WEBプログラミング/演習
第4回

スケジュール

- 第1回：Webプログラミングの基本
- 第2回：JavaScriptの基本
- 第3回：オブジェクト
- 第4回：関数
- 第5回：オブジェクト指向
- 第6回：クライアントサイドJavaScript
- 第7回：公開APIの利用(1)
- 第8回：公開APIの利用(2)
- 第9回：Pythonプログラミングの基礎(1)
- 第10回：　Pythonプログラミングの基礎(2)
- 第11回：サーバサイドプログラミング(1)
- 第12回：サーバサイドプログラミング(2)
- 第13回：サーバサイドプログラミング(3)
- 第14回：学習内容の振り返り

この講義について（再記）

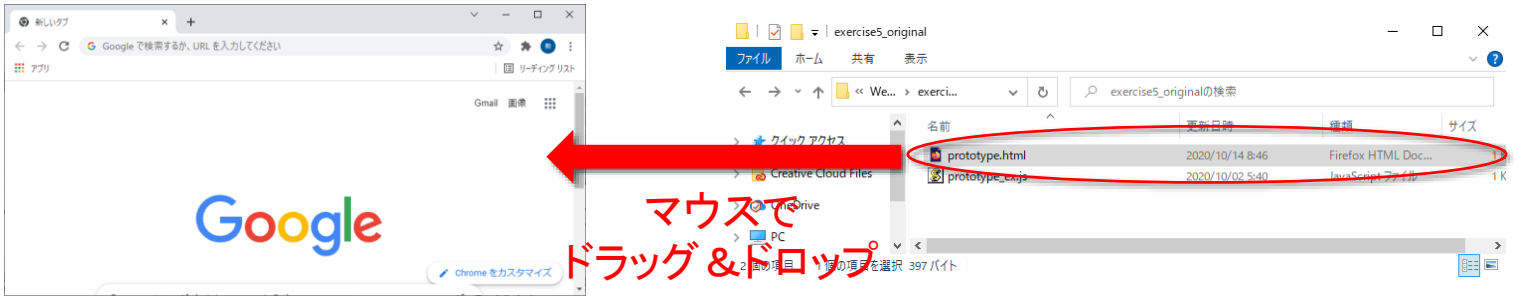
- 概要
  - オブジェクト指向スクリプト言語JavaScriptは、Webブラウザ、IoT機器のプログラミング言語としてまた、Pythonは、科学計算やWebサーバのプログラミング言語として広く使われています。
  - 本講義の前半ではWebブラウザで動作するプログラム（JavaScript）、後半ではサーバー上で動作するプログラム（python）を学び、Webアプリケーションが動作する仕組みを理解します。
- 成績評価
  - レポート課題（ほぼ毎回の授業で出題）と最終レポート課題で評価します。
  - 定期試験期間には試験は行いません．小課題と最終レポートの評価の割合は，4:6とします．
  - A+，A，B，C，D，Fの6段階評価を行い，D以上を合格とします．特別な理由なく全体の出席日数が3分の2に満たない場合は不合格となります．

関数

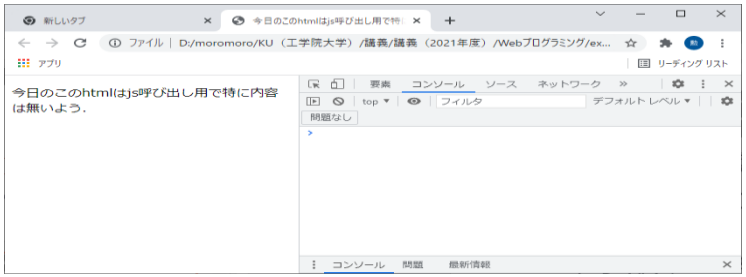
CoursePower（第4回>プログラム一式）からダウンロードください．これらはexercise4（フォルダ名）として扱いましょう．

今日の課題について

- 今日もGCPは使いません.
- ローカルの同じフォルダに今日DLしたファイル全てを置いて，htmlファイルをブラウザで開く.
  1. ブラウザ（Google Chromeなど）を起動して，CoursePowerからダウンロードした演習用HTMLファイルを，ブラウザ上にドラッグ＆ドロップする.



