情報科学実験II
JavaScript 言語について

2022/11/19

海谷 治彦

目次

- ウエブアプリの概要復習
- JavaScriptを動作させるシステム構成
- プログラム言語としての側面
- まとめ

- 演習2

- サンプルコードは授業ページからzipでまとめて落とせます.

https://www.sci.kanagawa-u.ac.jp/info/kaiya/e2/

復習

もっとも基本的な例

- ブラウザが見たいページをリクエストして，
- そのページのデータがレスポンスとして返ってくる．

復習

基盤技術の分類

- クライアントサイド・コンピューティング
  - ブラウザ側で処理を行う技術
  - 描画だけでなく，計算も行える．
  - プログラムがネットワーク上で転送される
    - Mobile Code とも呼ばれる．
  - 「フロントエンド」とも呼ぶ．
- サーバーサイド・コンピューティング
  - サーバー側で処理を行う技術．
  - 入出力データがネットワーク上で転送される
    - クライアントとはHTTPで通信する
  - サーバーは一台とは限らない．
    - 例: 商品の発注と，支払いの決済は別のサーバーが行う場合がある．

復習

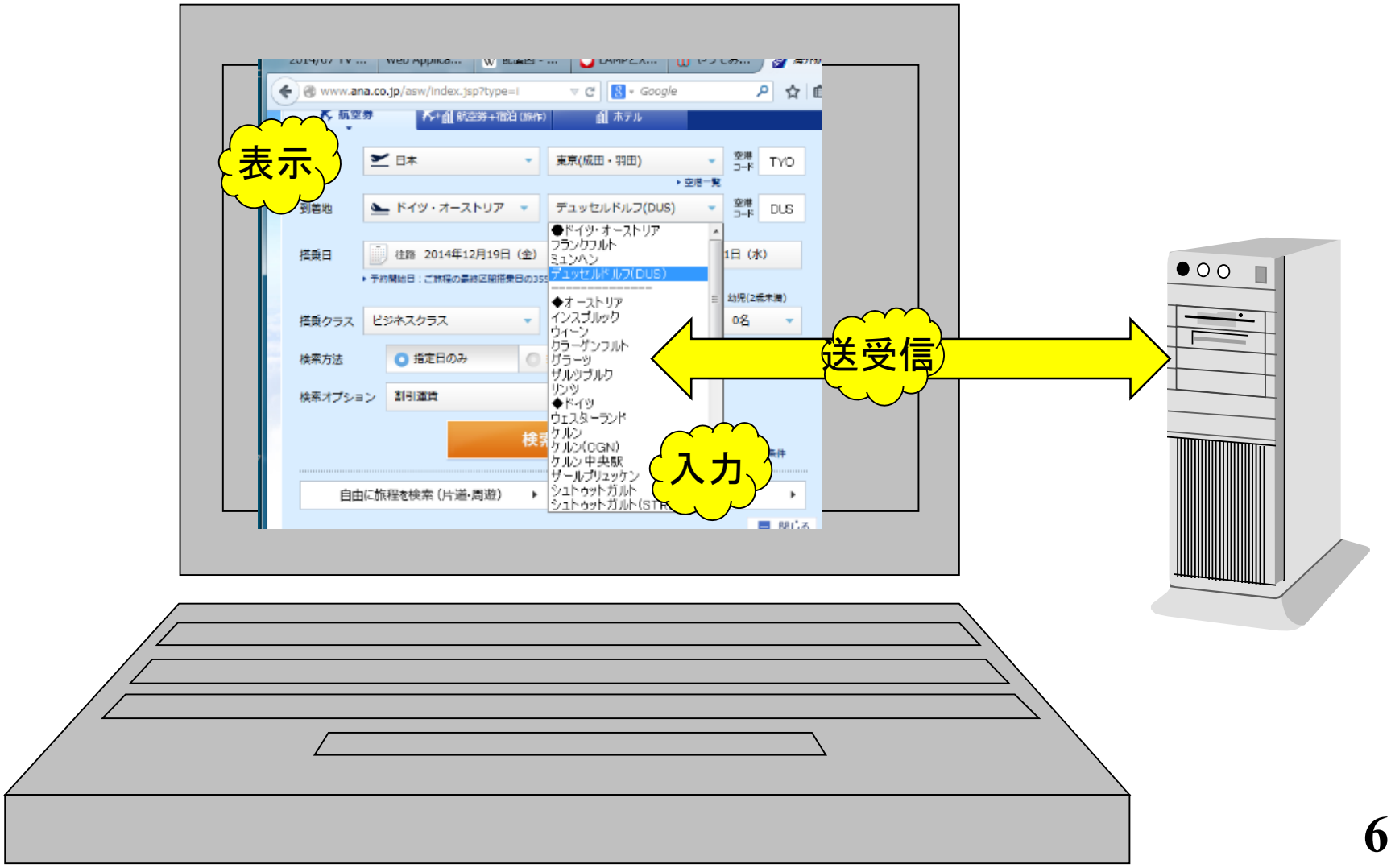
クライアントサイドの具体的技術

- HTML5, CSS3
  - ブラウザ上で文字や図形を描画するための言語
  - 音声や動画の再生も可能.
  - クライアント側でのデータ保存に関する仕様もある.
- JavaScript
  - ブラウザ上で計算を行うための言語.
  - 基本，C言語に似てる.
  - 文字列やデータ等の変換も計算として行える.
  - 事実上，唯一のクライアントサイドの処理言語
- 廃れたもの
  - Flash, Java/Applet, Java/FX等のRIA (Rich Internet Application)

