeof str); //string
```

このように、同じ 10 であっても、”” で囲むと全く別のデータになります。

また、文字列を改行するには、¥n と記述する必要があります。

```
let str = "吾輩は¥n猫¥nである。";
console.log(str);
```

```
吾輩は
猫
である。
```
main.js:11

真偽値（boolean）

真偽値には true と false の値があります。

```
let bool = true;
console.log(typeof bool); //boolean
```

真偽値はプログラミングで最も重要となる値です。

この真偽値の使い方は今後の内容で出てきます。
今の段階では、真偽値というデータ型の存在だけ知っていれば大丈夫です。

演算子とは？

演算子とは、＋ や ー のような、計算をする記号のことです。

演算子は、おおまかに分けて

- 算術演算子
- 代入演算子
- 比較演算子
- 論理演算子

など、多くの種類があります。

算術演算子

足し算や掛け算などの計算を行う演算子を算術演算といいます。
算術演算は数値だけではなく、文字列を結合することもできます。

```
// 足し算
let add = 5 + 2;
console.log(add); // 7

// 引き算
let sub = 5 - 2;
console.log(sub); // 3

// 掛け算
let mul = 10 * 5;
console.log(mul); // 50

// 割り算
let div = 10 / 5;
console.log(div); // 2

// 余りを求める
let rem = 11 % 2;
console.log(rem); // 1

// 文字列の結合
let greeting = "おはよう";
greeting = greeting + "ございます";
console.log(greeting); // おはようございます
```

代入演算子

変数に値を代入する時に使うのが代入演算子です。
計算と代入を同時に行うものもあります。

```
// 代入する
let num = 10;
console.log(num); // 10

// 足し算した値を代入
num += 5;
console.log(num); // 15

// 引き算した値を代入
num -= 5;
console.log(num); // 10

// 掛け算した値を代入
num *= 2;
console.log(num); // 20

// 割り算した値を代入
num /= 5;
console.log(num); // 4
```

比較演算子

ふたつの値を比較する時に使うのが比較演算子です。
比較の結果は、true か false が出力されます。

```
let a = 10;
let b = 20;

// a と b の値が等しい場合はtrue
console.log(a == b); // false

// a と b の値が等しくない場合はtrue
console.log(a != b); // true

// a が b より小さい場合はtrue
console.log(a < b); // true

// a が b 以下の場合はtrue
console.log(a <= b); // true

// a と b の値が等しくデータ型も同じ場合はtrue
console.log(a === b); // false
```

論理演算子

論理演算子はふたつ以上の比較演算を組み合わせるときに使います。

例えば、
・天気が晴れで、かつ人数が９人集まれば野球をする。
・天気が晴れ、もしくは人数が９人集まれば野球をする。
この２つの条件は似ているようで全く違います。

このように２つ以上の比較演算の組み合わせ方を指定するのが、論理演算子です。

```
let a = 10;
let b = 20;

// a が偶数かどうかの結果を代入
let isEven = a % 2 == 0;

// a が b より大きくて、かつ偶数である
console.log(a > b && isEven); // false

// a が b より大きい、もしくは偶数である
console.log(a > b || isEven); // true
```

インクリメントとデクリメント

処理を書いていると、数値を１ずつ増やしたい、１ずつ減らしたいといった場面が出てきます。
こういった時に使うのが、インクリメントとデクリメントです。

```
let a = 10;

// インクリメント
a++;
console.log(a);

