

デフォルト 引数

昔のやり方

```
function multiply(a, b) {  
    b = typeof b !== 'undefined' ? b : 1;  
    return a * b;  
}
```

```
multiply(7); //7  
multiply(7, 3); //21
```

デフォルト 引数

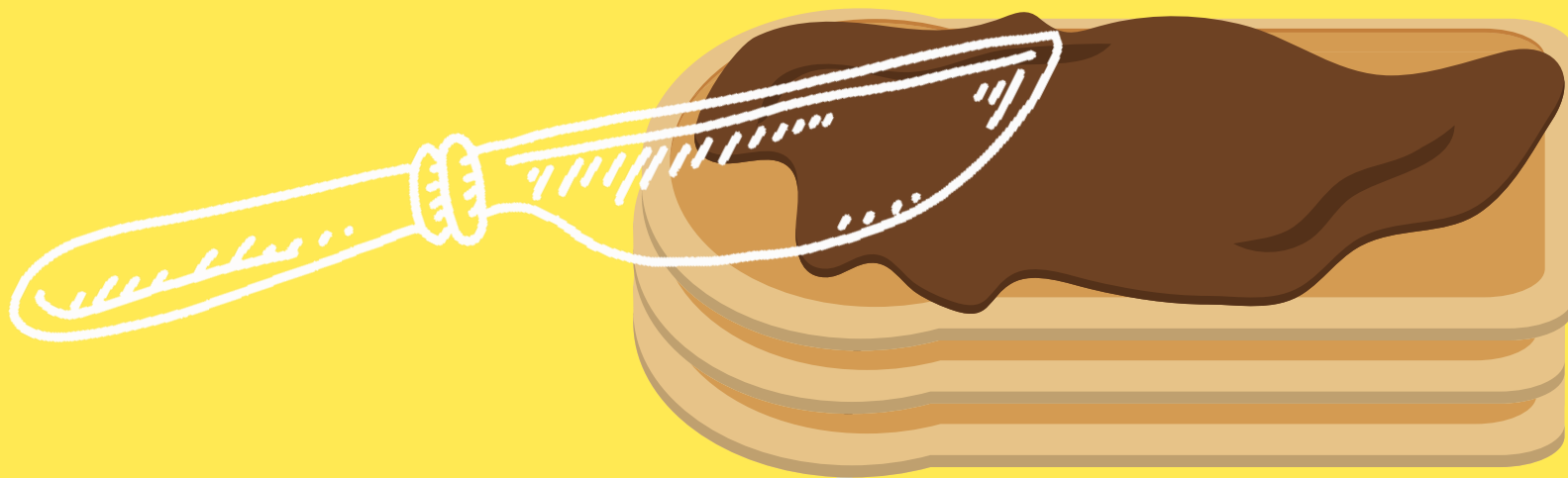
今どきのやり方



```
function multiply(a, b = 1) {  
    return a * b;  
}
```

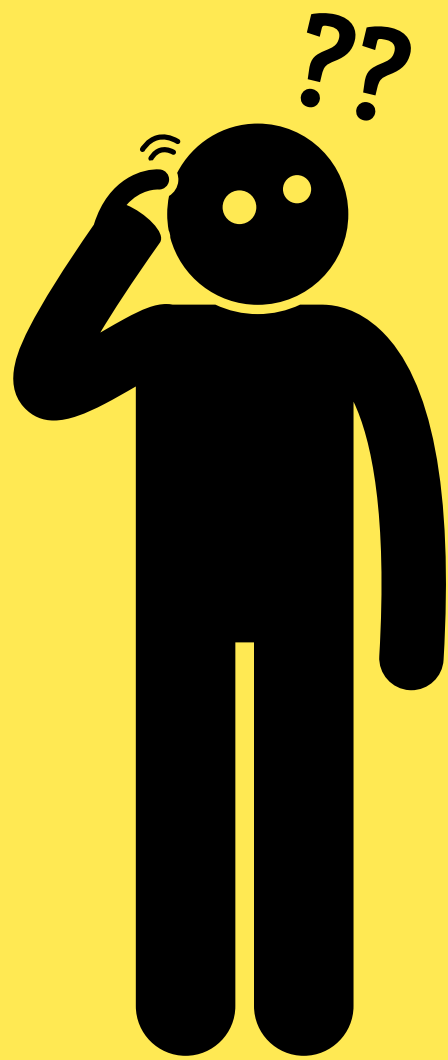
```
multiply(4); //4  
multiply(4, 5); //20
```

