

続・関数

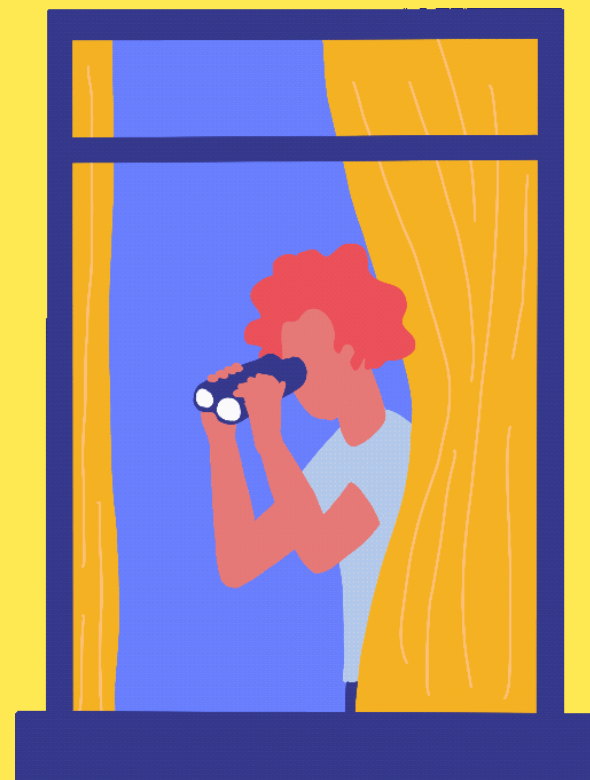
関数の大事な性質



スコープ

変数が「参照できるか」

どこで変数を定義したかで、どこからその変数を参照できるかが決まる。



関数のスコープ

```
function helpMe(){  
    let msg = "I'm on fire!";  
    msg; //"I'm on fire";  
}
```

```
msg; //NOT DEFINED!
```

msgはhelpMe関数内ではしか
参照できない

関数のスコープ

```
let bird = 'アオサギ';
```

```
function birdWatch() {  
  let bird = 'ムクドリ';  
  bird; // ムクドリ  
}
```

```
bird; // アオサギ
```

このbirdはbirdWatch関数
内でしか参照できない

ブロックスコープ

```
let radius = 8;

if(radius > 0){

    const PI = 3.14;


    let circ = 2 * PI * radius;

}

console.log(radius); //8
console.log(PI); //NOT DEFINED
console.log(circ); //NOT DEFINED
```

PIとcircはブロック内
でしか参照できない

レキシカルスコープ



```
function outer() {  
  let hero = 'ブラックパンサー';  
  
  function inner() {  
    let cryForHelp = `${hero}、助けて!`  
    console.log(cryForHelp);  
  }  
  
  inner();  
}
```

関数式



```
const square = function (num) {  
  return num * num;  
}  
square(7); //49
```


関数は
なんと...
オブジェクト
なのです!



高階関数



高階関数

関数を受け取ったり関数を返す関数

高階関数は：

- 引数として関数を受け取る
- 戻り値に関数を指定する

関数の引数関数

[illegible]

関数の戻り値が関数



```
function makeBetweenFunc(min, max) {  
  return function (val) {  
    return val >= min && val <= max;  
  }  
}  
  
const inAgeRange = makeBetweenFunc(18, 100);  
  
inAgeRange(17); //false  
inAgeRange(68); //true
```

メソッド

```
const math = {  
  multiply : function(x, y) {  
    return x * y;  
  },  
  divide   : function(x, y) {  
    return x / y;  
  },  
  square   : function(x) {  
    return x * x;  
  }  
};
```

オブジェクトのプロパティに関数を定義できる。

これを**メソッド**と呼びます

省略形

```
const math = {  
  blah: 'Hi!',  
  add(x, y) {  
    return x + y;  
  },  
  multiply(x, y) {  
    return x * y;  
  }  
}  
math.add(50, 60) //110
```

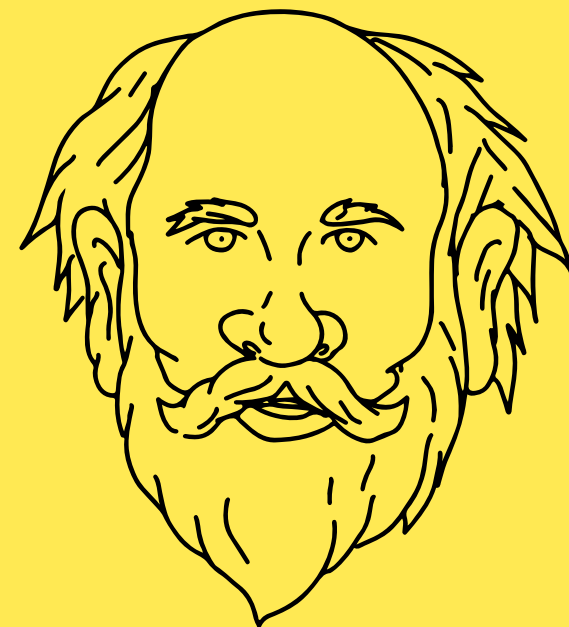
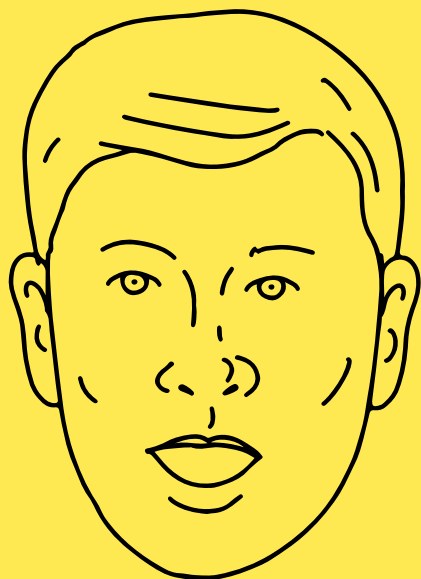
メソッドの定義は頻繁に行うので、省略して書くこともできる

メソッドの中のthis

同じオブジェクト内の、
他のプロパティを使いたいときにthisを活用しよう

```
const person = {  
  first: '太郎',  
  last: '山田',  
  fullName() {  
    return `${this.last} ${this.first}`;  
  }  
}  
  
person.fullName(); // "山田 太郎"  
person.last = '佐藤';  
person.fullName(); // "佐藤 太郎"
```


thisの値は、thisを使っている関数が
「どのように呼ばれたか」に依存する



同じ関数なのに

```
const person = {  
  first: '太郎',  
  last: '山田',  
  fullName() {  
    return `${this.last} ${this.first}`;  
  }  
}
```

```
person.fullName();  
// "山田 太郎"
```

違う結果？

```
const func = person.fullName;  
func()  
// "undefined undefined"
```

thisの値は、thisを使っている関数が
「どのように呼ばれたか」に依存する