// デクリメント
a--;
console.log(a);
```

プログラミングの三大処理

プログラミングには基本となる３つの処理があります。
この３つの処理を組み合わせることで、複雑なアプリやシステムは動いています。

① 順次

jsを含めたプログラミング言語は上から順番に処理を行います。
これを順次処理といいます。（朝起きて、電車に乗って、学校に行く）

② 分岐

雨が降っていたら→傘を持っていく、暑かったらエアコンをいれる、などの
条件によって処理の内容が変わることを分岐処理といいます。
入力が間違っていたら警告する、合っていたら処理を進めるなど、
プログラミングには分岐が必要な場面が多く見られます。

③ 反復

ご飯がなくなるまで食べる、スライムを倒すまで攻撃するなど、
何か条件をクリアするまで処理を繰り返すことを反復処理といいます。
これもプログラミングには必要不可欠な処理です。

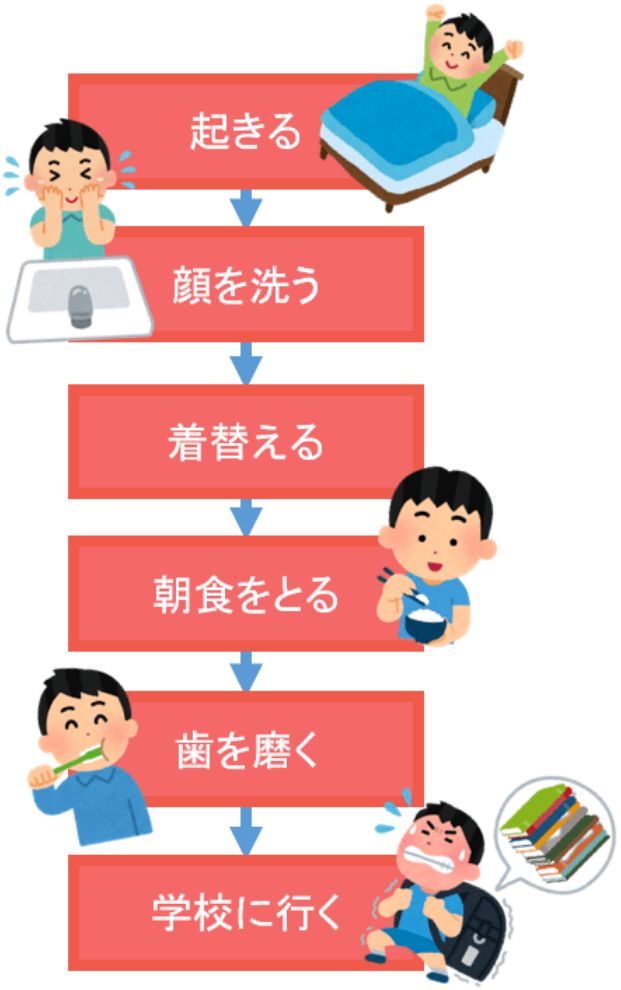
順次

①の処理のつぎに②の処理を実行する。
このように順番に処理をおこなっていくことを順次処理といいます。

一日の流れや運動会のプログラムなども、同じように順番に
実行していくものです。

プログラミングでも同じように、上から処理を行っていきます。

処理を記述する際に実行したい順番を意識するところから
プログラミングはスタートします。

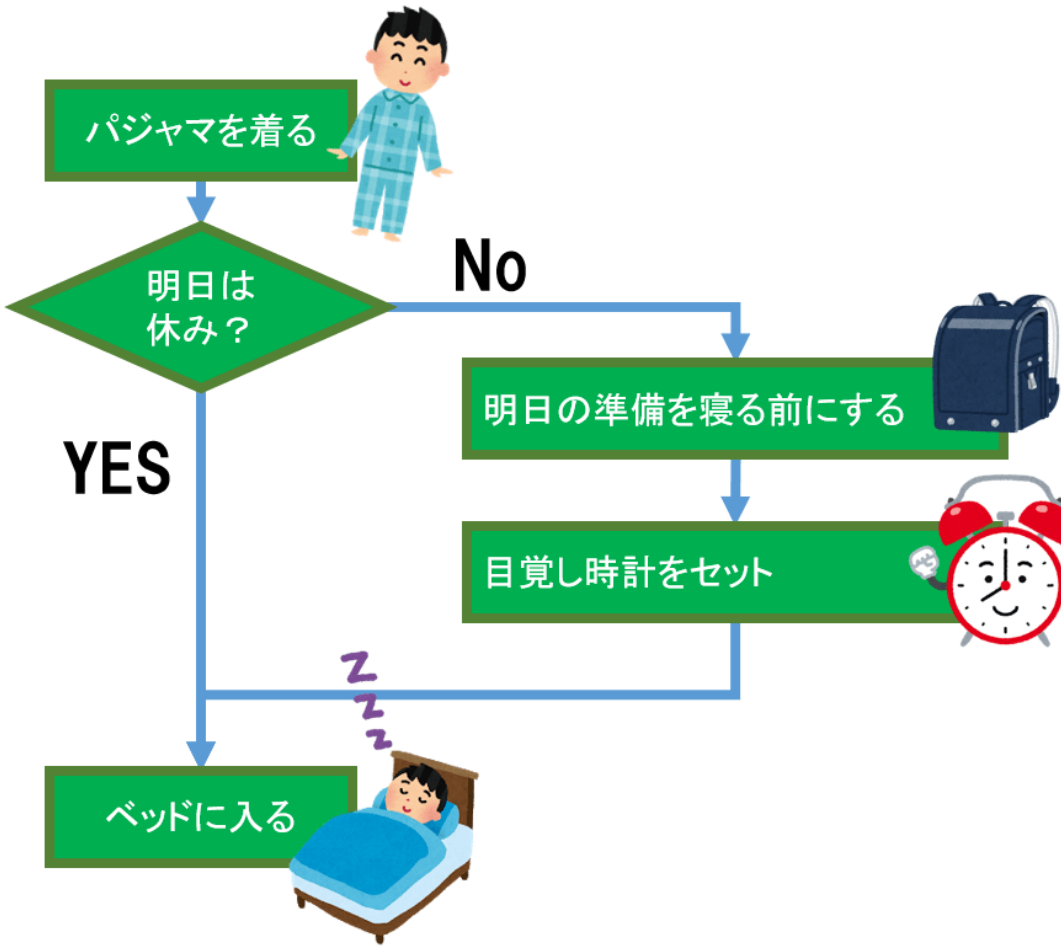


分岐

ある条件によって実行する内容を変えることを
分岐 といいます。

雨の日は傘をさす、子供なら映画が半額、など
日常生活でも分岐する場面は多くあります。

プログラミングでも分岐の条件を設定すれば
複雑な処理も実現することができます。



反復

ある条件を満たすまで同じ処理を繰り返すことを反復処理 といいます。
ループとも呼ばれます。

例えば

- 指定した温度になるまで、エアコンが動き続ける。
- バイトが終わる時間まで、接客を続ける。
- 渡されたティッシュがなくなるまで、通行人にティッシュを渡す。

などは、全て反復処理です。

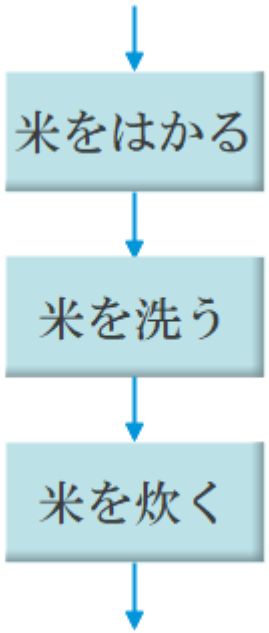
ループは非常に便利な処理ですが、ループを脱出する条件を
設定していないと**無限ループ**に陥ります。

ReStart

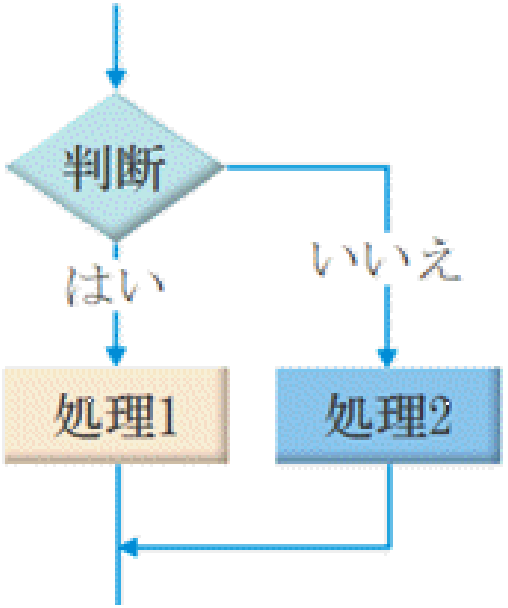
フローチャート

プログラミングの処理を表現する際に書くのがフローチャートです。
処理を説明する資料にも使われますが、プログラマーが頭を整理するために書いたりもします。

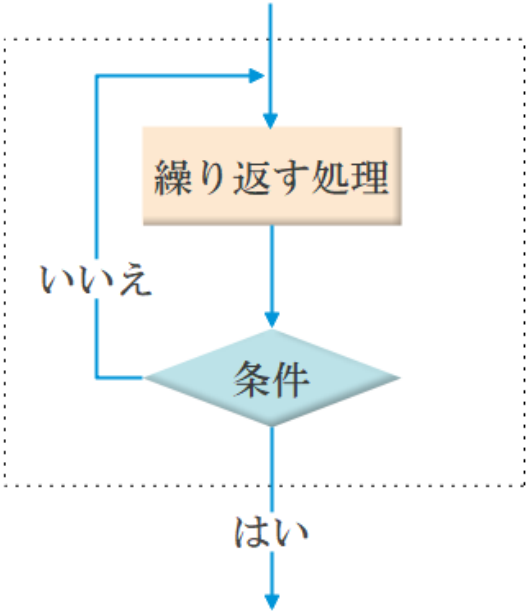
順次



分岐



反復



フローチャートを書いてみよう！

以下の処理の流れをフローチャートに書いてみよう！

① 10時から15時までキャベツを切り続ける。
水を飲む時間のみ休憩することができる。
キャベツをノルマ以上切れたら、ご褒美に違う仕事ができる。

② わたしの家のカレーの作り方。

※ヒント　フローチャートは下の記号を使って書こう！

if

if文を使うことで、分岐を記述することができます。