  2. あとはいつもどおりデベロッパーツールを開くだけ（F12を押す，Chrome上で右クリック->「検証」を選択，等）.



関数って何？

- 関数とは，与えられた入力（引数）に従って，なんからの処理を行い，結果（戻り値）を返す仕組み

- 一連の処理の「名前をつけて保存」と思えば直感的
  - 処理をつけた名前で呼び出すことで「何度も同じ処理を書く」ことを減らせる
  - バグがあったときに1箇所直せば良くなるので，メンテナンス性が上がる

- 関数定義の方法
  - function命令で定義する
  - Functionコンストラクタで定義する
  - 関数リテラル表現で定義する
  - アロー関数で定義する

function命令で定義

- もっとも基本的な定義方法

```
function 関数名(引数){
        任意の処理
        rerturn 戻り値;
}
```

- 関数定義の例（三角形の面積）

```
function getTriangle(base, height){
        let area = base * height / 2;
        return area;
}
```

  - base：底辺，height：高さ，area：面積
  - 与えられた引数を関数内で計算して，returnで結果を返している

関数の呼び出し

- 関数を実行したい箇所で,
  関数名(引数として与える値) の形で呼び出す

```
function getTriangle(base, height){
        let area = base * height / 2;
        return area;
}

let res = getTriangle(2, 6);
console.log(res); // 6が表示される
```

引数とreturn 命令

- 引数
  - その関数の中だけで有効な変数で，呼び出し元で与えられた値をその変数に入れて扱うというもの
  - カンマで区切って指定する
    - それぞれの引数は順番で対応づく
    - 先の例では，2がbaseに，6がheightにそれぞれ代入される

- return 戻り値
  - return命令の後に書かれるものが戻り値となって，呼び出し元に返る
    - 先の例では， getTriangle(2, 6);の実行結果 = returnで戻されるarea となって，それが呼び出し元のresに代入される
  - return 以降は実行されない
  - returnを省略した場合，関数は最後まで実行され，undefinedを返す

関数の命名規則

- まず，変数のルールを踏襲する
  - 次のスライド参照

- 動詞 + 名詞形式で命名する
  - 多くの場合，関数は「一連の処理」なので，どのような動作なのかを動詞で表現する
    - 値を得る関数なら，get
    - 値を設定する関数なら，set
    - などなど
  - camelCase記法で書く
    - 先頭は小文字，それ以降単語の先頭を大文字という記法
    - 必須ではないが慣例上，採用するほうが読みやすい
    - camel = ラクダ です．大文字部分がラクダのコブってこと．

[復習] 変数名の規則

- 変数名は「わかりやすい名前」をつけるのが原則
  - aとかbとかは，後から読んでわからないので避ける
- 規約上のルール
  - ルール1：1文字目は英字 or _ or $ のどれか
    - O: _name, $msg
    - X: 1st_name
  - ルール2：2文字目以降は，1文字目で使える文字 or 数字
    - O: msg1, _name0
    - X: c@c, name-0
  - ルール3：変数名の大文字小文字は区別される
    - Name != name
  - ルール4：予約語ではない
    - X: for, if, const, var, let, etc…

[課題1] 関数を定義しよう

- 2回目の講義スライド[課題6]をもとに以下の関数を定義せよ（2回目の[課題6]は次のスライド参照）

- 引数として rndm を受け取り，戻り値として引数として与えられたrndmの値に応じたおみくじの結果の文字列を返す関数 getOmikujiTest を定義せよ
  - rndmは0~1の値が渡されることを想定する

