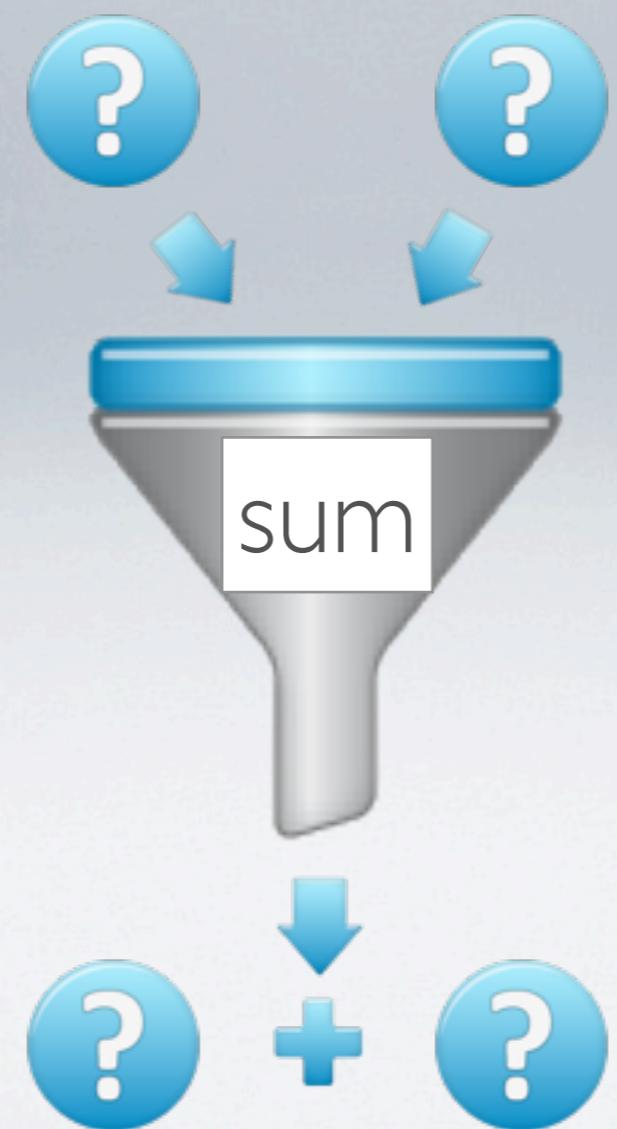


# Function

```
function sum (a, b) {  
    return a + b;  
}  
  
var result = sum(10, 20);  
console.log(result); // -> 30;
```

# Function

```
function sum (a, b) {  
    return a + b;  
}
```



# Function

```
function sum (a, b) {  
    return a + b;  
}
```

仮引数

返り値



sum(10, 20);

実引数

# Function

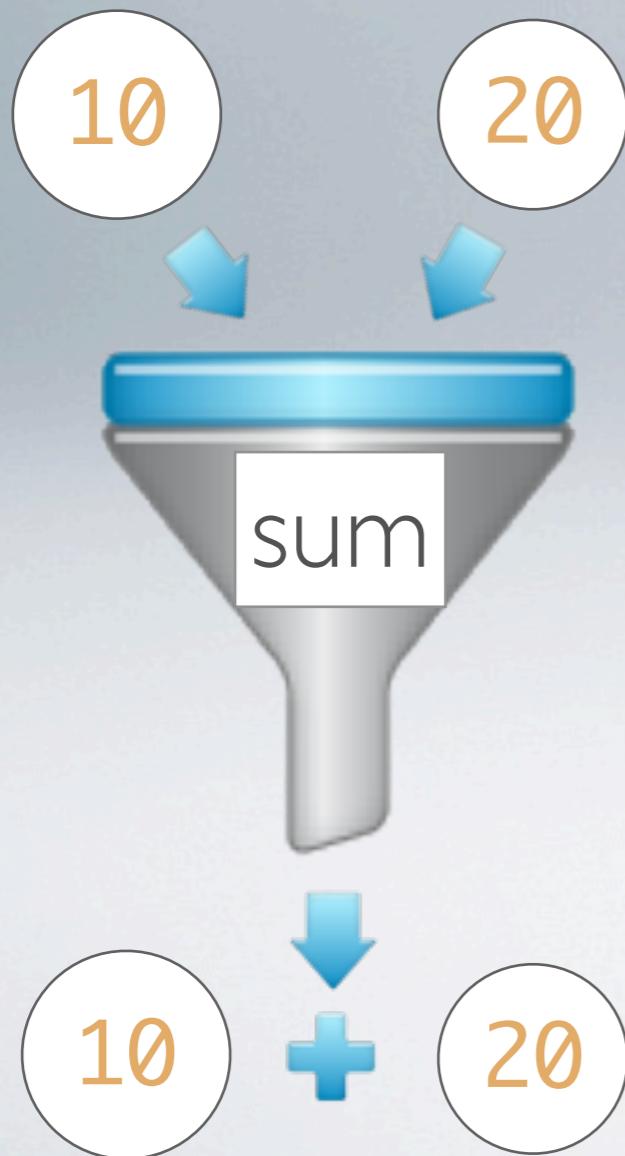
```
function sum (10, 20) {  
    return a + b;  
}
```



```
sum(10, 20);
```

