# JavaScript基礎実習

# 03. アロー関数

株式会社ジードライブ

# 今回学ぶこと

- 関数応用
  - 無名関数、アロー関数、コールバック関数
- 配列のメソッド
  - `forEach, map, filter, find, sort, reduce`

# 関数の定義

- 関数はfunctionキーワードを使用した構文で定義することができる

```
// 関数の定義
function calcBMI(height, weight) {
  const bmi = weight / (height * height);
  return bmi;
}

// 定義した関数の利用
const myBMI = calcBMI(1.71, 65.4);
console.log(myBMI);
```

```
22.365856160870017
```
コンソール

関数は関数式を使い、定義することもできる

# 関数式と無名関数

- 関数式は、関数(Functionオブジェクト)を生成する式
- 関数式の戻り値であるFunctionオブジェクトは変数に代入することができる
  - この時、関数名は付けなくてよい(**無名関数**)
  - 利用時は 変数名( ) で利用する

**無名関数**として記述することができる
(functionの後に、関数名を入れなくてよい)

```
// 関数の定義
const calcBMI = function(height, weight) {
  const bmi = weight / (height * height);
  return bmi;
};

// 関数の利用
const myBMI = calcBMI(1.71, 65.4);
console.log(myBMI);
```

```
22.365856160870017
```
コンソール

# アロー関数

- 無名関数の省略形として、アロー関数が存在する
  - functionというキーワードを取り、( )と{ }の間に => を挿入する
  - 引数の数や{ }内の処理によっては、( )や{ }も省略可能

  ※ 基本的に、Javaのラムダ式と同様のルールが適用される

```
// 関数の定義
const calcBMI = (height, weight) => {
  const bmi = weight / (height * height);
  return bmi;
};

// 関数の利用
const myBMI = calcBMI(1.71, 65.4);
console.log(myBMI);
```

コンソール
```
22.365856160870017
```

# アロー関数

- アロー関数の省略ルール

書式：引数が1つの場合は引数周りの ( ) を省略できる

```
const 変数名 = 引数名 => {
    // 何らかの処理
};
```

書式：関数内のステートメントが1つの場合はステートメント周りの { } を省略できる

```
const 変数名 = (引数名, 引数名, … ) => ステートメント;
```

書式：関数内のステートメントがreturn文だけの場合は、戻り値のみの記述に省略できる

```
const 変数名 = (引数名, 引数名, … ) => 戻り値;
```

書式：引数が無い場合は ( ) は省略できない

```
const 変数名 = () => ステートメント;
```

# 練習

- 練習03-1

# コールバック関数

- 引数として渡される関数を**コールバック関数**と呼ぶ
  - 引数や戻り値として関数を扱う関数を高階関数と呼ぶ
  - コールバック関数は、無名関数として定義することができる

```
// 高階関数の定義(第二引数がコールバック関数)
function repeat(num, callback) {
  for(let i = 1; i <= num; i++) {
      // このコールバック関数は引数を一つとるものとする
      callback(i);
  }
}

// 関数の利用(第二引数は無名関数として指定)
repeat(3, function(count) {
    console.log(count + ":Hello");
});
```

コンソール

```
1:Hello
2:Hello
3:Hello
```

# コールバック関数

- コールバックを無名関数にしない場合、関数名のみを引数として記す

```
// 高階関数の定義(第二引数がコールバック関数)
function repeat(num, callback) {
  for(let i = 1; i <= num; i++) {
     callback(i);
  }
}

// 上記関数のコールバックとして使用する関数の定義
function countHello(count) {
  console.log(count + ":Hello");
}

// コールバックを無名関数にしない場合、関数名のみを記す
repeat(3, countHello);
```

# ( ) の有無

- 関数は ( ) が付くことで実行される

```
// 関数の定義
function getMessage() {
  return "Hello";
}

// getMessage()が実行される
const x = getMessage();
console.log(x); // Hello

// getMessage()は実行されない
const y = getMessage;
console.log(y); // [Function: getMessage]

console.log(y()); // Hello
```

x には、getMessage()の実行結果"Hello"が代入される

yには、getMessage関数オブジェクトの参照が代入される

getMessage()が実行される

# 練習

- 練習03-2

# 配列のメソッド

- 配列(Arrayオブジェクト)は、以下のようなメソッドをもち、配列内の要素を順番に処理することができる
  - 一般的に引数はアロー関数の形式で記述される
  - これらのメソッドは組み合わせて利用できる
  - Javaにも同じようなメソッドが存在する(Stream API)

| メソッド | 説明 |
|---|---|
| forEach | 引数として与えられた関数を使い、配列の各要素を順番に処理する。戻り値なし |
| map | 元となる配列を加工し、新しい配列を返す |
| filter | 元となる配列から絞り込みを行い、新しい配列を返す |
| find | 配列から条件にマッチする最初の要素を返す |
| sort | 配列内の要素を並び替える ※配列そのものが変更される |
| reduce | 配列の要素を一つにまとめた結果を返す |