```
if(条件式){
    処理したい内容
}
```

実際に if文 を書いて、動きを確認してみましょう。
a の値を変更して、試してみましょう。

```
let a = 10;

// 偶数なら、偶数です！と出力
if (a % 2 == 0) {
    console.log("偶数です！");
}
```

else if

if文は複数の分岐にも対応できます。

```
let a = 10;

// 余りの数を出力
if (a % 3 == 2) {
    console.log("余りは２です！");
} else if (a % 3 == 1) {
    console.log("余りは１です！");
} else {
    console.log("３で割り切れます！");
}
```

このように、else if と記述すると、if文に続けて分岐を追加することができます。
最後の else では、条件に一致しなかった場合の処理内容を書くことができます。

ネスト

if文の中にif文を書くこともできます。
このように分岐やループ処理の中に、分岐やループ処理を書くことをネストといいます。
ネストが深くなると、処理の組み立てが複雑になります。
必ずインデントを忘れないようにしましょう。

```
let a = 10;

// 余りの数を出力
if (a % 3 == 2) {
    console.log("余りは２です！");
} else if (a % 3 == 1) {
    console.log("余りは１です！");
    if (a == 10) {
        console.log("a は 10 です！");
    }
} else {
    console.log("３で割り切れます！");
}
```

if文を使ってみよう！

ランダムな整数の生成を使って、課題を解いてみよう！

```
// 0 から n までのランダムな整数の生成
let n = 10;
let randomNum = Math.floor(Math.random() * (n + 1));

// 例：サイコロの場合（1から6までのランダムな整数）
let dice = Math.floor(Math.random() * 6) + 1;
```

①りんごの値段（２５０～３５０）をランダムに生成して表示。
値段が３００円以上なら買わない。３００円以下なら買う。

出力例）
りんごの値段が270円でした！
まさる君はりんごを買いました！

if文課題

② サイコロの目を表示。
　１が出ればアタリ。それ以外はハズレ。

出力例）
サイコロの目は2だった。
ハズレ！

---

③ 晴れと雨をランダムで表示。雨だったら傘を持っていく。

出力例）
今日は雨が降っている。傘を持っていこう。

---

④ 田中さんと山田さんがそれぞれ3分の1でくしゃみをする。
両方くしゃみしたらシンクロしました！と表示。

出力例）
田中さんがくしゃみをしました。山田さんがくしゃみをしました。
シンクロしました！

switch

switch文は、if文と同じように条件で分岐を行うことができます。