# Function

```
function sum (10, 20) {  
    return 10 + 20;  
}
```



```
sum(10, 20); // -> 30
```

# Function

```
var a = 10,  
    b = 20;
```

```
function sum (a, b) {  
    var sum_result = a + b;  
    return sum_result;  
}
```

```
console.log(sum(50, 30));
```

```
var result = sum(30, 40);  
console.log(result);
```

# Function

a	10
b	20
sum	function (a, b)
result	sum(30, 40)

sum  
関数

a	
b	
sum_result	a + b

# Function

a	10
b	20
sum	function (a, b)
result	sum( 30, 40 )

別物!

sum関数

resultの時

a	30
b	40
sum_result	a + b

# Function

```
function average (arr) {  
    var sum = 0,  
        i = 0,  
        len = arr.length;  
    for (; i < len; i = i + 1) {  
        sum = sum + arr[i];  
    }  
    return sum / len;  
}  
console.log(average([10, 20, 30, 40]));  
console.log(average([150, 33, 21]));
```

# Function

```
function average (arr) {  
  var sum = 0,  
      i = 0,  
      len = arr.length;  
  for (; i < len; i = i + 1) {  
    sum = sum + arr[i];  
  }  
  return sum / len;  
}  
console.log(average([10, 20, 30, 40]));  
console.log(average([150, 33, 21]));
```

# Function

```
function average (arr) {  
  var sum = 0,  
      i = 0,  
      len = arr.length;  
  for (; i < len; i = i + 1) {  
    sum = sum + arr[i];  
  }  
  return sum / len;  
}  
console.log(average([10, 20, 30, 40]));  
console.log(average([150, 33, 21]));
```

全体

average	function (arr)

# Function

```
function average (arr) {  
    var sum = 0,  
        i = 0,  
        len = arr.length;  
    for (; i < len; i = i + 1) {  
        sum = sum + arr[i];  
    }  
    return sum / len;  
}  
console.log(average([10, 20, 30, 40]));  
console.log(average([150, 33, 21]));
```