スプレッド構文



スプレッド構文

配列式や文字列などの反復可能オブジェクトを、0 個以上の引数 (関数呼び出しの場合) や要素 (配列リテラルの場合) を期待された場所で「展開」したり、オブジェクト式を、0 個以上のキーと値のペア (オブジェクトリテラルの場合) を期待された場所で「展開」したりすることができます。



スプレッド構文

関数呼び出しの場合



```
const nums = [9, 3, 2, 8];  
Math.max(nums); // NaN  
// スプレッド構文を使うと  
Math.max(...nums); // 67  
// ↓と一緒  
// Math.max(9, 3, 2, 8);
```

列挙可能なオブジェクト
(配列、Stringなど) を引
数に展開する

スプレッド構文

配列リテラルの場合

既存の配列から新しい配列を作成する。配列の要素を新しい配列にそれぞれ展開する。

```
const nums1 = [ 1, 2, 3 ];  
const nums2 = [ 4, 5, 6 ];  
  
[ ...nums1, ...nums2 ];  
//[1, 2, 3, 4, 5, 6]  
  
[ 'a', 'b', ...nums2 ];  
//["a", "b", 4, 5, 6]  
  
[ ...nums1, ...nums2, 7, 8, 9 ];  
//[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

スプレッド構文

オブジェクトリテラルの場合

```
const feline = {legs: 4, family: 'ネコ科'};
const canine = {family: 'イヌ科', bark: true};

const dog = {...canine, isPet: true};
// {family: 'イヌ科', bark: true, isPet: true}

const lion = {...feline, roar: 'ガオー'};
// {legs: 4, family: 'ネコ科', roar: true}

const catDog = {...feline, ...canine};
// {legs: 4, family: 'イヌ科', bark: true}
```

オブジェクトからオブジェクトにプロパティをコピーする。

レスト構文 (残余引数)

スプレッド構文に
似て非なるもの



arguments オブジェクト

```
function sumAll() {  
  let total = 0;  
  for (let i = 0; i < arguments.length; i++)  
  {  
    total += arguments[i];  
  }  
  return total;  
}  
sumAll(8, 4, 3, 2); // 17  
sumAll(2, 3); // 5
```

- すべて(*)の関数で使える
- 配列みたいなオブジェクト
 - length プロパティがある
 - push や pop のようなメソッドは使えない
- 関数に渡された引数をすべて含んでいる

*アロー関数は例外

レスト構文 (残余引数)

その位置にある残りの引数を
配列の中に入れることができる

```
function sumAll(...nums) {  
  let total = 0;  
  for (let n of nums) total += n;  
  return total;  
}
```

```
sumAll(1, 2); //3  
sumAll(1, 2, 3, 4, 5); //15
```


分割代入

- 配列の要素
- オブジェクトのプロパティを、別個の変数に割り当てる
すっきりと書ける構文



配列

分割代入



```
const raceResults = ['エリウド・キプチョゲ', 'フェイサ・リレサ', 'ゲーレン・ラップ'];

const [gold, silver, bronze] = raceResults;
gold; // "エリウド・キプチョゲ"
silver; // "フェイサ・リレサ"
bronze; // "ゲーレン・ラップ"

const [fastest, ...rest] = raceResults;
fastest; // "エリウド・キプチョゲ"
rest; // ["フェイサ・リレサ", "ゲーレン・ラップ"]
```

オブジェクト

分割代入

```
const runner = {  
  first: 'エリウド',  
  last: 'キプチョゲ',  
  country: 'ケニア',  
  title: 'Elder of the Order of the Golden Heart'  
}  
const {first, last, country} = runner;  
  
first; //"エリウド"  
last; // "キプチョゲ"  
country; // "ケニア"
```

関数パラメーター

分割代入

```
const fullName = ({first, last}) => {  
  return `${first} ${last}`  
}  
  
const runner = {  
  first: 'エリウド',  
  last: 'キプチョゲ',  
  country: 'ケニア'  
}  
  
fullName(runner); // "エリウド キプチョゲ"
```