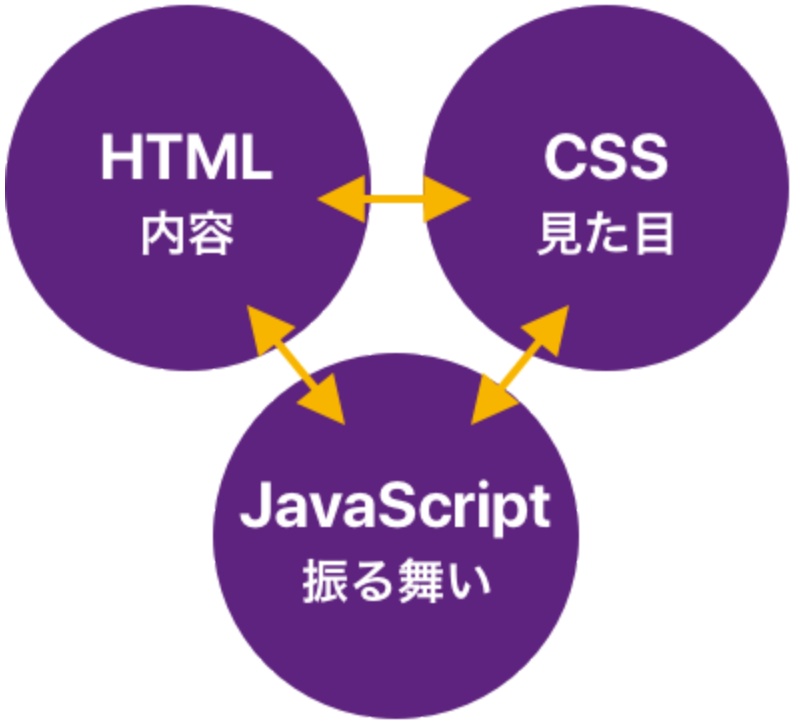
JavaScript

- JavaScriptの役割
- JavaScriptの記述場所
- JavaScriptの文法
- コンソール
- コメント

JavaScriptの役割

3つとも異なる言語でやりとりが必要



JavaScriptの記述場所

1. HTML内に記述
2. 外部jsファイルに記述

1. HTML内に記述

xxx.html

```
        </footer>
        <script>

        // ここにJavaScriptを記述する

        </script>
        </body>
</html>
```

