JavaScript 1 Day 講習 (2017.12.17)

共愛学園前橋国際大学 小柏伸夫

Nobuo Ogashiwa, Ph.D

Associate Professor
Department of International Social Studies
KYOAI GAKUEN UNIVERSITY
Email: ogashiwa@c.kyoai.ac.jp, ogashiwa@wide.ad.jp

JavaScriptとは

- JavaScript (ECMAScript)
  - プログラミング言語の一種
  - 様々なウェブブラウザ上で動作、ウェブブラウザ無しでも動作
  - 動作が軽快、学びやすい、応用の幅も広い

- 歴史
  - 当初NetScape社がJavaScriptを開発しNetscapeNavigatorに搭載
  - ECMAScriptとして標準化
  - 後に他社のブラウザもECMAScriptに準拠
  - Apple社SafariではAdobe Flashを捨てJavaScriptの普及を後押し

プログラミング言語の学習法

- “言語”の学習
  - 読む、書く (、聞く、話す)
- プログラミング言語の学習法
  - 写経
    - お経を書き写して学ぶような方法
      - 既存のプログラムを書き写して動かしてみる
      - 書きながら処理を考えながら読み解く
      - 何度も読み書きを繰り返し、”定跡”を身に付ける
      - **少し書き換えて予想どおり動くかどうか試行錯誤する**とより良い

実行環境

- ブラウザ上で動作させる
  - Firefox, Chrome, Safari, IEなど
  - URLバーでも「javascript:alert(“Hello World”)」等で動作
- node.jsをインストールして動作させる
  - node.jsをインストール後「node sample.js」
- Windowsの標準環境で動作させる
  - 「wscript sample.js」または「cscript sample.js」
- MacOSの標準環境で動作させる
  - 「osascript -l javascript sample.js」

テキストエディタ

- TeraPad (Windows用、フリー)
  - URL: http://www5f.biglobe.ne.jp/~t-susumu/library/tpad.html
  - Windows用で古くから用いられている、シンプル且つ高機能、無料
- Sublime Text (Windows/Mac用、商用)
  - URL: https://www.sublimetext.com
  - 比較的最近のもの、シンプル且つ高機能、有料
- emacs (主にLinux、Windows/Mac用も存在、フリー)
  - 古くからUNIX OS上で利用されているエディタ
- vi (主にLinux、Windows/Mac用も存在、フリー)
  - 古くからUNIX OS上で利用されているエディタ

コーディング時の注意点

- ほぼ全て半角
  - 全角が許される箇所はごく僅か
    - 文字列定数部分、コメント部分程度
  - 空白、タブ文字に注意 (見えない全角が入っているとエラーの元)
  - 記号(ダブルクォーテーション等)にも要注意

- 日本語関連
  - 今回は全てUTF-8で記述

入力してみましょう

- HTMLの入力と確認
  - 「sample0010.html」というファイル名で保存
    - 文字コードはUTF-8で保存
  - ブラウザで開いてみる
    - 右クリック → アプリケーションから開く → Firefox 等

```
<html>
<head>
<meta charset="UTF-8">
<title> sample </title>
</head>

<body>
<!-- HTMLでのコメント行の書き方 -->
<font color="red"> Hello World!! </font>
<br>
<a href="http://www.google.co.jp/"> googleへのリンク </a>


</body>

</html>
```

入力して動かしてみましょう

・ブラウザでalertウィンドウ表示
  ・「sample0020.html」というファイル名で保存
  ・ブラウザで開いてみる

```
<html>
<head>
<meta charset="UTF-8">
<title> sample </title>
</head>
<body>

<script type="text/javascript">
alert("Hello World!!");
// JavaScriptのコメントの書き方
/* JavaScriptのコメントの書き方 */
</script>

</body>
</html>
```

入力して動かしてみましょう

・ブラウザで時計表示

・「sample0030.html」というファイル名で保存

```
<html>
<head>
<meta charset="UTF-8">
<title> sample </title>
</head>

<body>

<script type="text/javascript">
var d = new Date();
var h = d.getHours();
var m = d.getMinutes();
var s = d.getSeconds();
document.writeln(h + "時" + m + "分" + s + "秒");
</script>

</body>

</html>
```

入力して動かしてみましょう

・ブラウザで時計表示 (HTML,JavaScriptファイル分離版)

```
<head>
<meta charset="UTF-8">
<title> sample </title>
</head>
<body>
<script type="text/javascript" src="sample0040.js"></script>
</body>
</html>
```

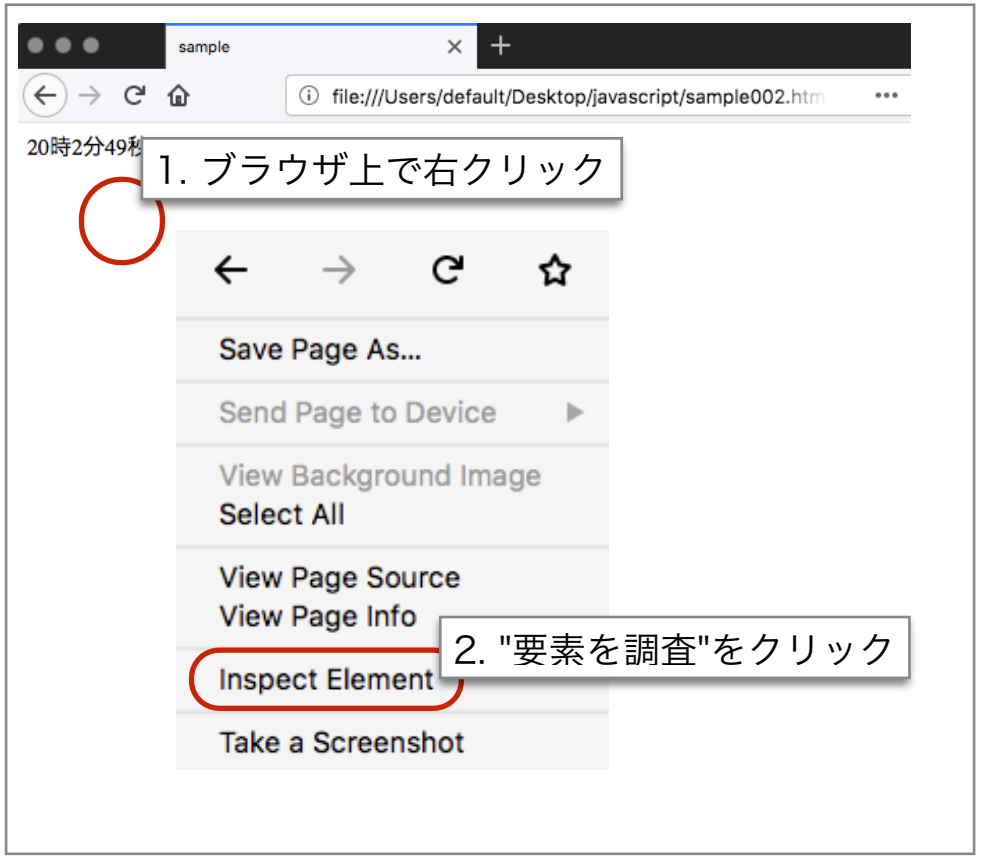
sample0040.html

```
var d = new Date();
var h = d.getHours();
var m = d.getMinutes();
var s = d.getSeconds();
document.writeln(h + "時" + m + "分" + s + "秒");
```