復習

クライアントサイドの主な役割

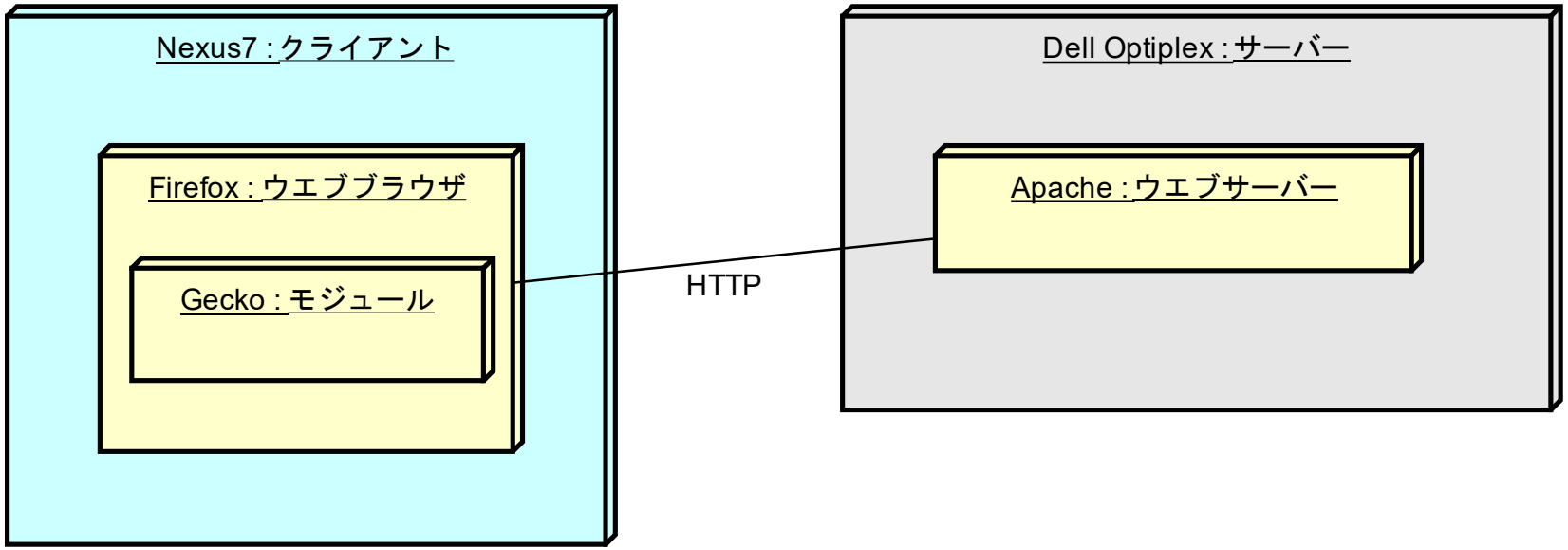
- 入力，表示，データの送受信



復習

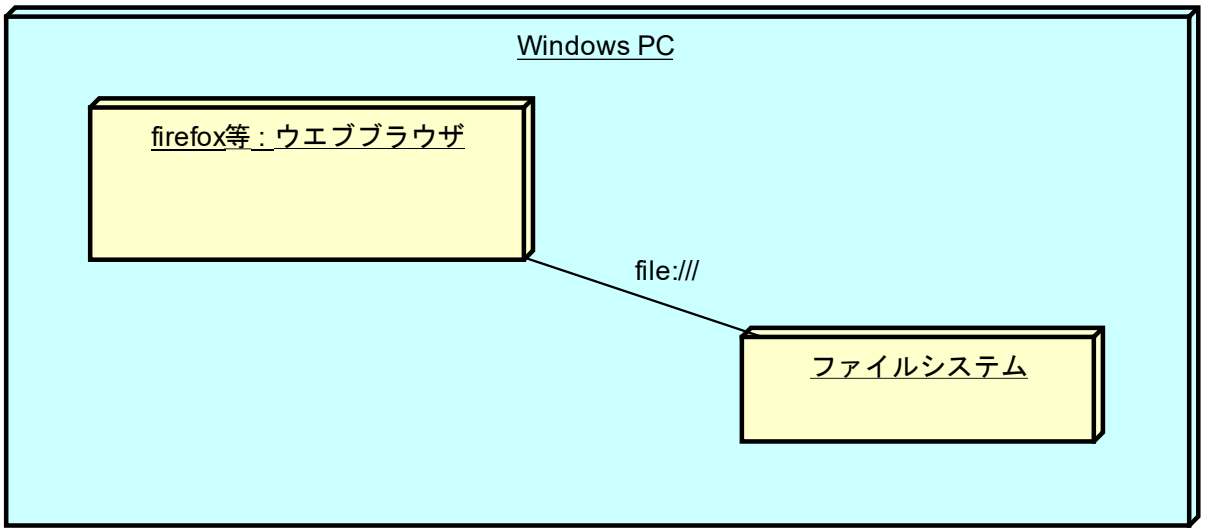
本来この構成でいきたいですが・・・

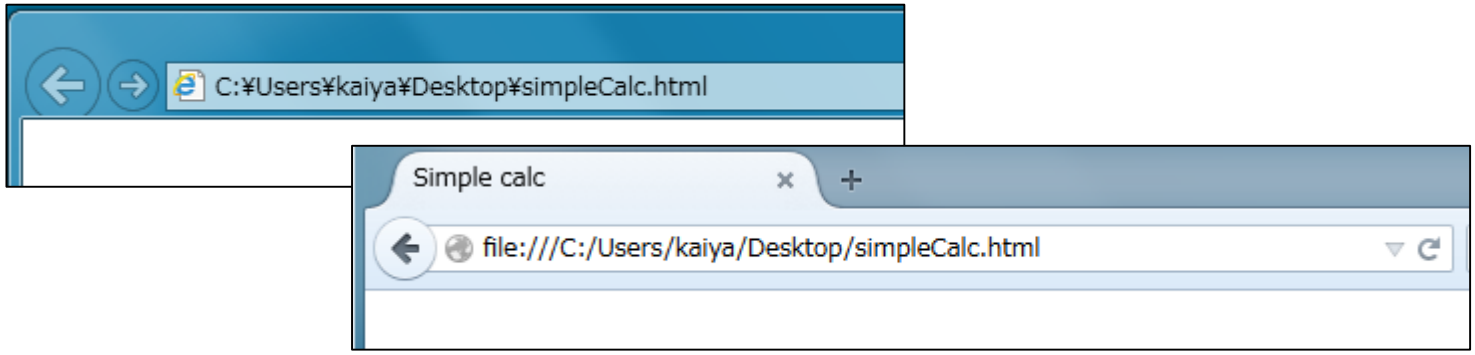
- 最低限の体裁をもつウエブアプリケーション



とりあえず以下の構成で試用してください

- 要はクライアント自身のファイルシステムを直接読む.





JavaScriptの特徴

- C言語同様，普通に計算等ができるプログラム言語.
- C言語やJavaと異なりデータ型に対して柔軟な対応をする．Perl やPythonにノリは近い.
- ウエブページ(html)内に埋め込んで利用するのが普通.
  - 結果として，素性の知れないページを見るだけで，自分のPCで怪しいプログラムが動く可能性がある.
- ウエブページ内の要素(たとえば，divで囲まれた区画等)を操作する機能がある．(次回)
- 対話的な操作を可能とする．(次回)
  - 結果として，ページ遷移無しにページに見えている内容を変更できる.
- ページレイアウトの標準化，作成の効率化ができる．(次回)

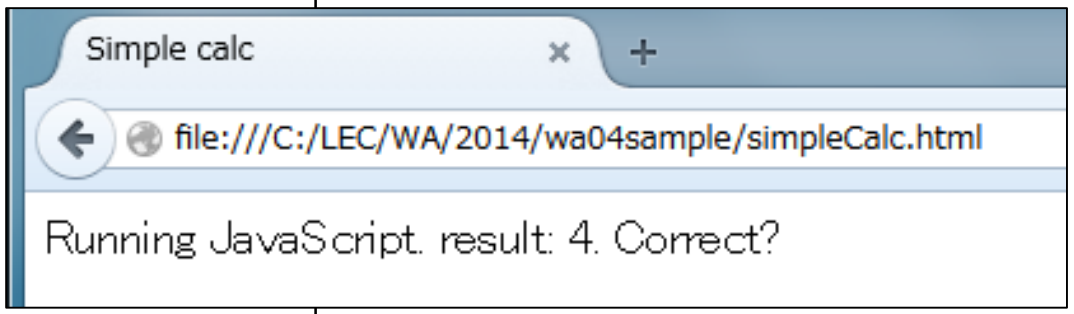
簡単なJavaScriptの例

- 太字の部分がJavaScript.
- コメントはCおよびC++風がOK.
- 変数の型は指定しない.
- 結果はHTMLに埋め込んでる.
- 以降の話題だが, ページ内の記述の更新も可能.

```
<!DOCTYPE html>
<!- simpleCalc.html -->
<html>
<head><title>Simple calc</title>
</head>
<body>Running JavaScript.
<script>
// 変数宣言, コメントはC++風.
var v1=5;
var v2=3;
var result;
/*
 結果の表示, コメントはC風.
 */
result = (v1+v2)/2;
document.write("result: "+result);
</script>
Correct?
</body> </html>
```

実行結果

```
<!DOCTYPE html>
<!- simpleCalc.html -->
<html>
<head><title>Simple calc</title>
</head>
<body>Running JavaScript.
<script>
// 変数宣言, コメントはC++風.
var v1=5;
var v2=3;
var result;
/*
 結果の表示, コメントはC風.
 */
result = (v1+v2)/2;
document.write("result: "+result);
</script>
Correct?
</body> </html>
```