- 引数として何も受け取らず，戻り値としておみくじの結果の文字列を返す関数 getOmikuji を定義せよ
  - この関数の中で乱数を発生させること

[参考][課題2-6] if文使ってみよう

- 以下の文で0~1の乱数を変数 rndm に代入できる

```
let rndm = Math.random();
```

- 乱数 rndm の値が，0.3以下の時に「大吉」，
  0.3より大きく0.7以下の時に「吉」，
  0.7より大きく0.95以下の時に「凶」，
  それ以外の時に「大凶」が出力される
  おみくじを作成せよ

- 確認のために rndm の値も出力せよ

Functionコンストラクタで定義

let 変数名 =
　　new Function(引数, 引数, …, 関数の本体);

- このように，関数オブジェクトとしても定義可能
- 引数も関数の中身も文字列で与える

- この定義は「へー」くらいで留めておきましょう
  - 一般的にはほとんど使われず，特殊なケースでのみ使える記述法です
  - 文字列で与えることができる = プログラム中で動的に関数の中身を生成して与えることができる

関数リテラル表現で定義

- 以下の2つの性質による定義
  - JavaScriptでは関数は「データ型」の一種なので，変数に格納できる
  - 匿名の関数が定義できる

```
let 変数名 = function(引数,引数,…){
        任意の処理
        return 戻り値;
}
```

  - function(){} が匿名関数を生成する関数リテラル
  - それを変数に格納することで，その変数が関数として振る舞う

アロー関数で定義

- アロー関数を用いると関数リテラルをもっとシンプルに記述できる

```
let 変数名 = (引数, 引数,…) => {
        任意の処理
        return 戻り値;
}
```

- () => {} がアロー関数で，引数と中の処理を結びつける
- functionとかのキーワードを書かないでよい

[課題2] 関数リテラル，アロー関数

- [課題1]で作成した getOmikujiTestとgetOmikujiと全く同じ処理をする関数を関数リテラル，アロー関数を用いて定義せよ.

- 関数リテラルの場合には，変数名を以下にせよ
  - getOmikujiTestByLit，　getOmikujiByLit

- アロー関数の場合には，変数名を以下にせよ
  - getOmikujiTestByArr，　getOmikujiByArr

関数定義の注意

- エラーが無い != バグがない
  - JavaScriptに限らず，プログラミング全般に言えることだけど…
  - 特にJavaScriptでは意識しよう
- JavaScriptは「柔軟に書ける言語」
  ＝ 「意図しない挙動をする可能性が高い」
  - 適宜console.logで値を確認するなど，意図した値であるかを確認する癖をつけておこう
- 関数定義に関して，エラーにはならないが，意図しない挙動を起こしやすいものを紹介する

注意：return直後の改行

- JavaScriptでは；で文を認識するが，
  文脈からも自動的に補完する

```
let getTriangle = function(base, height){
        return
        base * height / 2;
}
```

  - これは，undefinedが返ってくる
  - returnの後に改行されると， return; であると解釈する

注意：関数はデータ型の一種


```
let getTriangle = function(base, height){
        return base * height / 2;
}

console.log(getTriangle(5,2));
getTriangle = 0;
console.log(getTriangle);
```

- 関数が格納されている変数に，別の値を代入して上書きできちゃう

ちなみに，function命令でも


```
function getTriangle(base, height){
        let area = base * height / 2;
        return area;
}

console.log(getTriangle(5,2));
getTriangle = 0;
console.log(getTriangle);
```

- 同じく， getTriangleは上書きできてしまう

function命令と他の定義の差

- function命令
  - 静的に関数を定義
    - 実行前にすべての関数は定義される
    - つまり，関数定義の文の前に関数呼び出しがあっても問題なく実行できる
- 関数リテラル，アロー関数
  - 動的に関数を定義
    - 定義の文の実行時に初めて定義される = 変数と同じ
    - 関数定義文が実行される前に関数呼び出しがあるとエラー

スコープ

- スコープは，変数がプログラム中のどこの場所から参照可能か？を決める概念で，大きく以下の2種類です.