sample0040.js

デバグ(間違い修正)の方法1

- デバグ
  - 問題箇所を見つけて直すこと
- JavaScriptのデバグその1
  - 1. ブラウザ上で"要素を調査"
    - 英語表記: Inspect Element
  - 2. 問題箇所の行数等を確認



- 注意
  - 目視のみでプログラムの問題箇所を探すのは非効率、デバガ等利用推奨

JavaScriptのコメントの書き方

- コメント
  - 書き方
    - 方法1: 「//」から「行末」まで、方法2: 「/*」 から「*/ 」まで
  - チーム開発時にはコメント記述による可読性がとても重要

```
<html>
<head>
<meta charset="UTF-8">
<title> sample </title>
</head>
<body>

<script type="text/javascript">
alert("Hello World!!");
// JavaScriptのコメントの書き方
/* JavaScriptのコメントの書き方 */
</script>

</body>
</html>
```

定数

- 定数
  - 数値や文字を直接的に記述したもの
  - 種類: 数値定数、文字列定数、論理値定数
  - 例:

```
234
3.14
2.123456e10 (10の10乗)
5.2345678e-9 (10の-9乗)
```

数値定数の例

```
'Please enter your ID'
"パスワードが違います"
"234"
```

文字列定数の例

```
true
false
```

論理値定数の例

変数

- 変数
  - 値 (数値、文字列等) を入れる入れ物
  - 自由に名前 (変数名) を付けることができる
  - 変数宣言して利用する(右上図)
    - 宣言無しでも使えるが危険
  - 「変数名=値」で代入 (右下図)

  - その他特徴
    - 変数名に使える文字種
      - 半角、英大文字小文字、数字、一部記号
    - 変数の特殊な状態
      - 未定義:undefined, 空:null
    - 型について
      - JavaScriptでは自動的に型に対応

```
var age = 22;
var name = "tanaka";
var a;
var i, j, k;
var k = 0;
var z = null;
```

変数宣言の例

```
age = 22;
name = "tanaka";
i = 0;
```

変数への値の代入の例

算術演算子 / 文字列連結演算子

- 算術演算子
  - 加減乗除：乗算は「*」、除算は「/」
    - 右図ではaには14、bには10が代入される
  - 剰余演算子：剰余(割った余り)は「%」
    - 右図ではcには1が代入される
  - 1増やす、1減らす
    - インクリメント(++)、デクリメント(--)
- 文字列連結演算子
  - 「+」で文字列を連結
  - 注意
    - +の右に数値を配置可能
    - 自動的に文字列に変換される
    - 明示的に「文字列変換」して連結するのが安全

```
a = 2 + 3 * 4;
b = 5 - 6/3 + 7;
```

加減乗除の例

```
c = 10 % 3;
```

剰余演算子の例

```
a++;
++a:
a--;
--a;
```

インクリメント、デクリメント

```
str = "あいう" + "えお";
a = "おはよう";
msg = a + "ございます";
```

文字列連結演算子の例

代入演算子

- 代入演算子
  - 省略形の書き方
  - 加減乗除等の結果を同じ変数に代入
  - 頻繁に用いられる省略形

```
a = a*3;
a *= 3;

b = b+10;
b += 10;
```

代入演算子

入力して動かしてみましょう

・今後の例題コードの動かし方

```
<html><head> <title> sample </title> </head>
<body>
<script type="text/javascript" src="samplexxxx.js"></script>
</body>
</html>
```

sample0000.html

```
(JavaScriptのコード)
```

JavaScriptのファイル名を
HTMLのタグ内に書く

samplexxxx.js

入力して動かしてみましょう

・様々な演算子

```
str = "あいう" + "えお";
alert(str);

a = "おはよう";
msg = a + "ございます";
alert(msg);
alert(a);

a = 10;
alert(a);

a += 100;
alert(a);

a = 10 % 3;
alert(a);
```

sample0050.js

比較演算子

- 比較演算子を用いた式
  - 論理値(trueまたはfalse)を返し、主にif文(後述)の条件判断で使用

比較演算子	意味
==	左右が等しければtrue、それ以外のときfalse 「x == 5」はxが5のときtrue
>	左が右より大きければtrue、それ以外のときfalse 「x > 5」はxが5より大きいときtrue
<	左が右より小さければtrue、それ以外のときfalse 「x < 5」はxが5より小さいときtrue
>=	左が右以上のときtrue、それ以外のときfalse 「x >= 5」はxが5以上のときtrue
<=	左が右以下の時true、それ以外のときfalse 「x <= 5」はxが5以下のときtrue
!=	左右が等しくなければtrue、等しいときfalse 「x != 5」はxが5でなければtrue
===	左右の「値」と「型」がどちらも一致すればtrue、 それ以外のときfalse