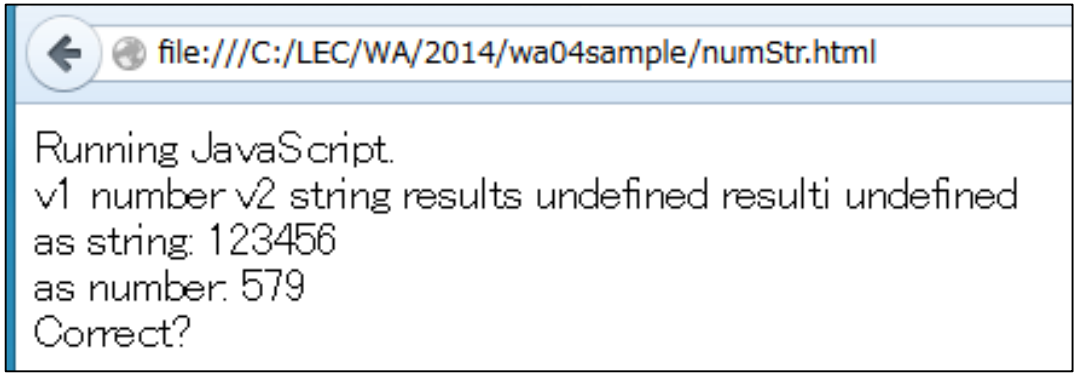


データ型と変数

- 多様なデータ型がある.
  - 数値，文字列
  - 配列 (後述)
  - 関数，オブジェクト (後述)
- 変数宣言時に型指定が無いので，異なるデータ型を同一の変数に記録することができる.
  - 数値と文字列は適宜読み替えをする，たとえば，
    - 123 と “456” を足すと，579になったり，”123456” になったり.
- 変数の型を調べる演算子があるので，一応，安心.
  - typeof

例題: 数値と文字列の読み替え等

```
<!DOCTYPE html>
<html><head><title>numStr.html</title></head>
<body>
Running JavaScript. <br>
<script>
// 変数宣言，コメントはC++風.
var v1=123;
var v2="456";
var results;
var resulti;

results=v1+v2; // 文字列に+演算子があるのでそっちにひっぱられる
resulti=v1+1*v2; // 文字列には*がないので数値に強制変換

document.write("as " + typeof results + ": " + results);
document.write("<br>");
document.write("as " + typeof resulti + ": " + resulti);
document.write("<br>");
</script>
Correct?</body></html>
```



undefined 未定義の変数値

- 変数は宣言した時点では値はおろか型もきまってない.
- こういった，初期化してない変数の値？を示すのに undefined を用いる.
- ある変数が undefined か否かを判定する述語(isUndefined的なもの)は無いようだ.
- ごく普通に,

```
var1 == undefined
```

と判定するか,

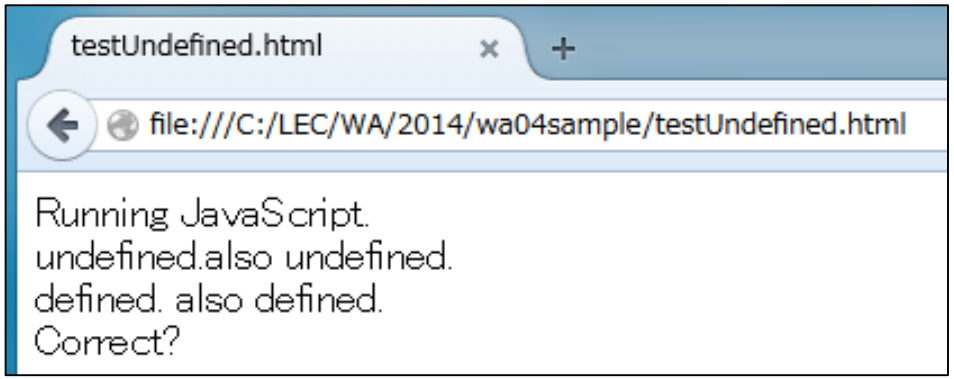
```
var1 == void 0
```

とするのが常套句らしい.

- 後者のほうが安全らしい. undefinedの値自体，上書き可能らしいので.

例題 testUndefined.html 抜粋

```
<script>
var result;

if(result == undefined) document.write("undefined."); else document.write("defined. ");
if(result == void 0) document.write("also undefined."); else document.write(" also defined. ");

result=10; // 何かをdefineする

if(result == undefined) {
        document.write("undefined.");
} else {
        document.write("defined. ");
}
if(result == void 0) {
        document.write("also undefined.");
} else {
        document.write(" also defined. ");
}
</script>
```



強制型変換

- 明示的に変数を特定の型に変換するビルトイン関数がある.
- parseInt
- Boolean
- parseFloat
- String

- typeConv.html 参照.

配列の特徴

- Javaと同じところ
  - 一次元配列しかない.
    - 多次元配列は配列の配列で構成.
  - 配列長を 配列名 .`length` で得られる.
  - 添え字は 0 からはじまる.
- Javaと違うところ
  - 異なる型の値が同居できる
  - 配列長を定義後に自由に伸縮できる.
  - 値未定義要素には 値 undefined が入っている.
  - 途中の要素を削除してつめることができる.