```
let num = 1;

switch (num) {
    case 0:
        console.log("0だよ");
        break;
    case 1:
        console.log("1だよ");
        break;
    default:
        console.log("0でも1でもないよ");
        break;
}
```

switch文はtrue false ではなく、値をもとに分岐を扱います。
case の部分に値を指定すると、その値の時の実行内容を記述できます。
break; は処理から抜けるためのもので、break が無いと、その下の処理も実行してしまいます。

分岐課題

① サイコロの目が1のときは大吉。2のときは中吉。3のときは小吉。
その他の目は全て凶。

出力例）
サイコロの目は2だった。
中吉！

---

② 月をランダムで選択。その月の祝日を表示する。
１月：元日、成人の日
２月：建国記念の日
３月：春分の日
４月：昭和の日
５月：憲法記念日、みどりの日、こどもの日
７月：海の日
９月：敬老の日、秋分の日
１０月：体育の日
１１月：文化の日、勤労感謝の日
１２月：天皇誕生日

ifとswitchどっちを使うべき？

ifとswitchは、どちらも同じような動きをします。
ifで可能な条件分岐はswitchでも可能ですし、switchで可能な分岐はifでも書けます。
どちらを使うのが良いのでしょうか？

この質問には答えがありません。
ですが、実務では積極的に if を使う現場がよく見られます。

あきらかにswitchを使った方が分かりやすい場合、
上司からの特別な指示がない場合は、
基本的にifを使うようにするのが無難です。

while

while文は 条件式 が true である間は、ループを続けます。

```
while(条件式){
    処理したい内容
}
```

実際に while 文 を書いて、動きを確認してみましょう。
a の値を変更して、試してみましょう。

```
let num = 5;

// カウントダウンして0になったらループを終了
while (num >= 0) {
    console.log("今の数値は" + num + "です！");
    num--;
}
```

for

for文は、繰り返す範囲（回数）を指定したループを書くことができます。

```
for(let i = 0, i < 回数,i++){
    処理したい内容
}
```

一見複雑そうに見えますが、回数の部分にループしたい回数を指定するだけです。
実際の記述はこうなります。

```
let num = 0;

// カウントアップして5になったらループを終了
for (let i = 0; i < 6; i++) {
    console.log("今の数値は" + num + "です！");
    num++;
}
```

break

break は、処理の途中で抜け出す文です。
while、for などの中で使うと、処理中のループを抜け出して次の処理にすすみます。

```
while(true){
   break;
}
//抜け出して次の処理へ
```

```
// 偶数が出たら即終了
for (let i = 0; i < 20; i++) {
    let randomNum = Math.floor(Math.random() * (11));
    console.log("数値は" + randomNum + "です。");
    if (randomNum % 2 == 0) {
        console.log("偶数なので終了します！");
        break;
    }
}
```

このように、ある条件で途中でループ処理を終わりたい時にbreakを使います。

continue

continue はループ中にその周の処理を終了して、次の周を実行します。
break とは違い、ループの中からは抜け出しません。

```
// 生成された数値のうち、偶数だけ表示する
let str = "";

for (let i = 0; i < 10; i++) {
    let randomNum = Math.floor(Math.random() * (11));
    if (randomNum % 2 == 1) {
        continue;
    }
    str += randomNum + ","; //偶数のみ文字列に結合
}
console.log(str);
```

このコードでは奇数だった場合、continue があるので文字列結合の処理に到達しません。
つまり、偶数の時のみ結合する処理を実行するようになります。

無限ループ

ループを書いたものの、ループから脱出できずにループが終わらないことを無限ループといいます。
特にwhile文で起きやすく、コードのミスから発生します。

```
let num = 1;

// 条件式が常にtrueになるため、無限ループする
while (num > 0) {
    console.log("今の数値は" + num + "です！");
    num++;
}
```

上のコードではwhileの条件式がfalseになることがないため、無限ループとなります。

無限ループが発生してしまった場合は、ブラウザを閉じれば強制的に処理を停止できます。
あわてずさわがず、そっとブラウザを閉じましょう。

スコープ

letで変数を作る場合、同じ名前を二度宣言することはできません。
ですが、下のコードの場合、エラーになることはありません。
なぜでしょう？

```
for (let i = 0; i < 10; i++) {
    let UserName = "tanaka";
}

let userName = "suzuki";
console.log(userName);
```

変数には有効範囲があり、 if や for などの { } 内で宣言された変数は、そのブロックの中だけ有効になります。
つまり、処理がブロックを抜けた時点で、その変数は消滅しているのです。
for の中で宣言されている userName はブロックを超えた時点で存在していないので、
下の行の userName を宣言する事ができます。

この変数の有効範囲のことを、スコープ といいます。

ループ課題

① ★を以下のように出力しよう。

★★★★★★★★★★
★★★★★★★★★★
★★★★★★★★★★

---

② ★を以下のように出力しよう。

★
★★
★★★
★★★★

---

③ 田中さんと山田さんがそれぞれ3分の1でくしゃみをする。
両方くしゃみしたらシンクロしました！と表示。
両方くしゃみするまで抽選し、両方くしゃみしたら「シンクロしました！」と表示。
両方くしゃみするまでにかかった抽選回数も表示する。

ループ課題

④ 毎日キャベツを切り続ける。1日につき3分の1の確率で水を飲んで休憩できる。
一日の最後に抽選を行い10分の1の確率で腱鞘炎になるが、
水を飲んで休憩していた場合、腱鞘炎になる確率が30分の1になる。
何日で腱鞘炎になったかを表示。

---

ex 下弦の壱、下弦の弐、下弦の参、下弦の肆、下弦の陸の鬼５人が集まる。
「頭を垂れて蹲え。平伏せよ」
下弦の壱以外の４人がそれぞれ平伏するかどうかを２分の１で抽選。
平伏しないと殺される。
次に下弦の壱以外で発言する順番を抽選。
発言する順に殺される。
全員殺された後で、下弦の壱の発言内容を２分の１で抽選。
「血を分けていただければ、私は必ず順応してみせます」
⇒「何故私がお前の指図で血を与えねばならんのだ」殺される。
「幸せでした～」
⇒血をふんだんにわけてもらって助かる。

配列とは？

配列とは、ひとつの入れ物に複数のものを入れられる箱のようなものです。
変数には１つの値しか入れられませんが、配列には番号をつけて複数の値を入れることができます。

上のお菓子の箱のように、それぞれの箱に番号が振られていて、
番号を指定して中身を見たり、入れ替えたりできます。

この番号のことを、インデックス といいます。
インデックスは このイラストのように、0からはじまります。

配列はどう便利？

配列は使い方を理解すれば、便利な武器になります。