論理演算子

- 基本の論理
  - true(真) または false(偽)
- 論理演算子
  - AND 「&&」
    - 「&&」の左右の式が両方trueのとき、全体をtrueとする
  - OR 「||」
    - 「||」の左右の式のいずれか一方または両方trueのとき、全体をtrueとする
  - NOT 「!」
    - 「!」の後の式の論理を反転する
  - 考え方の例
    - 3の倍数か3の付く数字のときtrue
    - 「 (a%3==0) || (String(a).indexOf('3') != -1)」
    - 主にif文(後述)の条件判断で使用

```
x == 1 && y == 2

x == 1 || y == 2

!(x == 1 && y == 2)
```

論理演算子の例

配列変数と添え字

- 配列変数
  - 入れ物(変数)が列になったもの
  - 複数の値を一括で扱える
  - 「変数名[添え字]」で指定
  - 「添え字」は数値定数でも(数値の)変数でも良い

```
var student = ["tanaka", "sato", "suzuki"];

//以下のように添え字で指定して個別に処理できる
alert(student[0]);
alert(student[1]);
alert(student[2]);
student[0] = "takahashi";
alert(student[0]);
var num = 2;
student[num] = "yoshida";
alert(student[num]);
```

配列変数の例1

条件分岐 (if，if～else ～，if ～ else if ～)

- 条件分岐
  - 制御文とも呼ぶ
  - 条件式の真偽に応じて流れを制御
  - 条件式がtrueなら直後のブロック実行
  - falseなら次の条件式チェックへ
  - 全てfalseならelseへ

  - 書き方のパターン
    - if (式)
    - if (式) else if (式)
    - if (式) else if (式) else
    - if else

```
if (式1) {
  文1-1;
  ・・・
}
```

if文

```
if (式1) {
  文1-1;
  ・・・
} else {
  文e-1;
  ・・・
}
```

if文

```
if (式1) {
  文1-1;
  ・・・
} else if (式2) {
  文2-1;
  ・・・
} else if (式3) {
  文3-1;
  ・・・
} else {
  文e-1;
  ・・・
}
```

if文

条件分岐 (switch)

- switch文
  - switch (変数) { case 値: 文; case ・・・
  - 変数==値 が真ならば：から break まで実行
  - 全て偽ならばdefault以降を実行

  - ポイント
    - if文でも同様の処理は記述可能
    - 可読性向上のため使用されるケースが多い

```
switch (a) {
 case 1: alert("1です");break;
 case 2: alert("2です");break;
 case 3:
  alert("3です");
  alert("素数です");
  break;
 default:
  alert("それ以外です");
  break;
}
```

switch文

繰り返し (while)

- 繰り返し
  - while (式) { 文 }
    - 式が真の間、文を繰り返し実行
  - do {文} while (式)
    - 一度"文"を実行し、式が真の間さらに文を実行

```
while (式1) {
  文1-1;
  ・・・
}
```

while文

- ポイント
  - "ループ" とも呼ぶ
  - 式が偽にならない場合 "無限ループ" と呼ぶ
  - サーバ的な常駐型処理はこの無限ループが処理の中心
  - 常駐せず終了すべきプログラムはループの条件を要確認
  - 無限ループの中はsleep等でCPUを数ミリ秒休ませる

```
do {
  文1-1;
  ・・・
} while (式1)
```

do-while文

繰り返し (for)

- for文
  - 繰り返しの別の書き方
  - for (式1; 式2; 式3) { 文 }
    - step1: 式1を1度実行し
    - step2: 式2が真ならば文を実行
    - step3: 式3を実行
    - step2に戻る (繰り返す)
    - つまりカウンタ的な変数処理を一括記述可能
    - while でも同じ処理を表現可能
    - 登場頻度は極めて高い

```
for (式1; 式2; 式3) {
  文1-1;
  ・・・
}
```

for文

```
for (i=0; i<10; i++) {
  alert("iの値は"+i);
  ・・・
}
```

for文の例

```
i=0
while (i<10) {
  alert("iの値は"+i);
  ・・・
  i++;
}
```

whileでも可能

繰り返しとcontinue, break

- break
  - 右図のように利用
  - breakが実行されるとループを脱出

```
for (i=0;i<100;i++) {
 処理1;
 if(条件) break;
 処理2;
}
```

break

- continue
  - 右図のように利用
  - continueが実行されると
  - i++部分が処理され
  - ループ先頭へ戻る
  - つまり処理2がスキップされる

```
for (i=0;i<100;i++) {
 処理1;
 if(条件) continue;
 処理2;
}
```

continue

繰り返しと条件分岐と配列

・繰り返しで配列の個々の要素にアクセス、条件分岐で処理切り替え
  ・配列を作成
  ・ループ(カウンタ変数を添え字に)で先頭からアクセス
  ・if文でひとつひとつ確認

sample0060.js

```
var list=[2,3,5,7,11,13,17,19,23,29,31,37];

for (i=0; i<list.length; i++) {
    if (String(list[i]).indexOf("3") != -1) {
        alert(i + "番目の素数は" + list[i] + "で、3を含みます");
    }
}
```

注1) String(数値変数) は数値を文字列に変換

注2) 文字列.indexOf(文字) は文字列中に文字が存在しなければ-1を応答

入力して動かしてみましょう

・3の倍数、または、”3”を含む数字のときだけ派手に表示

sample0070.js

```
var count = 1, countmax = 100;
var timer = setInterval(function(){

    if (count%3==0 || String(count).indexOf("3")!=-1) {
        document.getElementById("number").innerHTML =
            "<font color='red' size=30>"+count+"</font>";
    } else {
        document.getElementById("number").innerHTML = count;
    }

    if(count >= countmax) clearInterval(timer);
    count++;
},1000);
```