配列の作り方

- 宣言して要素の値を決める

```
var seasons = new Array();
seasons[0]="spring";
seasons[1]="summer",
seasons[2]="autumn";
seasons[3]="winter";
```

- 列挙する

```
var seasons = new Array("spring",
"summer", "autumn", "winter");
```

もしくは

```
var seasons = ["spring", "summer",
"autumn", "winter" ];
```

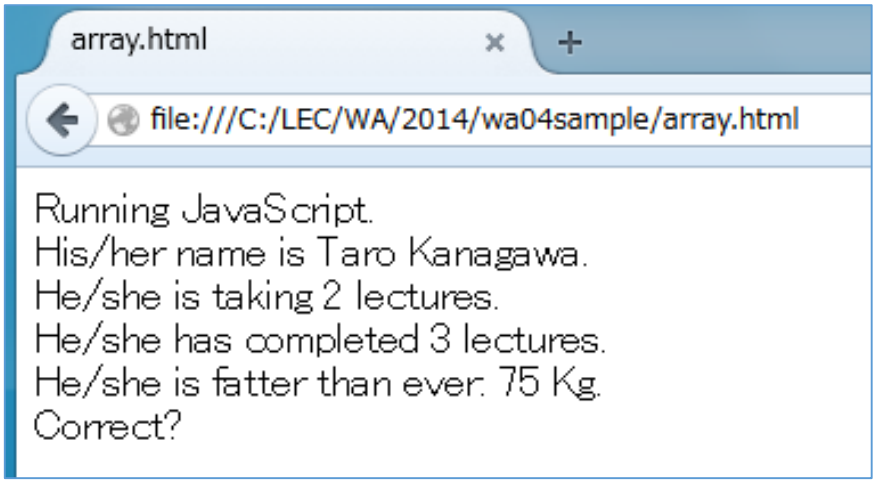
配列の使い方

- 長さを知る
  ```
  var i=seasons.length;
  ```
- 値を参照
  ```
  var fall=seasons[2];
  ```
- 値を更新
  ```
  seasons[2]="fall";
  ```
- 要素を追加できる.
  - `seasons.push("autumn");`

他，色々メソッドがある.

例 `arraySeasons.html`

例題 配列操作 array.html




```
<body>
Running JavaScript. <br>
<script>
// 本来は構造体やオブジェクトで
// 扱うべきですね．まぁ，デモってことで．
var take = ["math", "chem" ];
var finish = ["prog", "phys", "logic"];
var profile = [ "Taro", "Kanagawa",  65, 170, take, finish];

document.write("His/her name is "+profile[0]+" "+profile[1]+". <br>");
document.write("He/she is taking "+profile[4].length+" lectures. <br>");
document.write("He/she has completed "+profile[5].length+" lectures. <br>");

profile[2] += 10;

document.write("He/she is fatter than ever: "+profile[2]+" Kg. <br>");
</script>
Correct?
</body>
```

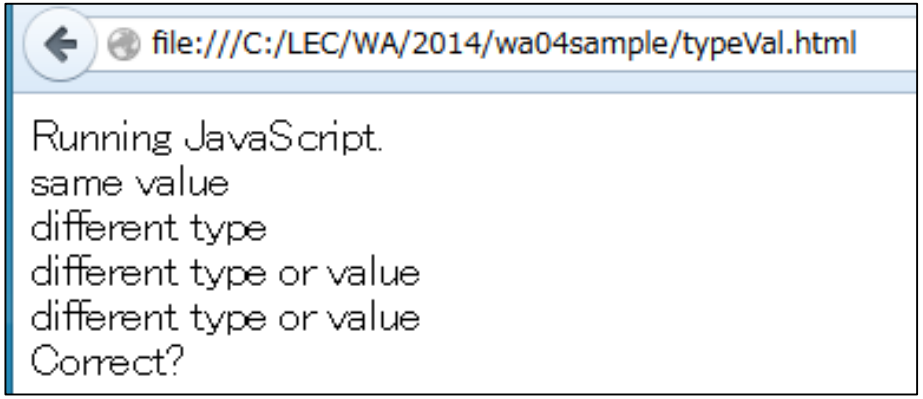
Cと同じ演算子，比較演算子

- 数値 + - * / %
- 代入 = += ++ 等
- 論理 && || !
- 三項演算子 ? :

- == != > < >= <=

Cと違う演算子

- (数値を示す)文字列と数値を適宜，読み替えする.
  - 言及済
- 型と値の双方が一致した場合のみ真の比較演算子
  ===　!==
  - typeVal.html を参照.




```
var numInt=23;
var strInt="23";

// 値の一致チェック
document.write(numInt == strInt? "same value": "different value");

// 型の一致チェック
document.write(typeof numInt == typeof strInt? "same type": "different type");

// 値と型の同時一致チェック
document.write(numInt === strInt? "same type and value ": "different type or value");
```

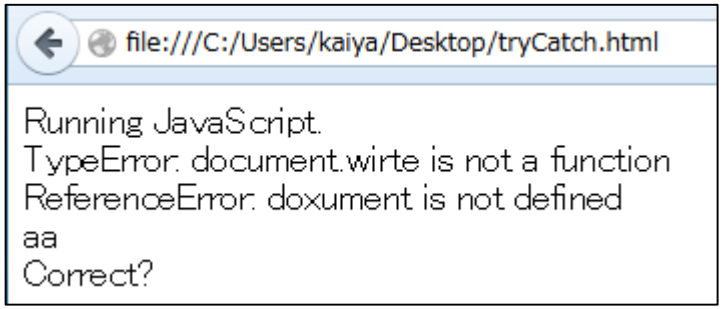
CやJavaと同じ制御文

- if 文, if else if 文
- switch文
- for文
- while, do while文
- break, continue 文

- try catch文
  - 実行時のエラーを補足するもの.
  - Javaの場合は，Throwableインタフェースを実装した，Error と Exception を catch できた.
  - JavaScriptの場合，Error というオブジェクトがありコレをcatch できる.