2. 外部jsファイルに記述

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

JavaScriptの文法

3つの記述パターン

1. 命令式 `()`
2. 代入式 `=`
3. グループ `{}`

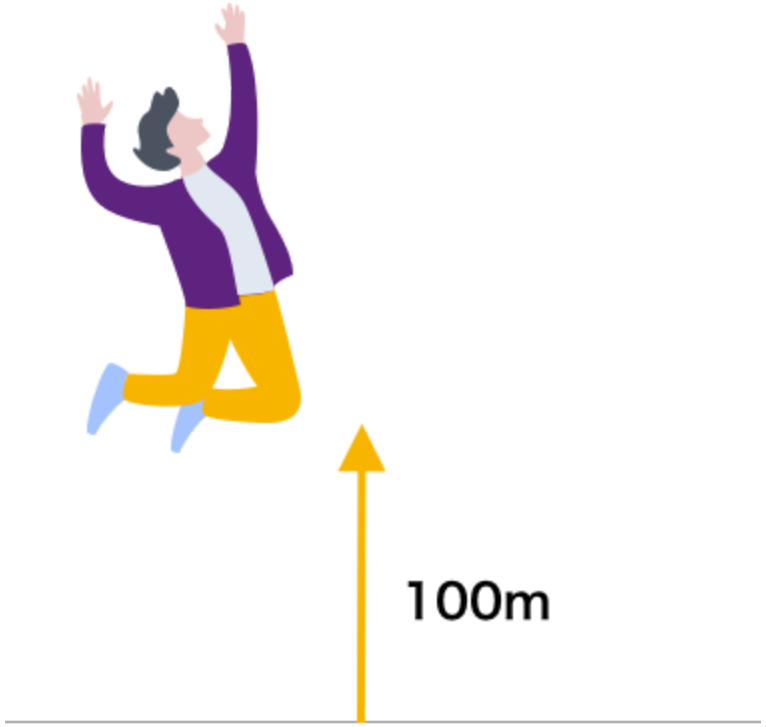
1. 命令文

`( )`

命令を出す

```
命令の名前(引数);
```

```
jump(100);
```



`jump`という命令名があって、`()`の中に値を入れるとその分だけ実行

引数とは命令に渡す値(パラメーター)

```
alert('alertのテスト');
```

- 記述は半角英数
- セミコロンが文の区切りになる

2. 代入式

`=`

何らかの値を保存させる

```
要素.属性 = 値;
```

- `=` の前後に半角スペースを入れてもOK

> プログラミングの = (イコール) は 左辺に右辺の値を保存すること

```
document.querySelector('h1').style.color = '#FF0000';
```

この文を日本語に訳すと、

HTML `document` からタグ `querySelector` を取り出して、CSS `style` の文字色 `color` を `#FF0000` に変更する

ドットシンタックス

オブジェクトと命令を .(ドット)でつなげて記述する

```
document.querySelector('h1').style.color = '#FF0000';
```

3. グループ

`{}`

処理をグループにする場合に使う

```
命令(){
    処理1
    処理2
    処理3
};
```

`{}` の中身はインデントする習慣をつけてください

```
function setColor(){
    document.querySelector('h1').style.color = '#FF0000';
    document.querySelector('h1').style.backgroundColor = '#DDDD';
}
```