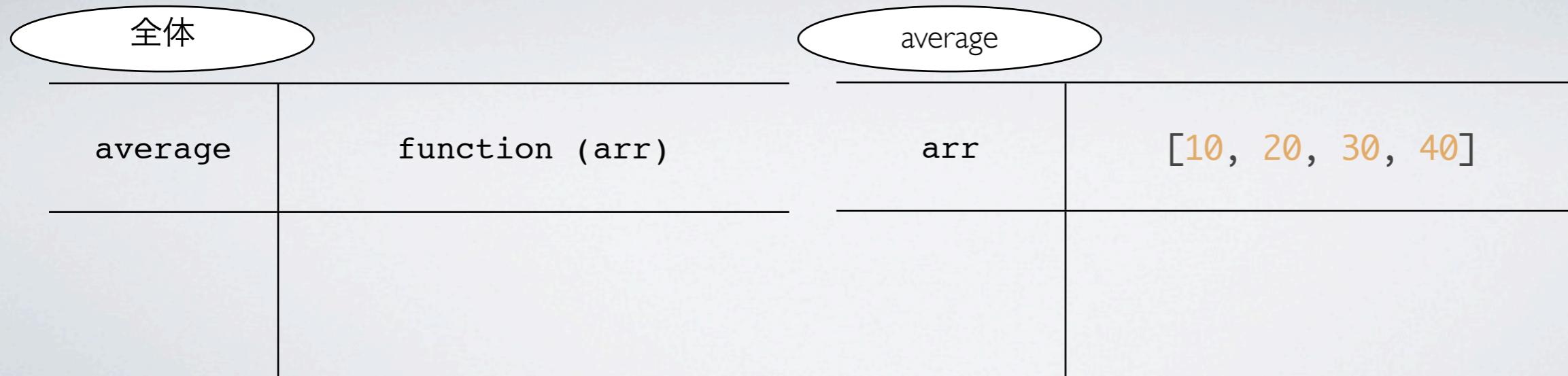
全体

average

function (arr)

# Function

```
function average (arr) {  
    var sum = 0,  
        i = 0,  
        len = arr.length;  
    for (; i < len; i = i + 1) {  
        sum = sum + arr[i];  
    }  
    return sum / len;  
}  
console.log(average([10, 20, 30, 40]));  
console.log(average([150, 33, 21]));
```



# Function

```
function average (arr) {  
  var sum = 0,  
      i = 0,  
      len = arr.length;  
  for (; i < len; i = i + 1) {  
    sum = sum + arr[i];  
  }  
  return sum / len;  
}  
console.log(average([10, 20, 30, 40]));  
console.log(average([150, 33, 21]));
```

全体

average	function (arr)

average

arr	[10, 20, 30, 40]
sum	0
i	0
len	4

# Function

```
function average (arr) {  
  var sum = 0,  
      i = 0,  
      len = arr.length;  
  for (; i < len; i = i + 1) {  
    sum = sum + arr[i];  
  }  
  return sum / len;  
}  
console.log(average([10, 20, 30, 40]));  
console.log(average([150, 33, 21]));
```

全体

average	function (arr)

average

arr	[10, 20, 30, 40]
sum	100
i	4
len	4

# Function

```
function average (arr) {  
  var sum = 0,  
      i = 0,  
      len = arr.length;  
  for (; i < len; i = i + 1) {  
    sum = sum + arr[i];  
  }  
  return sum / len;  
}  
console.log(average([10, 20, 30, 40])); // -> 25  
console.log(average([150, 33, 21]));
```

全体

average	function (arr)

# Function

```
function average (arr) {  
    var sum = 0,  
        i = 0,  
        len = arr.length;  
    for (; i < len; i = i + 1) {  
        sum = sum + arr[i];  
    }  
    return sum / len;  
}  
console.log(average([10, 20, 30, 40])); // -> 25  
console.log(average([150, 33, 21]));
```