例題 tryCatch.html 抜粋

```
<script>
var a=10;

try{
        document.wirte("hello"); // write の綴りミス
}catch(err){
        document.write(err);
}
document.write("<br>");


try{
        doxument.write("hello"); // 存在しないオブジェクトを参照 ヌルポっぽい
}catch(err){
        document.write(err);
}
document.write("<br>");
</script>
```



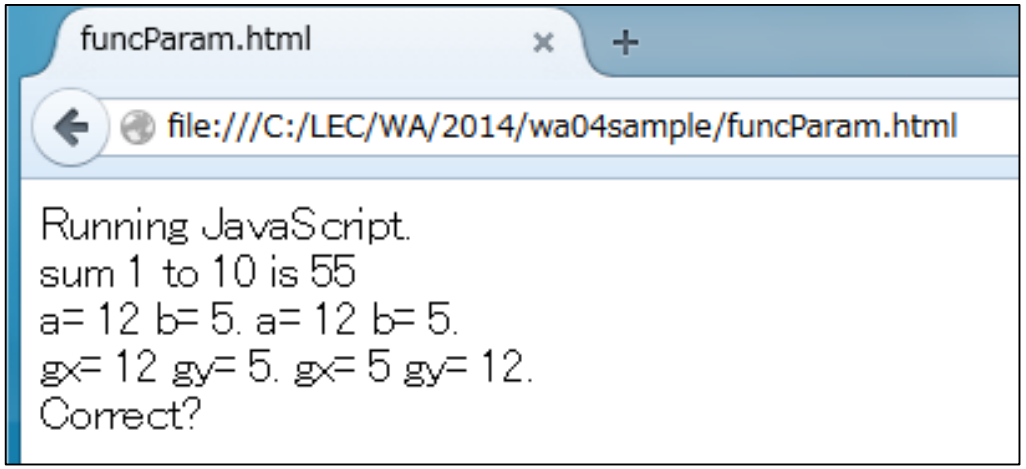
関数

- CやJavaのメソッド同様，値渡しである.
  - よって，基本，引数の改変はできない.
  - しかし，Java同様，後述のオブジェクト(への参照)を用いることで，オブジェクトの属性の変更は可能.
- 型を明示しないので，引数と返値の型指定も無い.
- 変数に var を付けないとグローバル変数になる.
  - 関数を呼び出した部分の同名の変数があれば，それと同一視される.
  - 関数内だけで使う変数は，必ず var をつける.
  - パラメタはvarがついているものとみなされている.

- 関数定義は HEADパートに書いてもよい.
  - funcHead.html を参照.
  - 別ファイルに書いても良い (後述)

例題 値渡し等 funcParam.html

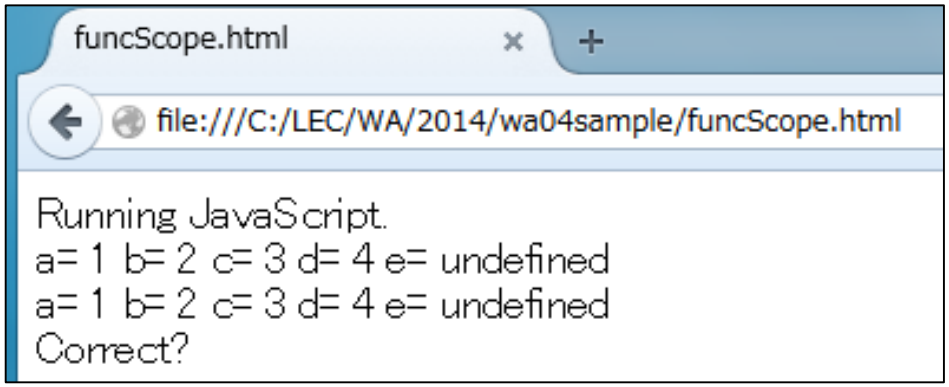
```
<script>
// 1からpまでの合計計算
function sum(p){
        var s=0, c;
        for(c=1; c<=p; c++){
                s += c;
        }
        return s;
}

// swapされないswap
function unswap(x, y){
        var tmp;
        tmp=x;
        x=y;
        y=tmp;
}
```




```
// sumのテスト
var result = sum(10);
document.write("sum 1 to 10 is "+result+"<br>");

// unswapのテスト
var a=12, b=5;
document.write("a= " + a + " b= " + b + ". ");
unswap(a,b);
document.write("a= " + a + " b= " + b + ". ");
document.write("<br>");
```