- グローバルスコープ
  - プログラムのどこからでも参照可能
- ローカルスコープ
  - 定義された関数内のみで参照可能

スコープの例

- 以下のコードはどのように表示されるでしょうか？
  - viewValue関数の中でもscopeの値を表示しています

```
let scope = “ぴかちゅう”;
function viewValue(){
        let scope = “いーぶい”;
        console.log(scope);
}
viewValue();
console.log(scope);
```

- 結論としては，関数の中と外で個別に宣言された変数はたとえ同じ名前でも，別ものとして扱われます.

[課題3] スコープを確認しよう

作業A.

スライド24のコードを実際に実行して，
どのように表示されるか確認せよ.

作業B.

viewValue内の
let scope = “いーぶい”;
を
scope = “いーぶい”;
に書き換え，動作を確認せよ.

- 作業AとBの結果について，なぜこのようになるのかを説明せよ.

引数のスコープ

- 引数は，【基本的に】ローカルスコープであり，用いられる関数の中だけの話になります

- 例外として「参照型」の場合には，呼び出し元のスコープを引き継ぎます
  - 参照型は配列など（2回目の講義スライド課題5を参照）
  - 参照型は「実際の値の格納場所」の情報だけをやり取りするので，引数にも「場所」だけが渡される

  - 早い話，関数内で引数の配列に追加，削除，並べ替えをすると，呼び出し元の変数が直接変更される

ブロックレベルのスコープ

- ローカルスコープよりも更に小さい範囲として，ブロックレベルのスコープがある

```
if(true){
        let a = “ぴかちゅう”;
}
console.log(a);
```

  - 変数aはスコープ外なので，エラーとなる.

- letにはブロックレベルのスコープがあるが，varには無いので注意.
  - 基本的にはletで変数を宣言すること

引数に関する注意

- 一般的なプログラミング言語では引数の数が合わないとエラーが起こるが，JavaScriptでは起こらない

```
function showMessage(value){
        console.log(value);
}
showMessage(); //A
showMessage(‘ぴかちゅう’); //B
showMessage(‘ぴかちゅう’,‘いーぶい’); //C
```

- Bはもちろん正しいが，AもCも特にエラーにはならない
- JavaScriptは引数の数のチェックをしない

該当する引数がない場合

- Aのケースでは，valueには何が入っていることになるのか？
  - 該当する引数がない場合は undefined が入る

- コーディングミスで引数がないということを想定せずに，意図的に（利便性のために）引数を省略することを許す場合には，引数のデフォルト値を設定することがある

```
function getTriangle(base, height = 1){
        return base * height / 2;
}
```

  - この場合，heightに該当する引数がなかった場合，heightは1として扱われる

argumentsオブジェクト

- Cのケースの引数のvalueには，0番目の引数である’ぴかちゅう’が渡される
- それとは別に，showMessage関数は，argumentsというすべての引数を格納するオブジェクトを自動的に生成して持つ

```
function showMessage(value){
        console.log(arguments[1]);
        console.log(arguments.length);
}
showMessage(‘ぴかちゅう’,‘いーぶい’); //C
```

- ‘いーぶい’ と 2 が表示される

可変長引数の関数（その1）

- argumentsを使えば，引数が何個きても大丈夫な関数を定義できる
- 例えば，引数中の最大値を返すmax関数

```
function max(){
        let v = Number.MIN_VALUE;
        for(let a of arguments){
                if(v < a){
                        v = a;
                }
        }
        return v;
}
console.log(max(-10,3,6,9));
```

- でも，こういう場合，配列を引数に受け取って処理する方が直感的で読みやすい．こんなこともできるよってくらいです．

可変長引数の関数（その2）

- argumentsを使わなくも，引数が何個きても大丈夫な関数を定義できる

```
function max(...values){
        let v = Number.MIN_VALUE;
        for(let a of values){
                if(v < a){
                        v = a;
                }
        }
        return v;
}
console.log(max(-10,3,6,9));
```