注1) 点線部分は定型文だと思って気にせずそのまま書きましょう

注2) String(数値変数) は数値を文字列に変換

注3) 文字列.indexOf(文字) は文字列中に文字が
存在しなければ-1を応答

注4) HTMLファイル内に右の赤字部分を追記しておきましょう

```
<html> <head></head>
<body>
<p id="number"> 0 </p>
<script type="text/javascript">
</script>
</body>
</html>
```

sample0070.html

休憩1

- 閑話休題
  - セミコロンについて
    - 正確にはJavaScriptの文法上、セミコロン ";" は必須
    - ところがセミコロン無しでも動くこともある
    - JavaScriptの実行環境が柔軟且つほぼ適切に「セミコロン忘れ」を処理
    - それでも念のためセミコロンを正しく書き入れることが安全
  - 間違えやすい記号について
    - シングルクォート " ' "
    - バッククォート " ` "

関数

- function 関数名 (引数リスト) { 文; 文; return 値; }
  - 処理をまとめて名前を付ける仕組み
  - 関数名で呼び出し
  - 呼び出し時に引数を渡せる
  - 応答時に値が返される

```
function 関数名(引数リスト) {
 文;
 ...
 return 値;
}
```

関数の例

  - ポイント
    - 同じ処理を何度も書きたくない場合に利用
    - 可読性のため利用
      - 長くて複雑な処理に名前を付けてまとめ見通し向上
    - HTML等から呼び出す処理をまとめて名前を付けるために利用

オブジェクト指向
クラス,インスタンス,メッセージ

- オブジェクト指向
  - クラス
    - 抽象的に捉えて記述
    - さらに、共通部分を親クラスで実現
      - 例えば自動車、飛行機等なら推進力、乗員数など
    - 以上のような「オブジェクト」のモデル化の道具
  - インスタンス
    - クラスとしての抽象化モデルを動作時のプログラム内で実体化したもの
  - メッセージ
    - クラス間では処理は「メッセージ」として扱われる

変数型

- 変数型
  - JavaScriptの場合
    - JavaScriptの処理系が自動的に処理
    - 厳密にはリテラルから型を推測
    - 変数には「型情報」と「値」が格納されている
  - 他の言語(例:C言語)の場合
    - 型を厳密に宣言する必要がある
    - 変数は単なるバイト列
    - 当該バイト列を別の型と見なす場合キャストする

```
var a;
a = 1;
alert(a);
a = "あいうえお";
alert(a);
a = true;
alert(a);
```

変数型

for in文

- for in 文
  - 繰り返しの一種でオブジェクトを対象とする
  - 「var 変数 in オブジェクト」
    - オブジェクトのプロパティが
    - 変数に順次格納され
    - ループする

```
var obj={a:1,b:2,c:3};
for (var k in obj) {
 alert(k);
 alert(obj[k]);
}
```

for-in文

書く/読む

- 実際の開発の場面
  - チームでコードを書く
  - 自分で書いた古いコードを読み直す
- 重要な点
  - 他の人にも読めることを意識
  - 全て忘れた将来の自分でも読めることを意識
- 実際
  - 技術的に高度すぎるトリッキーなコードは良く検討してから
  - コメントを書く
  - バージョン管理/共有のシステムも使う(先祖返りを防ぐ)

エラーハンドリング

- try〜catch〜
  - tryのブロック内でエラーが発生すると
  - catchブロックに処理が飛ぶ
  - その際に(右図ならerr変数で)
  - エラーメッセージ等を受け取れる
  - 例外処理 exception handling とも呼ぶ

```
try {
  この内部でエラーが発生すると
} catch (err) {
  ここのコードが実行される
}
```

try-catch

- throw
  - throw 式; で、式をexceptionとして投げる(throw)
  - 外側にtryがあれば「式」を受け取れる (catch)

```
throw 式;
```

throw

- ポイント
  - 古い実用プログラムは大量のエラー判定コードから成ることが多い
  - tryの仕組みでエラー判定を一括化可能

入力して動かしてみましょう

・エラーハンドリング

```
function func1() {
    alert("func1 begin");
    func2();
    alert("func1 end");
}

function func2() {
    alert("func2 begin");
    func3();
    alert("func2 end");
}

function func3() {
    alert("func3 begin");
    throw "dummy error";
    alert("func3 end");
}

try {
    func1();
} catch (err) {
    alert(err);
}
```

sample0080.js

デバグの方法

- デバッガを使う
  - 最も効率的なデバッグ方法
- printを使う
  - 内部処理の流れや、変数の変化を追うことができる
  - デバッガが使えない場合には効率的
- コメントアウトを使う
  - コンパイルエラーや文法エラー箇所の絞り込みが可能
  - エラー表示の行数以外の場所に原因がある場合には効率的
- 目で追う
  - 見た目で把握できないエラーの場合にはとても非効率

デバグしてみましょう: 間違い探し1

```
var nums = [911, 326, 145, 638, 576, 319, 820, 915, 155, 74, 734, 872,
            7, 306, 305, 391, 368, 302, 630, 714, 79, 730, 624, 285, 30,
            186, 18, 766, 381, 452, 972, 847, 862, 668, 116, 21, 328, 217,
            462, 486, 331, 865, 709, 564, 597, 620, 503, 133, 762, 909,
            209, 763, 908, 756, 978, 451, 553, 111, 487, 612, 229, 396,
            226, 71, 8, 404, 198, 613, 289, 728, 402, 437, 924, 56, 710,
            676, 664, 86, 698, 467, 531, 996, 50, 563, 538, 479, 822, 724,
            60, 917, 134, 161, 23, 855, 158, 672, 448, 125, 735, 657];
var sum = 0;
var avg = 0;
for (i=0; i<nums.length; i++) {
    sum += nums[1];
}
avg = sum / nums.length;
alert(avg);
```

sample0090.js

デバグしてみましょう: 間違い探し2

```
var nums = [911, 326, 145, 638, 576, 319, 820, 915, 155, 74, 734, 872,
            7, 306, 305, 391, 368, 302, 630, 714, 79, 730, 624, 285, 30,
            186, 18, 766, 381, 452, 972, 847, 862, 668, 116, 21, 328, 217,
            462, 486, 331, 865, 709, 564, 597, 620, 503, 133, 762, 909,
            209, 763, 908, 756,  978, 451, 553,  111, 487, 612, 229, 396,
            226, 71, 8, 404, 198, 613, 289, 728, 402, 437, 924, 56, 710,
            676, 664, 86, 698, 467, 531, 996, 50, 563, 538, 479, 822, 724,
            60, 917, 134, 161, 23, 855, 158, 672, 448, 125, 735, 657];
var sum = 0;
var avg = 0;
for (i=0; i<nums.length; i++) {
    sum += nums[i];
}
avg = sum / nums.length;
alert(avg);
```