例題 変数のスコープ 更新不能

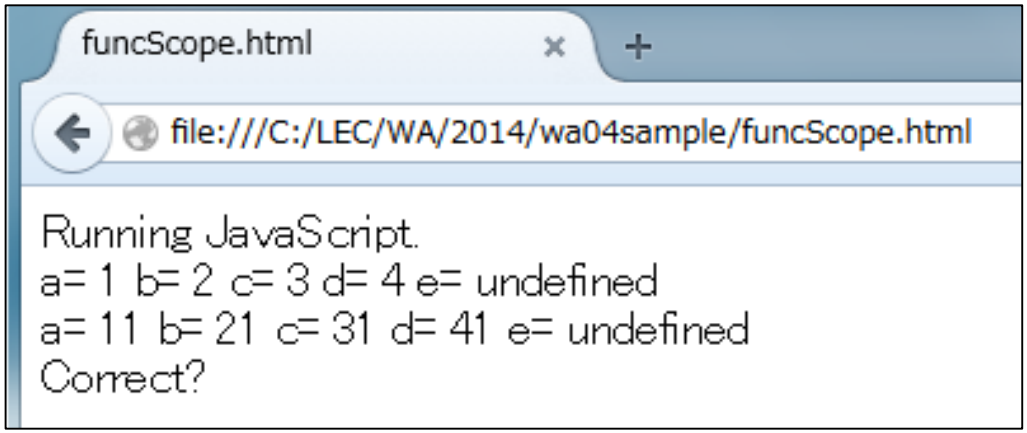
```
function printAll(){
        document.write(" a= "+a+ " b= "+b+ " c= "+c+ " d= "+d+ " e= "+e+"<br>");
}
// 実際は更新できない関数群
function unUpdateAB(){
var a, b;
        a=11; b=21;
        unUpdateCD();
}
function unUpdateCD(){
var c, d;
        c=31; d=41;
}

var a=1, b=2, c=3, d=4, e;

printAll();
unUpdateAB();
printAll();
```



funcScope.html 参照

例題 変数のスコープ 更新可能

```
function printAll(){
        document.write(" a= "+a+ " b= "+b+ " c= "+c+ " d= "+d+ " e= "+e+"<br>");
}

function updateAB(){
// var a, b;
        a=11; b=21;
        updateCD();
}
function updateCD(){
// var c, d;
        c=31; d=41;
}

var a=1, b=2, c=3, d=4, e;

printAll();
updateAB();
printAll();
```



funcScope.html 参照

変数宣言 var, let, const

- 再宣言: 同じ名前の変数を再度宣言できるか？
- 再代入: 二回目以降の代入ができるか？
- スコープ: ブロックか関数か？

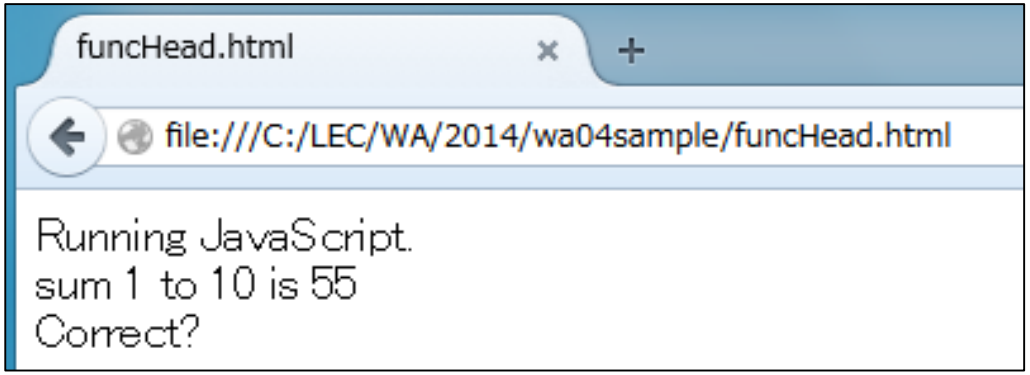
	再宣言	再代入	スコープ
var	○	○	関数
let	×	○	ブロック
const	×	×	ブロック？

- C言語等と同じ感覚で使うなら，let のほうがあってるのかもしれない.
- let のほうが宣言済の変数を別途宣言するミスを防げる.
- サンプル letconst.js

例題 関数定義はHEADでもいい

funcHead.html

```
<!DOCTYPE html>
<html>

<head><title>funcHead.html</title>
<script>
// 1からpまでの合計計算
function sum(p){
 var s=0, c;
 for(c=1; c<=p; c++){ s += c;}
 return s;
}
</script>
</head>

<body> Running JavaScript. <br>
<script>
var result = sum(10);
document.write("sum 1 to 10 is "+result+"<br>");
</script>
Correct?
</body>

</html>
```



オブジェクト

- Javaでいうインスタンスに相当.
  - 実体としては，連想配列
- 変数と関数(メソッド)をセットで有機的に使うという意味ではJavaと同じ.
  - Javaでのprivate的なものは無い模様.
- Javaとは異なりクラス定義が無いので，作り方が独特.
  - というか，変数と関数を配列っぽく並べるだけ.
- 属性値だけでなく，メソッド定義も実行時に更新できる.

既存のオブジェクト Built-in Objects

- Array 配列
- String
- Boolean
- Number
- Math 数学の定数や三角関数等を含む
- Date 日付関係の関数群
- Object
- Function
- RegExp 正規表現
- Error エラー try catch 関係
- window, document BOMとDOM (次回)
- JSON データ表現法(をサポートするオブジェクト)

オブジェクトを使う

- Java同様，”.”を用いて，変数や関数(メソッド)を指定する，たとえば，
  - `Math.PI` 円周率
  - `document.write(`文字列`)` 既出だが，HTML文書に文字列を挿入.
- 属性，関数への代入もできる，関数でさえ！
  - 関数の代入はoverrideと呼ばれる．結構，危険.
- privateは無いので情報隠蔽は困難.
  - 関数内のローカル変数を用いてprivateっぽいものを実現する模様.
- 関数引数にオブジェクトを渡し，メンバー変数を更新できる.