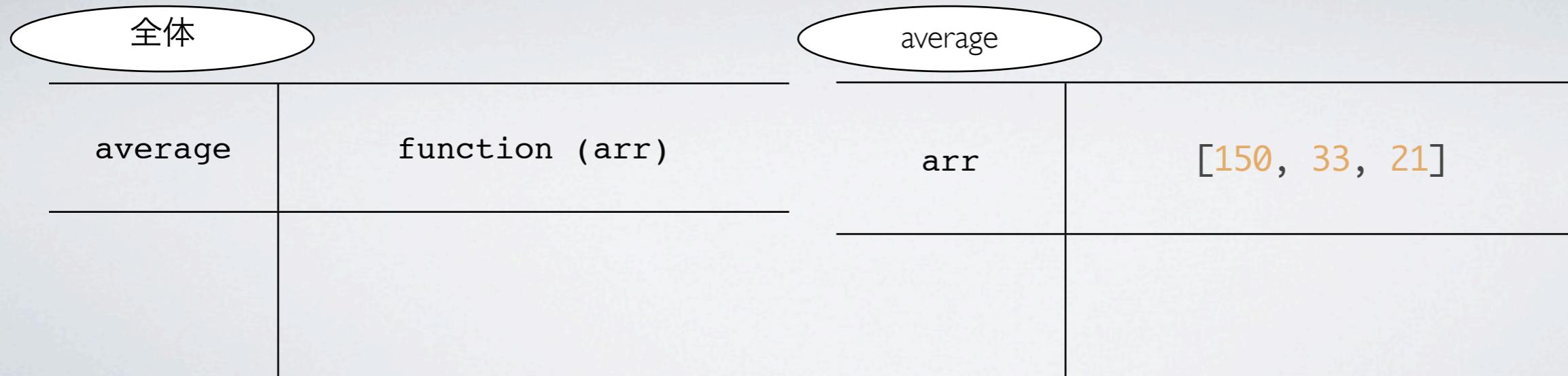
全体

average

function (arr)

# Function

```
function average (arr) {  
    var sum = 0,  
        i = 0,  
        len = arr.length;  
    for (; i < len; i = i + 1) {  
        sum = sum + arr[i];  
    }  
    return sum / len;  
}  
console.log(average([10, 20, 30, 40])); // -> 25  
console.log(average([150, 33, 21]));
```



# Function

```
function average (arr) {  
    var sum = 0,  
        i = 0,  
        len = arr.length;  
    for (; i < len; i = i + 1) {  
        sum = sum + arr[i];  
    }  
    return sum / len;  
}  
console.log(average([10, 20, 30, 40])); // -> 25  
console.log(average([150, 33, 21]));
```

全体	average
average	function (arr)
	arr [150, 33, 21]
	sum 0
	i 0
	len 3

# Function

```
function average (arr) {  
    var sum = 0,  
        i = 0,  
        len = arr.length;  
    for (; i < len; i = i + 1) {  
        sum = sum + arr[i];  
    }  
    return sum / len;  
}  
console.log(average([10, 20, 30, 40])); // -> 25  
console.log(average([150, 33, 21]));
```

全体

average	function (arr)

average

arr	[150, 33, 21]
sum	204
i	3
len	3

# Function

```
function average (arr) {  
  var sum = 0,  
      i = 0,  
      len = arr.length;  
  for (; i < len; i = i + 1) {  
    sum = sum + arr[i];  
  }  
  return sum / len;  
}  
console.log(average([10, 20, 30, 40])); // -> 25  
console.log(average([150, 33, 21])); // -> 68
```

全体

average	function (arr)

# Function lesson

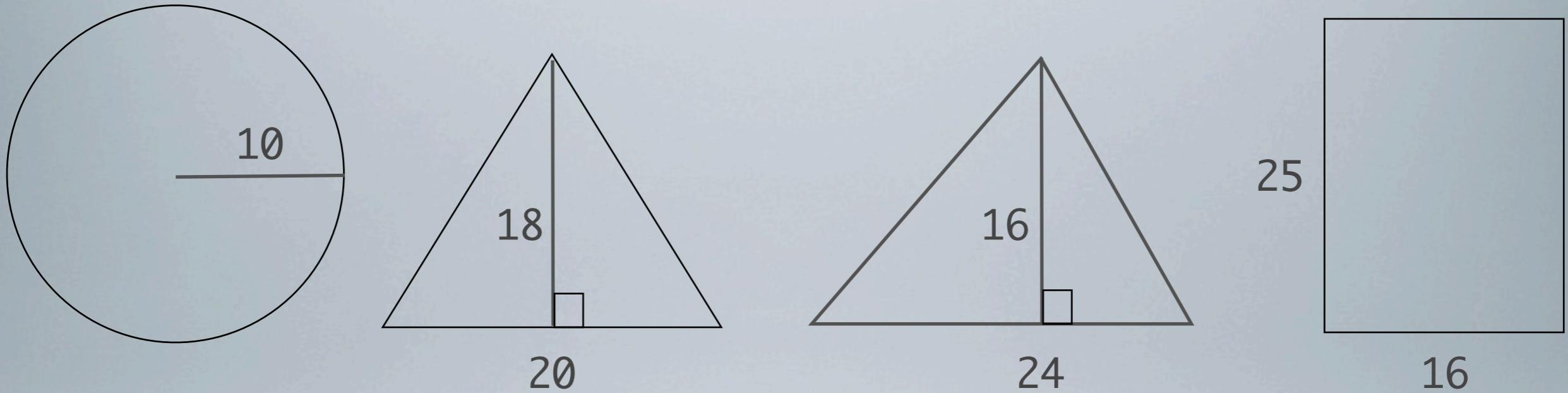
1. 数値を一つ引数にとり、  
+10して返す関数を作る
2. 上記関数の呼び出し結果を変数に入れず、  
`console.log()`の中で直接関数を呼び出す