```
// 名前を全部表示する
let array = ["山田","鈴木","田中","佐藤"];

for (let i = 0; i < array.length; i++) {
    console.log(array[i]);
}
```

配列はループと相性がよく、このように短い記述で格納されている値をすべて取り出せます。

また、 配列名.length　と記述すると、その配列の箱の数を取得することができます。
このコードだと配列の箱の数だけループします。
配列のインデックスがいくつあっても、 .length を設定するだけできちんとインデックスの数だけループしてくれます。
便利ですね。

配列への要素の追加と削除

JavaScriptの配列は、要素の追加や削除などの操作ができます。
要素を配列に追加するには push 、インデックスを指定して削除する場合は splice を使います。

```
let array = ["山田","鈴木","田中","佐藤"];
array.splice(3,1); //(インデックス,削除する個数)
array.push("並木");
```

配列は他にもいろいろなデータの操作ができます。
必要に応じて調べて記述しましょう。

配列課題

① 0から9までの連番の配列を作って、出力してみよう。

出力例）0 1 2 3 4 5 6 7 8 9

---

② AさんとBさんが10回じゃんけんをする。
10回じゃんけんが終わったら、 AさんBさんそれぞれが何を出したかを出力しよう。

出力例）Aさんはパーを出した。Bさんはグーを出した。
Aさん履歴 1,パー 2,グー 3,チョキ...

---

③ 0から99までのランダムな数値を10個生成して出力。
出力した数値の合計も出力しよう。

出力例）12 33 43 25 29 59 0 5 19 28
合計は 253 です。

配列課題

④ 0から9までのランダムな数値を10個生成して出力。
ただし、重複してはだめ。

出力例）0 5 3 7 2 9 8 6 1 4

---

⑤ ④の課題で出力した10個のランダムな数値を、昇順に並び替えてみよう。

出力例）0 5 3 7 2 9 8 6 1 4
0 1 2 3 4 5 6 7 8 9

---

⑥ ④の課題で出力した10個のランダムな数値を、降順に並び替えてみよう。

出力例）0 5 3 7 2 9 8 6 1 4
9 8 7 6 5 4 3 2 1 0

関数とは？

関数とは、ある一連の動きをひとまとめにして、何度も使えるようにする機能です。
ピンときにくいと思うので、以下の動画を見てみましょう。

関数を宣言する

関数をつくるときはfunction と書きます。

```
function 関数名(引数){
    return 戻り値;
}
```

function … 関数をつくるための宣言です。

関数名 … 関数の名前で、自由に命名できます。この関数名を呼び出して関数を実行します。

引数 … 関数は何かを受け取って実行する場合が多いです。この受け取るものを引数といいます。

戻り値 … 関数の中では色々な処理を実行しますが、最終的に何か値を返す事ができます。
これを戻り値といいます。

関数の作り方

それでは関数をつくってみましょう。

```
// 引数に渡さないパターン
function greeting(){
    console.log("おはようございます！！");
}

// 引数に名前を渡すパターン
function nameGreeting(name){
    console.log(name + "さん！おはようございます！！");
}
greeting();
nameGreeting("田中");
```

このように、関数を呼び出す時には、関数名( ) と呼び出せば実行されます。
また、この( )の中に関数に渡したいものを入れてあげると、受け取って処理してくれます。
このコードだと、名前を受け取って、その人に挨拶してくれてますね。

引数

引数は自由に命名して、関数の処理内で使うことができます。
また、引数は複数設定することもできます。

```
// 年齢と名前を渡す
function ageNameGreeting(age,name){
    console.log(age + "歳の" + name + "さん！おはようございます！！");
}

ageNameGreeting(40,"田中");
```

コードにあるように、引数を複数設定する場合は、　, (カンマ) で区切ります。
呼び出すときも同じように , (カンマ) で区切って値を関数に渡します。

戻り値

関数が処理した結果を返すように処理する場合は、 return を使います。

```
// 偶数か奇数か
function isEven(num) {
    let result;
    if (num % 2 == 0) {
        result = "これは偶数！";
    } else {
        result = "これは奇数！";
    }
    return result;
}

console.log(isEven(33));
```

この戻したい値のことを、戻り値といいます。
return の後に戻り値を設定すると、関数の処理の結果を返すようになります。

変数に関数を代入する

関数は関数名をつけずにつくることもできます。

```
let isEven = function (num) {
    let result;
    if (num % 2 == 0) {
        result = "これは偶数！";
    } else {
        result = "これは奇数！";
    }
    return result;
}

console.log(isEven(33));
```

このように、直接変数に代入して関数をつくれます。
呼び出すときは、変数名を呼び出せば処理を実行することができます。

関数課題

① 名前を渡すと、さんをつけて返す関数

② 数値の配列を渡すと、配列の数値を全て合計した数値を返す関数

③ 身長(cm)と体重(kg)を渡すと、BMIを返す関数 ※BMIで検索してみよう！

※BMI＝体重（kg）÷（身長（m）×身長（m）

④ 数値の配列を渡すと、昇順でソートして配列を返す関数

⑤ 語尾に「にゃん！」をつける関数

出力例）
吾輩は猫である。名前はまだない。
↓
吾輩は猫であるにゃん！名前はまだないにゃん！

HTMLとJsの連携

JsはHTMLを操作するための言語です。
HTMLの要素とＪｓを連携させ、要素を操作することができます。

HTML