- 引数の前に...をつけることで，可変長引数であることを明示できる
- argumentsの時と異なり引数を受け取るメソッドであることがわかる
- 直感的に理解できる変数名を設定できる
- 引数は配列として扱われる

[課題4] 可変長引数を使う関数

- 以下のAかBに取り組むこと
  - いずれの場合も可変長引数は...表記を用いて実装すること

- A.受け取った引数の個数に応じて，”ぴーーーーかちゅう“のように伸ばし棒を増やして表示する関数 pikapika を実装せよ.

- B. 可変長な引数を受け取る必要のある関数を考えて，実装せよ.
  - どのような関数にするかは各自自由に設定して良い
  - どういう関数なのかの簡単な説明も書いてください.（プログラムの近所にコメントアウトして書いてもらえればOK）

複数の戻り値を返す

- 複数の値を返す関数も作れます
  - 実態はシンプルで配列を返すだけです

```
function getMaxMin(...nums){
        return [Math.max(...nums),Math.min(...nums)];
}
```

  - numsは配列として扱われるので，Math.maxメソッドにわたす時には，...をつけて，値の列に変換してます

```
let res = getMaxMin(10,35,-5);
```

  - resには[35,-5]が入っています

```
let [max,min] = getMaxMin(10,35,-5);
```

  - maxに35, minに-5が入ります
  - このように配列を分解して代入することもできます

再帰呼び出し

- 関数の中で自分自身を呼び出すってやつです
- もちろんJavaScriptでもできます

- よくある例だけ

```
function factorial(n){
        if(n != 0) {return n * factorial(n-1);}
}

console.log(factorial(5));
```

  - 階乗を求めています
  - n! = n * (n-1)! というのを素直に実装していると見ればよいです.

高階関数

- 引数や戻り値に関数をとる関数
  - 関数は変数に格納できるデータ型なので，引数として渡すことも，戻り値として戻すこともできます

```
function showTypeA(text){ console.log(text + “だよ”); }

function showTypeB(text){ console.log(text + “ですよ”); }

function showText(text, f){ f(text); }

showText(“ぴかちゅう”, showTypeA);
showText(“ぴかちゅう”, showTypeB);
```

  - 引数にわたす関数によって，showTextの実行内容が決まる

匿名関数

- 高階関数に渡す関数は大抵がその場限りで処理させたい関数を書くのが一般的です

```
function showText(text, f){ f(text); }

showText(“ぴかちゅう”,
        function(text){
                console.log(text + “だよ”);
        });
```

  - 先程のshowTypeA関数を関数リテラルの形で直接書き込みました
    - このような，関数名がつけられず，その場限りの関数を匿名関数と呼びます

組み込みオブジェクトの高階関数

- 前回やったArrayオブジェクトにも高階関数があります（使用例は次頁参照）.
  - fncはたいてい匿名関数

- forEach(fnc)
  - 配列の内容を順に処理する
- map(fnc)
  - 配列を指定された内容で加工する
- some(fnc) /every(fnc)
  - 配列に条件に合致する要素が存在するか確認する
- filter(fnc)
  - 配列の内容を条件で絞り込む
- sort(fnc)
  - 配列の内容を独自ルールで並べ替える

forEach


```
let data = [2,3,4,5];
data.forEach( //dataの内容を順に処理する
        function(value,index,array){ //匿名関数
                console.log(value * value);
        }
); //4,9,16,25 が出力される
```

- forEachでは引数となる関数の引数に以下を渡す
  - 第1引数に要素の値
  - 第2引数にインデックス番号
  - 第3引数に元の配列
- 匿名関数の中で使わない引数もある
  - 今回はindex, arrayは使っていない

map


```
let data = [2,3,4,5];
let res = data.map( //dataの内容を加工
        function(value,index,array){ //匿名関数
                return value * value;
        }
); //return は [4,9,16,25] が代入される
console.log(res); //4,9,16,25 が出力される
```