オブジェクトを作る

- Literal notation
  - 単に変数と関数を{}で並べるだけ
- コンストラクタ関数を定義
  - Javaっぽい書き方

- 他にもありますが，とりあえずこれくらい紹介します．

例題 オブジェクト関係

- 例題は別途htmlファイルをダウンロードしてみてください. (Zipで固めてあります)

- objLiteral.html
  - Literal notation で作成したオブジェクトをいじくる例題.
- objConst.html
  - コンストラクターを定義してオブジェクトを作る.

Literal vs Constructor

```
// Literal notation と呼ぶようです
// 円をあらわすオブジェクト 属性 半径 色 関数 円周 面積 半径のsetter/getter
var circle = {
  radius: 30,
  color: "red",
  circumference: function(){ return 2*this.radius*Math.PI; },
  area: function(){return this.radius*this.radius*Math.PI; },
  getRadius: function(){return this.radius;},
  setRadius: function(r){this.radius=r;}
};
circle.radius=20;
```

```
function circle(r, c){
  this.radius=r;
  this.color=c;
  this.circumference=function(){ return 2*this.radius*Math.PI; }
  this.area=function(){return this.radius*this.radius*Math.PI; }
  this.getRadius=function(){return this.radius;}
  this.setRadius=function(r){this.radius=r;}
}

// new演算子でオブジェクトをつくります.
var cir1=new circle(30,"red");
```

for in 制御文

- 配列の添え字を列挙することが可能な構文.
- ちょっと foreach に似てる.
- 実はオブジェクトは添え字が文字列の配列(連想配列と呼ばれる)なので，for in を適用可能
- サンプル
- forInArray.html
  - 配列を列挙する例
- forInObj.html
  - オブジェクト内の属性と関数名およびそれぞれの定義を列挙する例.
  - 属性も関数も後から更新や追加できる！

withステートメント

- with(オブジェクト){. . . . } で，{}中の変数に オブジェクト. がついているとみなすことができる.
- オブジェクトのメンバーアクセスが頻繁に起こる箇所では便利.
- objWith.html を参照． objConst.html と比較してみてね.

Scriptの書き方

- 直接，HTMLに埋め込む
  - HEADには関数定義等を
  - BODYには実行を開始するコードを

- 他のファイルに分ける
  - `<script src="URL名"></script>`
  - おき場所は上記に準拠，異なるサイトでもよい.
  - 例題 funcHead2.html を参照

ファイルをサーバーにおく

- HTMLおよびJavaScriptファイル(.js)は当然，ウエブサーバーにおくことができる.
- 結果，ページを見ただけで，他人のプログラムを自身で実行してしまうという，結構，危ないことになる.

- 素性も知らないフリーソフトやアプリをホイホイとインストールするよりは，かなりマシではあるが・・・

アンドロイドでも見れます

まぁ，簡単なJavaScriptなので，動くのは当たり前ですが．

本日の演習2

- 以下の問題をwebclassの WA演習2 に出してください.
- 木曜までに出してください.
- ファイル名 wa2.html 必要なら wa2.js 等，1つでもzipにまとめて出してください.

1. n人がm回勝負でじゃんけん勝負をする様を示すプログラムを作成せよ.
2. 各回で誰が勝ったかの名前を表示せよ．引き分けの場合は draw と表示せよ.
3. 最終的な勝者(無しおよび複数可)と，その勝った回数を最後に表示せよ．無論，勝った回数が一番多い人が勝者で，全員同数の場合，勝者無しで draw と表示せよ.
4. n, m はプログラムの冒頭で値を変数に代入せよ．n=3; m=10; 等．n≧2, m≧3とせよ.
5. 手の出し方はランダムに決まるものとせよ.
6. じゃんけんをする人に相当する Player クラス(オブジェクト)を必ず作成せよ.
7. 人は “player0”, “player1” 等の名前をもつものとせよ.
8. 各回のじゃんけん勝敗は，参加者の手が，ある二種類（たとえばグーとパー）のみになる時につくものとし，それ以外は引き分け(draw)とせよ.
9. グー，チョキ，パー等の手は内部的には 0, 1, 2 等の数値で扱ってよい.
10. コメント文はかくようにせよ.

画面例 (全く同じである必要は無い)

- 4人の10回勝負

```
G C P G draw
G C G P draw
G P C P draw
P C P C Winner(s): player1 player3
G G P P Winner(s): player2 player3
C C C C draw
G G C P draw
P G G C draw
P C P P Winner(s): player1
G C C P draw

Winner(s) [2 time(s)]: player1 player3
```

```
C C G P draw
P C G G draw
P C P G draw
P P P G Winner(s): player0 player1 player2
G G C P draw
P C G G draw
G C P C draw
G C C C Winner(s): player0
G G P P Winner(s): player2 player3
G P G P Winner(s): player1 player3

draw
```

参考 Node.js について

- 普通？のインタプリタとして JavaScript を動作させる環境.
- Python とか Perl 感覚で JavaScript が使える.
- JavaScriptでウェブサーバーも運営でき，サーバーもクライアントもJavaScriptでということもできるらしい.
  - LinkedIn (ビジネス系SNS)のサーバーで運用していたらしい．今は知らない.

- 興味がある人はググって使ってみてください.

本日は以上

Q/Aもしくは流れ解散