```
<button id="exam" onclick="pushBtn()">押してみて</button>
```

JavaScript

```
function pushBtn(){
    alert("ボタンが押されました");
}
```

この onclick にJsの関数を設定すると、クリック時に関数を実行するようになります。
このようにアラートを表示する以外にも、ボタンを押したら色が変わる、ボタンを押したら
テキストが変更されるなど、様々な操作を行うことができます。

HTML要素の指定

HTMLで作る画面には、無数の要素があります。
Jsで操作するHTML要素を指定するには、どうすればいいのでしょう？

まずは、操作したいHTMLの要素を指定することが必要です。

```
let elm = document.getElementById(‘exam');
```

要素の指定を行うには、getElementById を使います。
この場合、HTMLからbutton という id が設定された要素を探します。

HTMLで基本的にidを使わないのは、js で要素を指定するためです。
js で操作したい要素にのみ id を設定すれば、ソースもわかりやすくミスも少なくなります。

他にも要素を指定する方法はたくさんあります。
どのような条件で指定することができるのか、調べてみましょう。

HTML要素の操作

要素を指定することができたら、要素をJsで操作してみましょう。

```
function pushBtn() {
    let elm = document.getElementById('exam');
    elm.innerHTML = "書き換えました";
}
```

このコードでは、ボタンを押すとボタンに表示されているテキストが変更されます。
このように要素を指定して動きを設定することで、思い通りの操作を実現できます。

ボタンを押したら画面の背景の色を変更、ボタンを押したら表示されている画像が
切り替わるなど、押した要素ではない別の要素を操作することもできます。

イベントの付与

クリック時に～する、スクロールしたら～するなど、操作するタイミングや方法のことを
イベントといいます。
このイベントをJs側で付与するには、addEventListener() を使います。

対象の要素.addEventListener(イベント,関数,false);

```
let elm = document.getElementById('exam');

elm.addEventListener('click', function () {
    alert('ボタンがクリックされました！');
}, false);
```

第一引数にはイベントの種類、第二引数には実行する関数を入れます。
第三引数には別の設定を行うためのbool値が入りますが、今のところは固定でfalseを
設定しておきましょう。

HTMLの要素にイベントを付与する際は、**基本的にはJs側で設定**します。

HTML要素の取得

JsはHTMLを操作するだけでなく、HTMLから値を取得してJsで使うことができます。

HTML

```
<input type="text" id="input">
<button id="exam">押してみて</button>
```

JavaScript