sample0100.js

デバグしてみましょう: 間違い探し3

```
var nums = [911, 326, 145, 638, 576, 319, 820, 915, 155, 74, 734, 872,
            7, 306, 305, 391, 368, 302, 630, 714, 79, 730, 624, 285, 30,
            186, 18, 766, 381, 452, 972, 847, 862, 668, 116, 21, 328, 217,
            462, 486, 331, 865, 709, 564, 597, 620, 503, 133, 762, 909,
            209, 763, 908, 756, 978, 451, 553, 111, 487, 612, 229, 396,
            226, 71, 8, 404, 198, 613, 289, 728, 402, 437, 924, 56, 710,
            676, 664, 86, 698, 467, 531, 996, 50, 563, 538, 479, 822, 724,
            60, 917, 134, 161, 23, 855, 158, 672, 448, 125, 735, 657];
var sum = 0;
var avg = 0;
for (i=0; i<nums.1ength; i++) {
    sum += nums[l];
}
avg = sum / nums.1ength;
a1ert(avg);
```

sample0110.js

プログラミング言語の実践1: 入力と出力

- プログラムの基本
  - 通常は必ず入力または出力、または両方が存在
  - 何も入力も出力もしないプログラムは電気を熱にするだけ

  - プログラムを考えるときの基本
    - 何が入力され
    - どのように処理し
    - 何を出力するのか

- 基本的な入出力
  - 文字情報の入出力：標準入出力、ファイル入出力、通信ソケットなど
  - バイナリの入出力：画像処理や映像処理など

標準入出力

- 標準入出力
  - standard I/O (Input Output)
  - 標準的なテキスト入出力方法が決まっているプログラミング言語が多い
    - C言語の標準出力: printf("Hello World!!\n");
    - C言語の標準入力: scanf("%c", &a);

- JavaScriptの場合
  - 標準入出力は無い
  - 「ホスト環境」(ブラウザやNode.jsなど) 毎にテキスト情報出力が異なる
  - 出力例
    - Node.js: console.log("Hello World!!");
    - Windows: WScript.Echo("Hello World!!");
    - ブラウザ(BOM): alert("Hello World!!");

ブラウザとの入出力

- 入力
  - prompt, getElementById().innerHTML など
- 出力
  - document.write, getElementById().innerHTML など

```
<html>
<head></head>
<body>
<script type="text/javascript">
var str = "";
str = prompt("名前を入力してください");
document.write("あなたの名前は「" + str + "」さんです。");
</script>
</body>
</html>
```

sample0120.html

文字列処理

- 文字列処理
  - 文字列連結、文字列置換、文字列検索、文字列比較、文字列分割
    - "abc" + "def"
    - 文字列.indexOf("文字")　(無ければ-1、見つかれば0以上の位置を応答)
    - 文字列==文字列、文字列===文字列
  - 文字列と数値の相互変換
    - String(数値)
    - Number(文字列)
  - RegEx (Regular Expression 正規表現)

数値処理

・平均値を計算

```
var nums = [911, 326, 145, 638, 576, 319, 820, 915, 155, 74, 734, 872,
            7, 306, 305, 391, 368, 302, 630, 714, 79, 730, 624, 285, 30,
            186, 18, 766, 381, 452, 972, 847, 862, 668, 116, 21, 328, 217,
            462, 486, 331, 865, 709, 564, 597, 620, 503, 133, 762, 909,
            209, 763, 908, 756, 978, 451, 553, 111, 487, 612, 229, 396,
            226, 71, 8, 404, 198, 613, 289, 728, 402, 437, 924, 56, 710,
            676, 664, 86, 698, 467, 531, 996, 50, 563, 538, 479, 822, 724,
            60, 917, 134, 161, 23, 855, 158, 672, 448, 125, 735, 657];
var sum = 0;
var avg = 0;
for (i=0; i<nums.length; i++) {
    sum += nums[i];
}
avg = sum / nums.length;
alert(avg);
```

sample0130.js

数値処理

・分散と標準偏差を計算

```
var nums = [911, 326, 145, 638, 576, 319, 820, 915, 155, 74, 734, 872,
            7, 306, 305, 391, 368, 302, 630, 714, 79, 730, 624, 285, 30,
            186, 18, 766, 381, 452, 972, 847, 862, 668, 116, 21, 328, 217,
            462, 486, 331, 865, 709, 564, 597, 620, 503, 133, 762, 909,
            209, 763, 908, 756, 978, 451, 553, 111, 487, 612, 229, 396,
            226, 71, 8, 404, 198, 613, 289, 728, 402, 437, 924, 56, 710,
            676, 664, 86, 698, 467, 531, 996, 50, 563, 538, 479, 822, 724,
            60, 917, 134, 161, 23, 855, 158, 672, 448, 125, 735, 657];
var sum = 0;
var avg = 0;
var variance = 0;
for (i=0; i<nums.length; i++) {
    sum += nums[i];
}
avg = sum / nums.length;
alert("平均は"+avg);

for (i=0; i<nums.length; i++) {
    variance = variance + Math.pow(nums[i]-avg, 2);
}
variance = variance / nums.length
alert("分散は"+variance);
alert("標準偏差は"+Math.sqrt(variance));
```