# forEachメソッド

- 配列の各要素を、コールバック関数を使い、順番に処理する

```
const array = [100, 200, 300];


array.forEach((num) => {
    console.log(num);
});
```

```
// for構文で記述する場合
for(const num of array) {
    console.log(num);
}
```

コンソール

```
100
200
300
```

アロー関数を省略して記述する場合

```
array.forEach(num => console.log(num));
```

# mapメソッド

- 配列の要素を加工し、新しい配列を作成する

```
const array = [100, 200, 300];


const newArray = array.map((num) => {
    return num * 2;
});

console.log(...newArray);
```

新しい配列に格納するデータをリターン

```
200 400 600
```
コンソール

アロー関数を省略して記述する場合

```
const newArray = array.map(num => num * 2);
```

# filterメソッド

- 配列から要素を絞り込み、新しい配列を作成する

```
const array = [100, 200, 300];


const newArray = array.filter((num) => {
    return num >= 150;
});

console.log(...newArray);
```

絞り込みの条件式をリターン

```
200 300
```
コンソール

アロー関数を省略して記述する場合

```
const newArray = array.filter(num => num >= 150);
```

# findメソッド

- 条件にマッチする最初の要素を返す

```
const array = [100, 200, 300];


const elm = array.find((num) => {
    return num >= 150;
});

console.log(elm);
```

検索の条件式をリターン

```
200
```
コンソール

アロー関数を省略して記述する場合

```
const elm = array.find(num => num >= 150);
```

# sortメソッド

- 配列の要素を並び替える
  - 戻り値を格納する変数は、元の配列変数と同じものを参照する

```
const array = [400, 200, 300, 100];

const sameArray = array.sort((num1, num2) => {
    return num1 - num2;
});

console.log(...array);
console.log(...sameArray);
```

昇順に並び替える場合: num1 - num2
降順に並び替える場合: num2 - num1

同じものを参照

コンソール

```
100 200 300 400
100 200 300 400
```

アロー関数を省略して記述する場合

```
const sameArray = array.sort((num1, num2) => num1 - num2);
```

# sortメソッド

- 元の配列を変更したくない場合は、スプレッド構文を利用する

```
const array = [400, 200, 300, 100];

const newArray = [...array].sort((num1, num2) => {
    return num1 - num2;
});

console.log(...array);
console.log(...newArray);
```

配列を複製

変更されていない

コンソール

```
400 200 300 100
100 200 300 400
```

# reduceメソッド

- 配列の要素を一つにまとめた結果を返す

```
const array = [100, 200, 300];

const elm = array.reduce((accum, num) => {
    return accum + num;
}, 0);

console.log(elm);
```

最終的な結果を蓄積していくための変数

accumに格納するデータをリターン

accumの初期値

```
600
```
コンソール

アロー関数を省略して記述する場合

```
const elm = array.reduce((accum, num) => accum + num, 0);
```

# 配列操作のメソッド使用例

例: 配列要素の絞り込み、並び替え、加工の組み合わせ

```
const items = [
    {name: 'テレビ', price: 78000},
    {name: '洗濯機', price: 32000},
    {name: '冷蔵庫', price: 99000},
    {name: '掃除機', price: 24000}
];

const newItems = items.filter(item => item.price >= 30000)
     .sort((item1, item2) => item1.price - item2.price)
     .map(item => {
        const newPrice = Math.floor(item.price * 1.1);
        return {...item, price: newPrice, type: '家電'}
     });

console.log(newItems);
```

コンソール

```
[
  { name: '洗濯機', price: 35200, type: '家電' },
  { name: 'テレビ', price: 85800, type: '家電' },
  { name: '冷蔵庫', price: 108900, type: '家電' }
]
```

# 練習

- 練習03-3