```
elm.addEventListener('click', function () {
    let value = input.value;
    console.log(value + 10);
}, false);
```

このコードでは、入力してボタンを押すと、入力した値に10を足してコンソールに出力します。
ですが、なぜか計算が間違っていますね。なぜでしょう？
①計算が間違う理由を考えてみましょう。
②きちんと計算ができるように、コードを修正してみましょう。

Jsでどんな操作ができる？

JsはHTMLをどこまで操作できるのでしょう？
JsはHTMLの要素を動かしたり、書き換えたり、増やしたり、削除したりできます。
その操作項目は非常に多く、思いつく操作はほぼできます。

- テキストを変更する
- HTMLタグを変更する
- 属性を追加したり削除する
- 画像のスライドショーを作成する
- お問合せフォームの必須制御を行う
- ポップアップウィンドウを表示する
- 3Dモデルを動かす

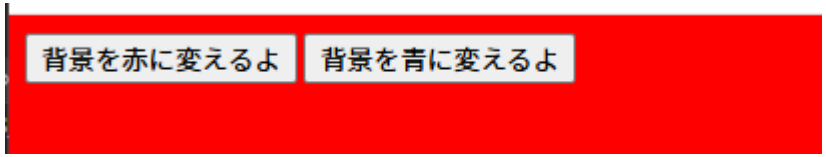
etc.....

など、たくさんのことができます。

その全てを暗記する必要はありません。よく使うものだけ覚えていれば大丈夫です。

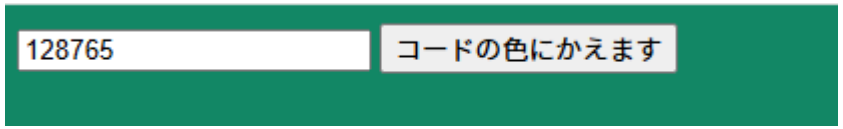
DOM課題①

① ボタン押すと背景の色が赤に変わる。もう一つのボタンを押すと背景が青に変わる。



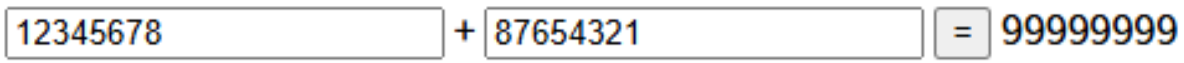
---

② ボタンを押すと、背景が入力したカラーコードの色に変わる。



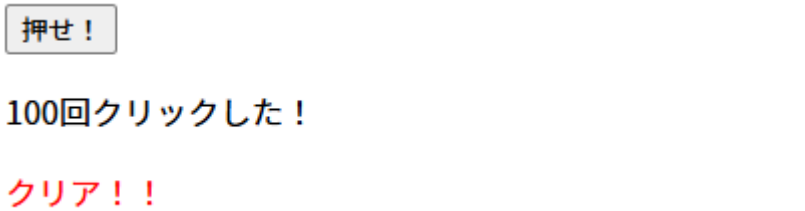
---

③ ボタンを押すと、足し算してくれる。



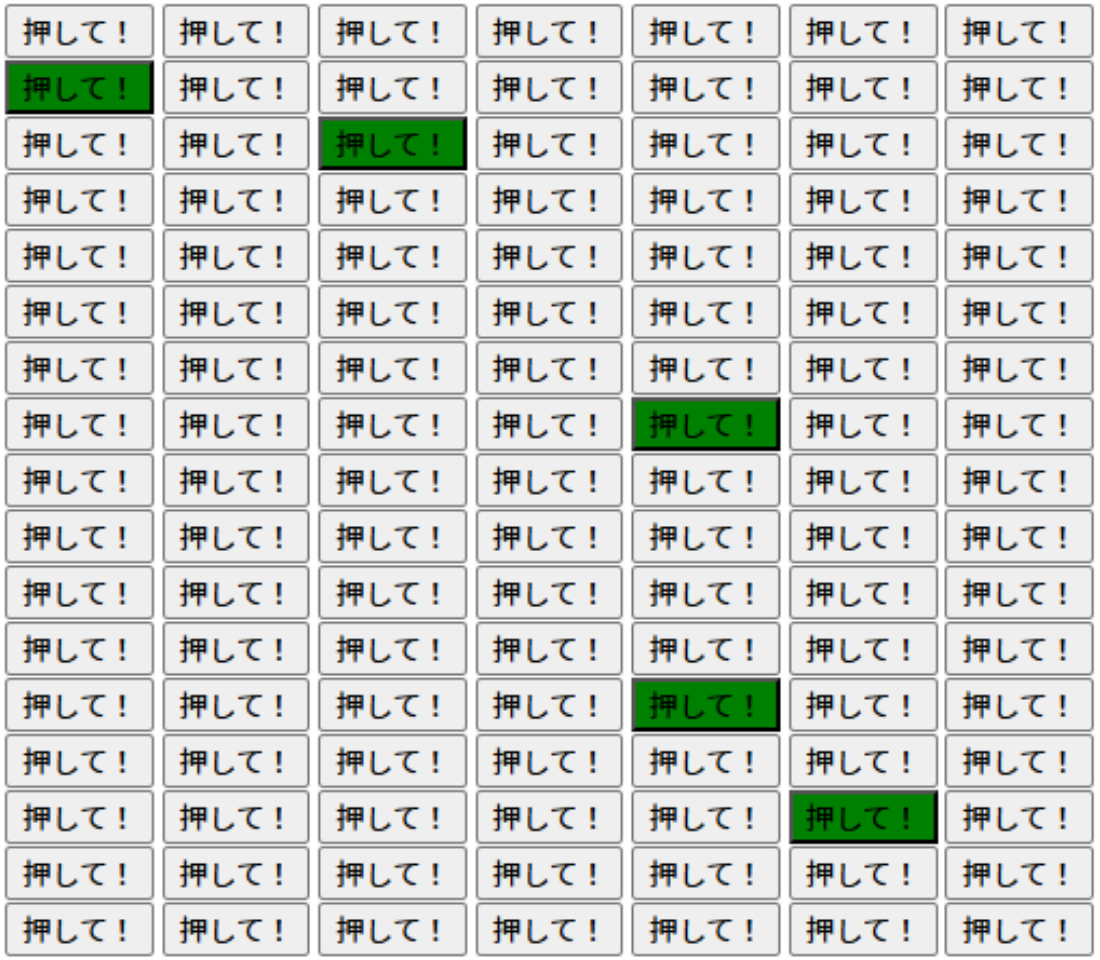
---

④ ボタンを押した回数がカウントされ、100回になると赤いクリアの文字がでる。



DOM課題②

⑤ 大量のボタンを配置。クリックしたボタンの色が変わる。



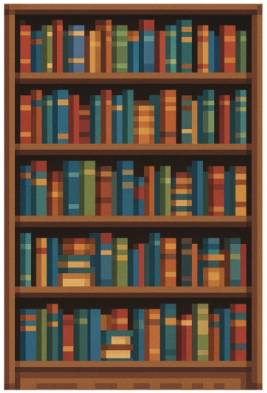
ヒント：Jsで指定できるのはIDだけじゃない

ライブラリとは

コードを書いていると、この機能って前にも書いたなと思う事があります。
また使いそうな機能は、関数にしておいて取っておくとまた使いまわせます。
この使い回しができるプログラムのことを、ライブラリといいます。

自分用につくったプログラムだけをライブラリと呼ぶわけではありません。
世界中のプログラマーが制作し、その多くが無料で公開されています。
このライブラリを活用すれば、自分でコードを書くことなく、複雑な機能を実装できるのです。

複雑なアニメーションや、高度な処理など、ライブラリには便利なものがたくさんあります。
このライブラリを使いこなすことにより、作業期間を大きく短縮することができます。



ライブラリを使ってみよう

以下のコードを書いて、ライブラリをつかってみましょう。
jsフォルダを作って、配布の shuffle-text.js を格納します。

HTML

```
<h1 id="myText">文字をシャッフルするよ</h1>
<script src="js/shuffle-text.js"></script>
<script>
  const text = new ShuffleText(document.querySelector('#myText'));
  text.start();
</script>
```

画面に何かアクションがあれば成功です。
このように、自分でつくると時間がかかりそうな動きも、ライブラリを使えば一瞬です。

ライブラリを使う方法

```
<script src="js/shuffle-text.js"></script>
```

ライブラリを使うには、まずライブラリのファイルを読み込まないといけません。
このように、外部jsファイルを読み込むのと同じようにライブラリを読み込みます。