sample0140.js

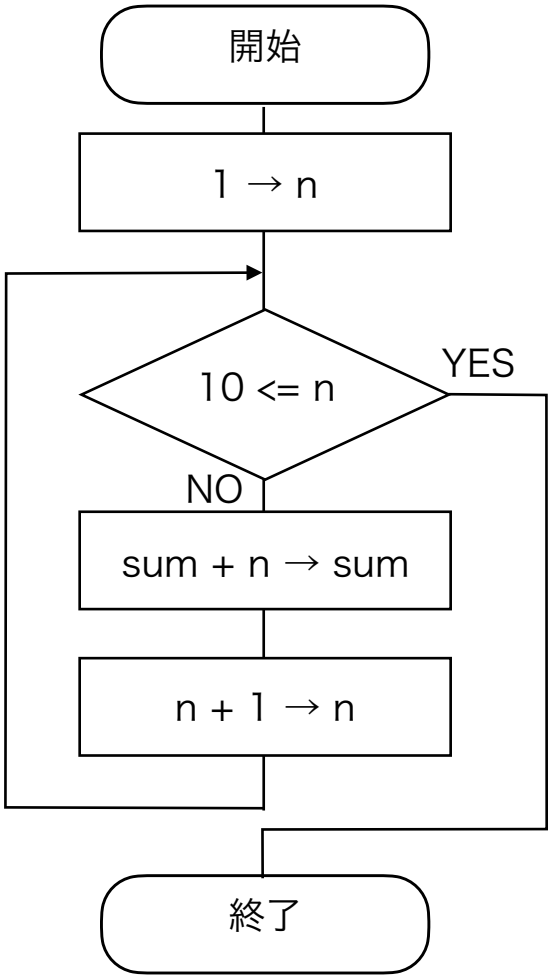
プログラミング言語の実践2: アルゴリズム

- 情報系科目の体系
  - 入門(プログラミング関係)
    - 理論: アルゴリズム
    - 実践: 各種プログラミング言語

アルゴリズムとは

- アルゴリズム
  - 処理の手順
  - フローチャートで記述
  - 分割単位は
    - 入力
    - 出力
    - 分岐
    - 処理

  - 例:
    - 右のフローチャート
    - 1〜9までの数の総和を求めるアルゴリズム



ソート(並べ替え)のアルゴリズム

・バブルソートを行ってみる

sample0150.js

```
function printall(a) {
    for (k=0; k<a.length; k++) {
        document.write(" "+a[k]+", ");
    }
    document.write("<br>\n");
}

var nums = [911, 326, 145, 638, 576, 319, 820, 915, 155, 74, 734, 872,
            7, 306, 305, 391, 368, 302, 630, 714, 79, 730, 624, 285, 30,
            186, 18, 766, 381, 452, 972, 847, 862, 668, 116, 21, 328, 217,
            462, 486, 331, 865, 709, 564, 597, 620, 503, 133, 762, 909,
            209, 763, 908, 756, 978, 451, 553, 111, 487, 612, 229, 396,
            226, 71, 8, 404, 198, 613, 289, 728, 402, 437, 924, 56, 710,
            676, 664, 86, 698, 467, 531, 996, 50, 563, 538, 479, 822, 724,
            60, 917, 134, 161, 23, 855, 158, 672, 448, 125, 735, 657];

var swapflag = false;
do {
    swapflag = false;
    printall(nums);
    for (i=0; i<nums.length-1; i++) {
        if (nums[i]>nums[i+1]) {
            tmp = nums[i];
            nums[i] = nums[i+1];
            nums[i+1] = tmp;
            swapflag = true;
        }
    }
} while (swapflag);
```

探索のアルゴリズム

・線形探索

```
var target = 111;
var nums = [911, 326, 145, 638, 576, 319, 820, 915, 155, 74, 734, 872,
            7, 306, 305, 391, 368, 302, 630, 714, 79, 730, 624, 285, 30,
            186, 18, 766, 381, 452, 972, 847, 862, 668, 116, 21, 328, 217,
            462, 486, 331, 865, 709, 564, 597, 620, 503, 133, 762, 909,
            209, 763, 908, 756, 978, 451, 553, 111, 487, 612, 229, 396,
            226, 71, 8, 404, 198, 613, 289, 728, 402, 437, 924, 56, 710,
            676, 664, 86, 698, 467, 531, 996, 50, 563, 538, 479, 822, 724,
            60, 917, 134, 161, 23, 855, 158, 672, 448, 125, 735, 657];

for (i=0; i<nums.length; i++) {
    if (nums[i] == target) {
        alert(String(i)+"番目の値が"+nums[i]+"です。");
    }
}
```

sample0160.js

探索のアルゴリズム

・二分探索 (ソート済みが前提) (再帰関数の概念を用いても良い)

```
var target = 111;
var nums = [7,8,18,21,23,30,50,56,60,71,74,79,86,111,116,125,133,134,145,155,158,
            161,186,198,209,217,226,229,285,289,302,305,306,319,326,328,331,368,
            381,391,396,402,404,437,448,451,452,462,467,479,486,487,503,531,538,
            553,563,564,576,597,612,613,620,624,630,638,657,664,668,672,676,698,
            709,710,714,724,728,730,734,735,756,762,763,766,820,822,847,855,862,
            865,872,908,909,911,915,917,924,972,978,996];

var a,b,c,d;
a = 0;
d = nums.length-1;
b = Math.floor((a+d)/2);
c = b+1;
do {
    if (nums[a]<=target && target<=nums[b]) {
        a = a;
        d = b;
    } else {
        a = c;
        d = d;
    }
    b = Math.floor((a+d)/2);
    c = b+1;
} while (a!=b && c!=d);

if (nums[a]==target) alert(String(a)+"番目の値が"+nums[a]+"です。");
else alert(String(c)+"番目の値が"+nums[c]+"です。");
```

sample0170.js

休憩2

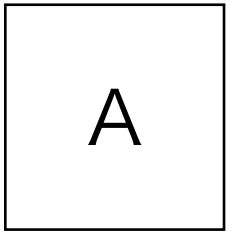
・ブラウザ上の画像操作 (次の話題に向けて)