その他

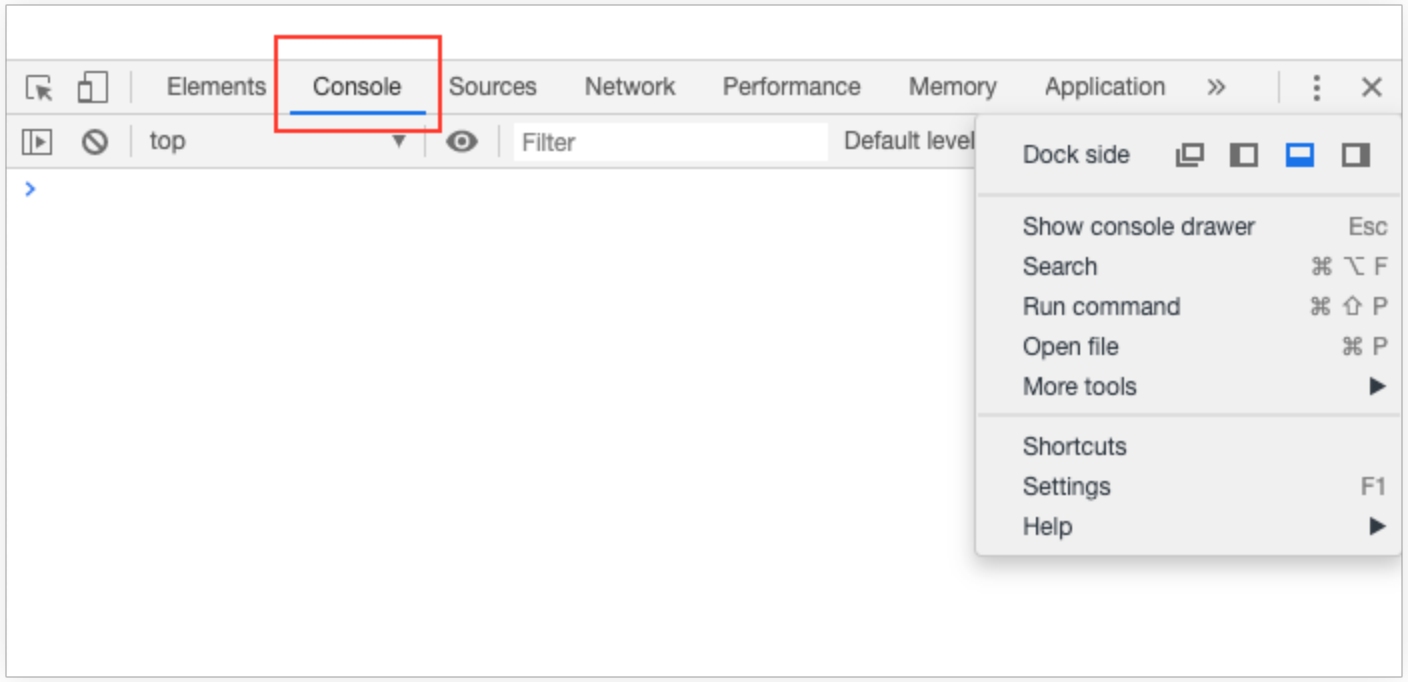
コンソール

JavaScriptで書いたソースの内容を確認する方法

コンソールの出し方

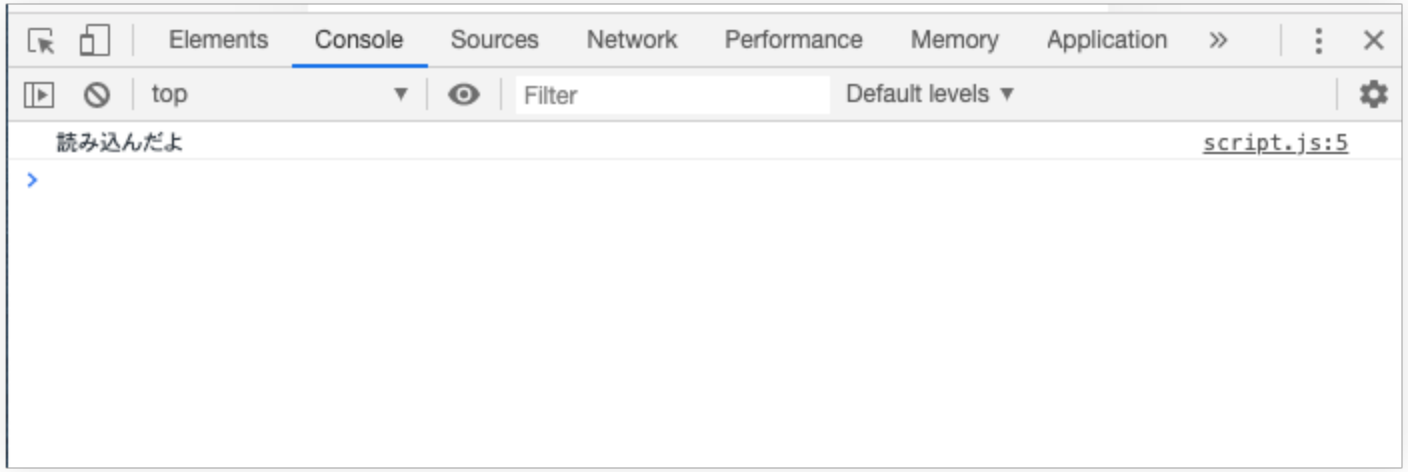
1.ブラウザ上で右クリックして `検証`

2. `Console` タブを選択

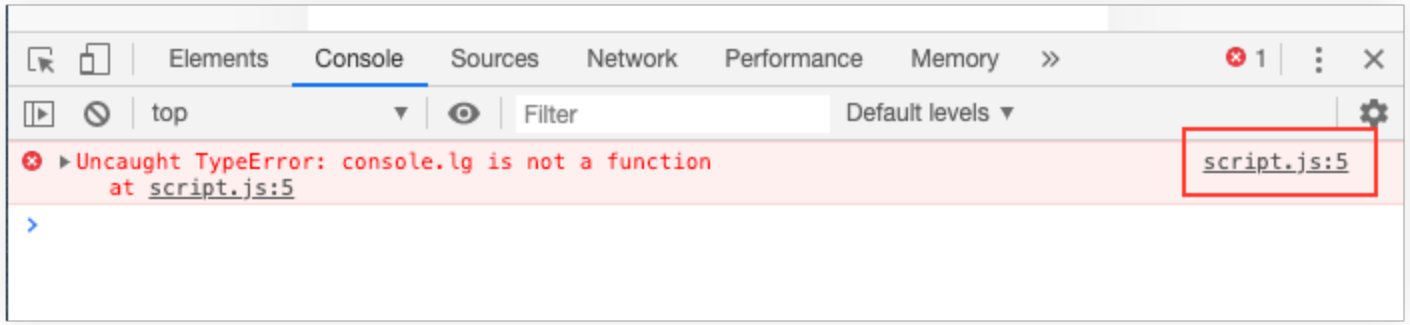


コンソールの実行

```
console.log('読み込んだよ');
```



コンソールでエラー確認



エラーの場合でも確認ができる。

どのファイルの何行目がエラーか教えてくれる。

コメント

コメントの書き方

コメントとはプログラミング内に記述する「メモ書き」のようなもの

```
//  1行のコメント
```

```
/*
複数行コメント
複数行コメント
*/
```

変数

変数の宣言

変数とはデータの保存
固有の名前を与えて、一定期間記憶し必要なときに利用できるようにする

```
var colorValue; // 変数colorValueを宣言
```

> JavaScriptの新しいバージョン(ES6)では、値を変更できない `const` と 参照範囲を有効にできる `let` が加わった

変数を宣言し値を代入

変数を命名して宣言する。変数に値を保存する。

```
var colorValue; // 変数colorValueを宣言

colorValue = '#FF0000'; // 変数colorValueに値 '#FF000' を代入
```

> 変数の保存はパソコンのメモリにデータを保存するイメージ

変数宣言と代入を同時に

```
var color = '#FF0000'; // 変数colorを宣言し、値 '#FF0000' を代入
```

変数に保存できる値の種類

- 数値
- 文字列
- 真偽値
- Null

数値　の保存

値に数字をそのまま記述して保存すると、数値として計算処理できる

```
var numA = 3; // 変数numAに数値3を代入
var numB = 6; // 変数numBに数値6を代入
var result = numA + numB; //変数numAと変数numBを足す
console.log(result);
// 結果は9
```

文字列