```
<script>
  const text = new ShuffleText(document.querySelector('#myText'));
  text.start();
</script>
```

この部分は、ライブラリの機能をHTML要素に設定しています。
1行目で参照した要素に、2行目でシャッフルするアクションをつけています。

ライブラリの設定の方法はライブラリによって違います。
サイトなどでライブラリを見つけた際は、設定方法も必ず確認しておきましょう。

CDNとは

CDNとは、サーバにあるライブラリに直接アクセスしてライブラリを使う仕組みです。
ファイルをＤＬしなくて良い、常に最新バージョンを使う事ができるなどの利点があります。

```
<script src="https://cdn.jsdelivr.net/particles.js/2.0.0/particles.min.js"></script>
```

せっかくなので、配布の particle_setting.js を読み込んで動かしてみましょう。
動きが見えやすいように、背景を黒にしています。

```
<script src="js/particle_setting.js"></script>

<style>
  #particles-js {
    background-color: black;
  }
</style>
```

ライブラリ課題

① swiper.js を使って、スライダーを実装してみよう！

https://swiperjs.com/

複数の画像を見せるのに、スライダーは必須とも言えます。
swiper.jsを読み込んで実装してみましょう！

---

② AOS.js を使って、スクロールアニメーションを実装してみよう！

https://michalsnik.github.io/aos/

最近のサイトには必ずと言っていいほど実装されているスクロールアニメーション。
まずはお手軽なAOSを使って実装してみましょう。

JQueryとは

JQueryとは、JavaScriptライブラリの中でも特に人気が高いライブラリです。
JQueryの特徴は、JavaScriptで書くと長くなる処理を、短いコードに簡略化できる点です。
要素の選択、イベント付与、アニメーションなど、より短いコードで実装することができます。

```
// 要素の追加
// js
let target = document.getElementById(“tanaka");
let p = document.createElement("p");
let text = document.createTextNode("こんにちは！！");
p.appendChild(text);
target.appendChild(p);

// JQuery
$("#hoge").append('<p>こんにちは！！ </p>');
```

このように Javascript であれば数行かかる処理が、JQuery であれば1行で書けます。
開発効率を重視する場合、JQueryを使うのが効果的です。

JQueryを使ってみよう

JQueryはライブラリの1種です。
他ライブラリと同じようにファイルをダウンロード、もしくはCDNで使うことができます。

JQuery 公式サイト
https://jquery.com/download/

CDN

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.1/jquery.min.js"></script>
```

HTMLファイルに読み込み処理を記述すれば、JQueryを使えるようになります。

読み込み処理は状況によって変わりますが、headタグ内に書くのが無難です。
また、他ライブラリがJQuery依存なケースもあるので、一番上に書きましょう。

要素の指定

JQueryはCSSと同じようなセレクタ指定で、要素を指定することができます。

CDN

```
// 要素の取得
// id指定
let idText = $('#sakura').text();

// クラス指定
let classText = $('.sakura').text();

// タグ指定
let tagText = $('p').text();
```

その他、子要素の指定や、条件を設定した指定の方法もあります。
どのような指定ができるか調べてみましょう。

メソッド

メソッドは関数のようなもので、指定した要素を操作することができます。
引数にパラメータを記述することで、操作する内容を指定できます。

よく使うメソッド

.css()	指定した要素のCSSを変更
.append()	指定した要素のHTMLの最後に要素を追加
.remove()	指定した要素を削除
.addClass()	指定した要素にクラスを追加
.toggleClass()	指定した要素のクラスを切り替える
.attr()	指定した要素の属性値を取得or変更
.fadeIn()	指定した要素をフェードインさせる
.slideToggle()	指定した要素をスライドアップ・ダウンさせる

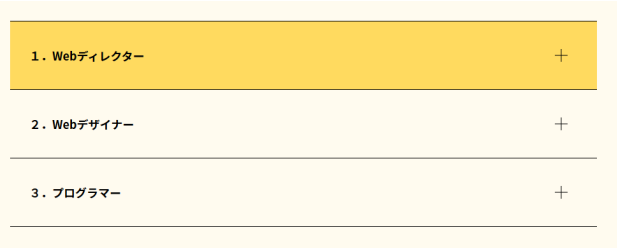
JQueryで簡略化できる処理はたくさんあります。
Jsで機能を実装する場合は、用意されたJQueryメソッドがないか調べましょう。

JQuery課題

① アコーディオンメニューを作ってみよう

https://xd.adobe.com/view/4c49563b-1f6e-475d-ac66-d460c5078a96-191a/

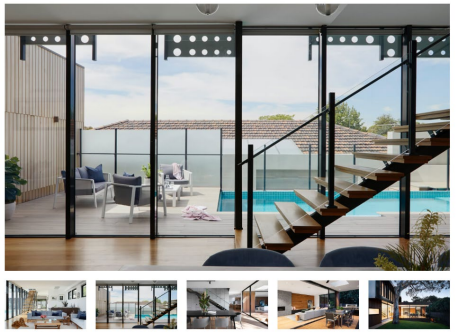
サイトのFAQなどでよく見かける
アコーディオンメニューを実装してみましょう。
それぞれのタイトルをクリックしたら、
詳細が表示されるように実装しましょう。



② 画像の切替セクションを作ってみよう

https://xd.adobe.com/view/bb66e71f-c6d6-4763-8c8d-76a8776815c7-ee35/

下の小さい画像をクリックすると、上の画像がその画像に
切り替わります。
ECサイトなどでよく見られますね。
余裕があれば、自動で切り替わったり、選んでいる画像が
わかるような表示を追加してみましょう。



Re sTart