・画像ファイルを3つ用意して以下のコードを作成・実行

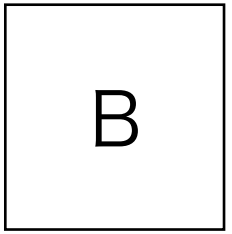
```
<html>
<head> <title>sample</title> </head>
<body>

<img id="target" onclick="func()" src="sushi.png" width="320">

<script type="text/javascript">
var cnt = 0;
var imgfiles = ["a.png", "b.png", "c.png"];
function func() {
  cnt++;
  document.getElementById("target").src = imgfiles[cnt % 3];
  document.getElementById("target").width *= 1.03;
  if (cnt % 30 == 0) {
    document.getElementById("target").width = 320;
  }
}
</script>

</body>
</html>
```

sample0175.html

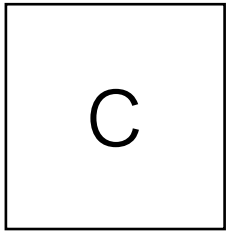
a.png



b.png



c.png

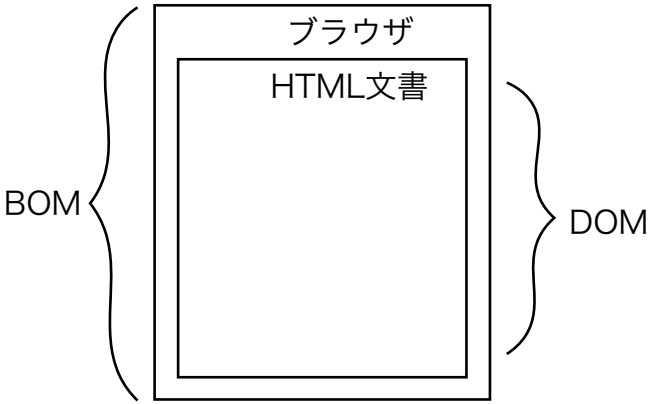


JavaScriptの実践: ウェブブラウザ連携

- JavaScript実行環境
  - 前述のとおり: Node.jsや各種ブラウザ等々
  - ブラウザ上で動作するというのは大きな強み
- 従来の言語環境
  - UI(User Interface)の実現は工夫やライブラリが必要
  - Xlib, Tcl/Tk, など
- JavaScriptのウェブブラウザとの親和性
  - 歴史的経緯から、JavaScriptとウェブブラウザは強く連携
  - BOM, DOMの概念、それらのオブジェクトライブラリを介して
  - JavaScriptはウェブブラウザ上で様々なことが行える
    - => UIが既に準備されているとも言える

BOM, DOM

- Browser Object Model (BOM)
  - ブラウザの情報に関するオブジェクトモデル
  - ブラウザを操作、情報取得する枠組み
- Document Object Model (DOM)
  - HTML等の文書に関するオブジェクトモデル
  - 文書を操作、情報取得する枠組み




```
// BOMの使用例
alert("ブラウザのウィンドウの高さは" + window.innerHeight);
alert("ブラウザのウィンドウの幅は" + window.innerWidth);
alert("ページのURLは" + location.href);
window.open() // ブラウザウィンドウをオープン

// DOMの使用例
document.write("文書に書き込むにはこのメソッド")
var a = document.getElementById("myid");
```

jQuery

- jQueryとは
  - JavaScriptのライブラリの一種
  - 主にDOMベースで文書操作、ブラウザ上の各種イベント処理が可能
    - DOM利用と同様の処理ができるがDOM直接利用よりも便利なライブラリ
    - 非常に多数のウェブサイトで利用されている

```
<html><head></head><body>

<!-- <script type="text/javascript" src="jquery-1.11.0.min.js"></script> -->
<script type="text/javascript"
 src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.0/jquery.min.js"></script>

<div id="test1">str</div>
<div id="test2">str</div>
<div id="test3">str</div>
<div id="test4">str</div>
<div id="test5">str</div>
<script type="text/javascript">
for (i=1;i<=5;i++) {
  $("#test"+i).text("Hello World! No."+i);
}
</script>

</body></html>
```

sample0180.html

オブジェクト指向方法論

- オブジェクト指向方法論
  - 「もの(オブジェクト)」でモデル化する考え方
- オブジェクト指向設計
  - オブジェクト指向設計、その定跡
    - singletonパターンやproxyパターンなど
- オブジェクト指向プログラミング言語
  - オブジェクト指向方法論/設計に基づいてコーディングしやすい言語
  - オブジェクト指向プログラミング言語の例
    - C++、Java、JavaScript、SmallTalk、Rubyなど

クラス、オブジェクトを作る

- クラス、オブジェクト作成の例
  - 学生(Student)クラスを作成し、3つのインスタンスを作成
  - GetAllString()メソッドで、内部の属性を一括して文字列化

```
function Student(param) {
    this.name = param.name;
    this.id = param.id;
    this.age = param.age;
    this.GetAllString = function() {
        return String(this.id) + ":" + this.name + ":" + this.age;
    }
}

var students = [new Student({name:"tanaka",id:12345, age:20}),
                new Student({name:"suzuki",id:11111, age:21}),
                new Student({name:"satoh", id:22222, age:22})];

for (i=0;i<students.length;i++) {
    document.write(students[i].GetAllString()+"<br>");
}
```

sample0190.js

様々なライブラリ

- node.js
  - https://nodejs.org/en/
- Backbone.js
  - http://backbonejs.org
- jQuery
  - https://jquery.com
- jQueryUI
  - https://jqueryui.com
- express.js
  - https://expressjs.com
- D3.js
  - https://d3js.org
- enchant.js
  - http://enchantjs.com/download/

その他、多数のライブラリが存在

定跡:forとifの組み合わせ

```
var nums = [911, 326, 145, 638, 576, 319, 820, 915, 155, 74, 734, 872,
      7, 306, 305, 391, 368, 302, 630, 714, 79, 730, 624, 285, 30,
      186, 18, 766, 381, 452, 972, 847, 862, 668, 116, 21, 328, 217,
      462, 486, 331, 865, 709, 564, 597, 620, 503, 133, 762, 909,
      209, 763, 908, 756, 978, 451, 553, 111, 487, 612, 229, 396,
      226, 71, 8, 404, 198, 613, 289, 728, 402, 437, 924, 56, 710,
      676, 664, 86, 698, 467, 531, 996, 50, 563, 538, 479, 822, 724,
      60, 917, 134, 161, 23, 855, 158, 672, 448, 125, 735, 657];

var max = 0;
var i;
for (i=0; i<nums.length; i++) {
    if (max < nums[i]) max = nums[i];
}

alert(max);
```