値を `"` ダブルクォーテーションまたは `'` シングルクォーテーション で囲んで代入すると、文字列として処理される

```
var message = 'Hello';
var numA = '3'; // 変数numAに文字列3を代入
var numB = '6'; // 変数numBに文字列6を代入
var result = message + numA + numB;
console.log(result);
// 結果はHello36  すべて文字として連結される
```

`"` ダブル、 `'` シングルどちらでも使えるが、必ず開始と閉じを同じにする。JavaScriptは、HTMLと同時に記述することがあるので、 `'` シングルを推奨される。（HTMLのタグ内は必ず `"` ダブルで書くため）

真偽値

真(true)と偽(false)の2種類の値だけを扱う

```
var isFlag = true; // 変数myFlagに真(true)を代入
```

Null

データがない、返す値がないことを表す

```
var myTest = null; // 変数myTestは空っぽ
```

関数 function

関数の作り方

関数とは記述をグループ化して、何度でも繰り返し使えるようにすること。

自分で関数名をつける

```
function 関数名(引数){
    // ここに処理内容を記述する
}
```

関数の実行

関数を実行するには命名した「関数名」に`()`をつけると、関数内の文章が実行される。

```
関数名();
```

関数の定義と実行

自分で関数の名前を命名する。半角英数であれば特に命名のルールは無いが、慣習として命令の意味が伝わる単語繋げると良い。単語区切りは２単語目以降を大文字にすると読みやすくなる。

```
// 関数の定義
function changeStyle(){
    document.querySelector('#ttl').style.color = '#FF0000';
}

// 関数の実行
changeStyle();
```

引数

引数とは

引数とは関数内で使用する変数。実行時に関数の中へ値を受け渡すことで、関数内処理ができる。

```
// 関数の定義
function changeStyle(aColor){
    document.querySelector('#ttl').style.color = aColor;
}

// 関数の実行
changeStyle('#00FF00');
```