- mapでは同じく引数となる関数の引数に以下を渡す
  - 第1引数に要素の値
  - 第2引数にインデックス番号
  - 第3引数に元の配列
- mapは加工した結果を返す必要があるので，returnが必要

some / every


```
let data = [2,3,4,5];
let result = data.some( //dataの内容を確認
        function(value,index,array){ //匿名関数
                return value % 3 === 0;
        }
); //result は true or false が代入される
```

- 3の倍数が含まれるか？というチェックをしている
- some/everyでもforEach/mapと引数は同じ
- someは，匿名関数の判定結果が1つでもtrueなら，someの結果はtrueになる
- 同様なものにeveryメソッドもあり，こちらはすべてがtrueのときにtrueを返す

```
let data = [2,3,4,5];
let result = data.filter( //条件で抜き出し
        function(value,index,array){ //匿名関数
                return value % 2 === 1;
        }
); //result は [3,5] が代入される
```

- 奇数を取り出している
- filterでも引数は同じ
- filterの結果は true となった要素の配列になる

sort

```
let ary = [5,25,10];
console.log(ary.sort()); //結果1:[10,25,5]
console.log(ary.sort(function(x,y){
                return x – y;
})); //結果2:[5,10,25]
```

- デフォルトは文字列としてソートされる（結果1）
- 引数は2つで，戻り値の正負で並び替えを決定する
  - 数値としての大小でソートがされることになる．つまり，匿名関数の引数x, yに配列aryの任意の2つの値を渡したとき，その戻り値が負ならxが先に，正ならyが先になるようにソートされる．（結果２）

sort その2


```
let classes = ['部長','課長','主任','担当'];
let members = [
        {name: '佐藤', clazz: '主任'},
        {name: '鈴木', clazz: '部長'},
        {name: '高橋', clazz: '担当'},
        {name: '田中', clazz: '課長'},
];
console.log(members.sort(function(x,y){
        return classes.indexOf(x.clazz)
                - classes.indexOf(y.clazz);
}));
```

- clazzの値に関してclassesの配列の順番になるように並べ替える
- 数値順や文字列順以外にも，比較基準を与えればソート可能

[課題5] いい感じに書き換えよう

- 2回目の講義スライド[課題7,8]（次頁にあり）をもとに以下を満たすように実装せよ

- 引数に n を受け取り，n回おみくじを引いた結果をArrayオブジェクトで返す関数 getOmikujiArray を実装せよ.
  - 内部で getOmikuji を呼び出すこと

- 引数に Array オブジェクトを受け取り，おみくじの合計得点を返す関数 calculateOmikujiPointを実装せよ.
  - mapメソッドで得点に変換した配列 points を得よ
  - forEachメソッドを使って合計得点 sum を求めよ
    - for of は使わなくてよい．置き換わるイメージ.

[参考][課題7] for文を使ってみよう

- 課題6で作成したおみくじを10回引いて，その結果を出力せよ
  - 吉，大吉，凶，大吉，吉，吉，… のように出力できればよい
- 工夫できそうであれば，各おみくじの結果が何回ずつ出たのかという集計結果も合わせて出力できるようにしよう.

[参考][課題8] for of 文を使ってみよう

- 大吉であれば3点，吉は1点，凶は0点，大凶は-1点というポイントが与えられるとして，課題7での出力結果を格納した配列に対して，for of 文を用いてポイントの合計を計算せよ.
  - 配列の定義は手作業で書き込んでも良い
  - イメージとしては以下のような形になる

```
let fortune = 結果の配列;
let point = 0; //合計ポイントを格納する変数
for(let f of fortune){
        得点判定とポイント加算を行う処理
}
```

成果物の提出（〆切：10/15 中）

- 作成および編集した以下のファイルを提出せよ
  - function_ex.js

- 提出はCoursePowerで行うこと