sample0200.js

定跡:forとforの組み合わせ

- 30 x 30 までのかけ算の表を作成

```
<html>
<head></head>
<body>
<script type="text/javascript" src="ttable.js"></script>
</body>
</html>
```

sample0210.html

```
document.write("<table border='1'>");
for (i=1; i<=30; i++) {
    document.write("<tr>");
    for (j=1; j<=30; j++) {
        document.write("<td>" + (i*j) + "</td>");
    }
    document.write("</tr>");
}
document.write("</table>");
```

sample0210.js

定跡:まずはコメント (// /** /)

```
// print関数 配列を受け取り表示
function printall(a) {
    for (k=0; k<a.length; k++) {
        document.write(" "+a[k]+", ");
    }
    document.write("<br>\n");
}

var nums = [911, 326, 145, 638, 576, 319, 820, 915, 155, 74, 734, 872];

// 全体のループdo-whileループ、swap発生フラグ
var swapflag = false;
do {
    swapflag = false;
    printall(nums);  // 毎回print
    for (i=0; i<nums.length-1; i++) { // for 0からnums.length-1まで iとi+1の中身比較
        if (nums[i]>nums[i+1]) { // if i+1の方が小さい場合 swap処理 flag=true
            tmp = nums[i];
            nums[i] = nums[i+1];
            nums[i+1] = tmp;
            swapflag = true;
        }
    }
} while (swapflag);  // flag==trueの間ループ
```

赤字部分が最初に書き始めた部分

定跡: 様々なライブラリを使う

- ライブラリの使用方法は必要に応じて都度習得
  - 有名なライブラリは日本語情報(サイト、書籍)も多数
  - ライブラリの使い方
    - ダウンロードして「〜.js」ファイルを参照するタイプ
    - ダウンロードせずネット経由で「http://〜/〜.js」を参照するタイプ
    - 各ライブラリの公式サイト、参考書等で使い方を確認

良いプログラムとは？

- プログラムの判断材料
  - 期待通りに動作する (バグが少ない)
  - セキュリティ的に安全
  - 実行時の速度が速い
  - 可読性が高い
  - 再利用性が高い
  - 資源(メモリ等)利用効率が高い
  - 長期間動作可能 (サーバ用途等)
  - など

  - チーム開発の場合、上記のような側面を議論しながら開発

実用プログラム (日常ツール)

- 日常作業に利用してみる
  - csvファイル (カンマ区切りテキストファイル) を読み込んで
  - 目的の処理を行って出力するプログラムなどは常に書いてみる
  - 例えば
    - 授業の出席が 0と1 で15回表現されていて中間試験100点、期末試験100点
    - 授業3割、中間3割、期末4割で点数付け
    - 90点以上はS, 80点以上はA, 80点未満は落第
    - それを処理するプログラムは？
    - ついでにグラフも作成するとしたら？
    - さらに平均は統計情報も出力するとしたら？

Excelでいいじゃないか、ではなく、JavaScriptとD3.jsだな、
と思えれば立派なプログラマ

実用プログラム (バックエンドシステム等)

- システムの俯瞰的な理解
  - 物理的に登場する機器は？
  - 各機器はどのように通信する?
    - L3,4以上 TCP HTTP セッション維持？
  - 入力と出力は？
  - プログラムはどのように起動される？
  - 無限ループの場合
    - 適切なsleep
    - メモリリークしていないか常に気にする
    - 消費するリソースは？ ログファイルの肥大は？

ハンズオン講習に向けた今後の学習

- ハンズオン講習に向けて
  - 予習用資料等
    - 1) 機材の準備⇒ Lチカをやってみよう ⇒ JS BinでLチカのコードを変更
      - (ハンズオン講習ではPCは使わずRaspberry Pi 3 のブラウザ経由でデバイス操作)
      - https://qiita.com/tadfmac/items/82817476615fdc7394b3
    - 2) 実際にコードを書きながらWeb GPIO API の基本的な利用方法を学ぶ
      - https://qiita.com/tadfmac/items/ebd01cfe46e30de70f3d
    - 3) Web I2C APIの学習
      - https://qiita.com/tadfmac/items/36d5467f79b6fd3114fb#_reference-791a5e7e5e7822bd7c0f
    - 4) 応用編
      - ○複数のセンサーの組み合わせなど
        - https://qiita.com/tadfmac/items/b17d8c6a35b31c495a36
      - ○Web GPIO API と Web I2C APIを組み合わせたWebアプリを作る
        - https://qiita.com/tadfmac/items/d627f8d2fec3c5f8711b#_reference-b24fd7f51432dcb011a4

JavaScriptのさらなる高度な学習

- JavaScript
  - ブラウザ上だけでなく様々な環境で動作する
  - 効率的な開発のため、とても応用の幅が広い言語

- JavaScript学習のロードマップ

	文法	変数	値	制御	関数
初級	字句、式と文、 コメント	宣言、参照、 代入	定数、配列、 オブジェクト	演算子、条件分岐、 繰り返し、try-catch	関数宣言、 関数呼び出し
中級	typeof、 instanceof	環境チェーン	プリミティブ値、 型判定、クラス	実行コンテキスト	関数式、 this束縛
上級	予約語	プロパティ記述子	型変換	非同期処理	クロージャ

  - まずは動くコードを書けて、且つ、
  - ある程度サンプルコードを読み解けるところまでできたら
  - 次ステップ(中級)を検討

JavaScriptを身に付けるために

- 今後の学習1
  - csvファイルのデータを前提に、データ処理プログラムを作ってみる

- 今後の学習2
  - D3.jsなど見た目でも楽しめるプログラムを
  - Node.jsなどをインストールしてローカル環境で様々なプログラムを
  - jQueryなどで動きのあるHTMLを
  - enchant.jsなどでスマホアプリも

- 今後の学習3
  - 「JavaScript Raspberry Pi」「JavaScript CHIRIMEN」等で検索
  - 様々なサンプルコードが出てきますので処理を読み解きましょう
  - システム全体を見通しましょう

この資料はクリエイティブ・コモンズ 表示 4.0 国際
ライセンスの下に提供されています。