引数は`,`カンマ区切りで複数受け渡すことができる。

return

関数内で `return` を使うと処理された値を返す（戻す）ことができる。

```
// 関数の定義
function calcNum(aNum1, aNum2){
    var result = aNum1 + aNum2;
    return result;
}

// 関数の実行
var resultNum = calcNum(2,3);

// 実行結果の確認
alert(resultNum); // 結果は5
```

グローバル変数とローカル変数

変数は宣言された場所によって **スコープ（参照できる範囲）** がある。

グローバル変数

関数`{}`の外側で宣言された変数は**グローバル変数**と呼ばれ関数内外で参照することができる。

```
// グローバル変数
var colorValue = '#FF0000';

//  関数の定義
function changeStyle( ){
  document.querySelector('#ttl').style.color = colorValue;
}

//関数の実行
changeStyle();
```

ローカル変数

関数`{}`の内側で宣言された変数は**ローカル変数**と呼ばれ関数内でしか参照することができない。

```
function changeColor( ){

  // ローカル変数
  var colorValue = '#FF0000';
  document.querySelector('#ttl').style.color = colorValue;
}

alert(colorValue); // エラー 変数"colorValue"は見つからない
```

グローバル変数とローカル変数の違い

- グローバル変数はどこからでも参照できるがメモリに残り続けるので負荷が高い。
- ローカル変数は参照範囲が絞られる為、一時的にメモリを消費して実行が終わると破棄されるので負荷が低い。

条件分岐

if

```
var today = sunny; //変数todayにsunnyを保存

// もしtodayがsunnyだったら
if(today == sunny){
  // 条件を満たす
  console.log('今日は晴れ');
}
```

if else

```
var today = rainy; //変数todayにrainを保存

// もしtodayがsunnyだったら
if(today == sunny){
  // 条件を満たす
  console.log('今日は晴れ');
} else {
  //条件を満たさない
  console.log('今日は晴れ以外');
}
```

if else if

```
var today = cloudy; //変数todayにsunnyを保存

if(today == sunny){
  // 条件1を満たす
  console.log('今日は晴れ');
} else if(today == rainy) {
  //条件2を満たす
  console.log('今日は雨');
} else{
  //条件1,2を満たさない
  console.log('今日は晴れ/雨以外');
}
```

数値による条件分岐

if else

elseはifの条件に当てはまらないものが対象となる

```
var now = new Date(); // 日時を取得
var hour = now.getHours(); // 現在時間を取得
if(hour < 12){
  document.write('現在の時刻は午前中です');
} else {
  document.write('現在の時刻は午後です');
}
```

条件式

演算子	意味
>	より大きい
<	より小さい
>=	以上（より大きいか等しい）
<=	以下（より小さいか等しい）
===	等しい
!==	等しくない

```
var now = new Date();
var hour = now.getHours();
var month = now.getMonth();
month = month + 1; // 0はじまりなので1足す

if(month <= 2 || month === 12){
    console.log('冬です');
    document.write('冬です');
} else if( month <= 5) {
    console.log('春です');
    document.write('春です');
} else if( month <= 8){
    console.log('夏です');
    document.write('夏です');
} else {
    console.log('秋です');
    document.write('秋です');
}
```

複数条件による分岐

論理演算子

演算子	意味
&&	AND
||	OR
!	NOT

論理和　|| (or)

```
var now = new Date(); // 日時を取得
var hour = now.getHours(); // 現在時間を取得
if(hour >= 14 || hour< 16){
  document.write('現在の時刻は14時か16時のどちらかです');
}
```

論理積 && (and)

```
var now = new Date(); // 日時を取得
var hour = now.getHours(); // 現在時間を取得
if(hour >= 14 && hour< 16){
  document.write('現在の時刻は14時から16時の間です');
}
```

否定！(not)

```
var now = new Date(); // 日時を取得
var hour = now.getHours(); // 現在時間を取得
if(hour !== 14){
  document.write('現在の時刻は14時ではありません');
}
```