# Function lesson

1. 数値を一つ引数にとり、  
+10して返す関数を作る
2. 上記関数の呼び出し結果を変数に入れず、  
`console.log()`の中で直接関数を呼び出す

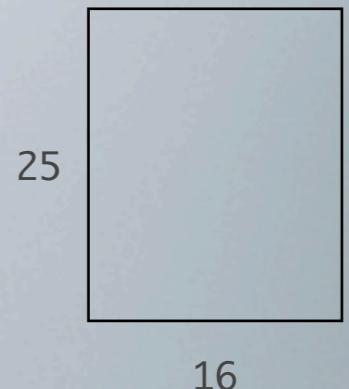
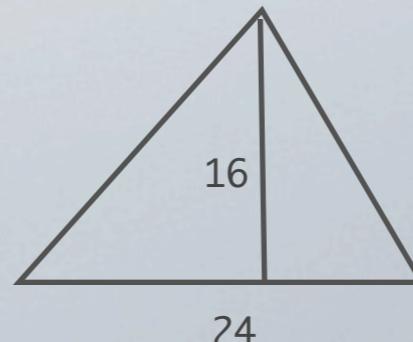
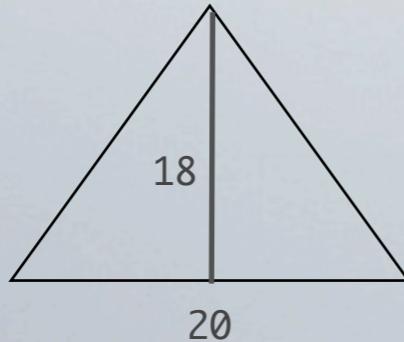
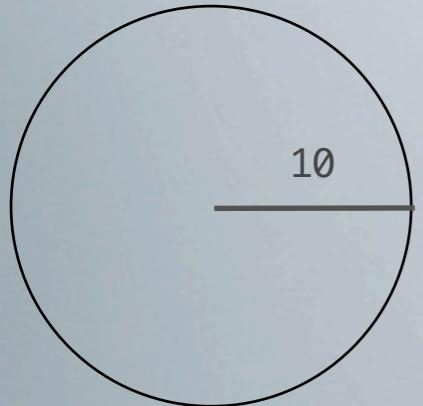
- ※ ちゃんと出力されましたか？
- ※ 変数の宣言で「var」を忘れてませんか？
- ※ セミコロン忘れてませんか？

# Syntax last challenge



1. 上記それぞれの図形をオブジェクト型で表現し、`shapes`という変数に配列として持たせる
2. 円、三角形、四角形の面積を求める関数を作成する
3. `for`文を使って 1で作った`shapes`の各要素に対し、2で作った関数を用いて、`area(面積)`を追加する

# Hint 1/16



1. 上記それぞれの図形をオブジェクト型で表現し、`shapes`という変数に配列として持たせる

```
// 半径が10の円を表現
```

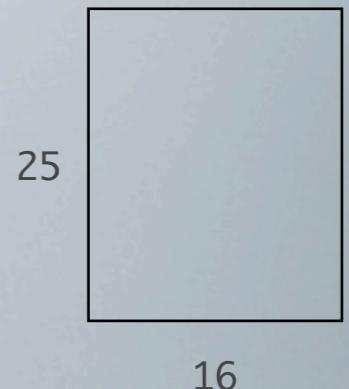
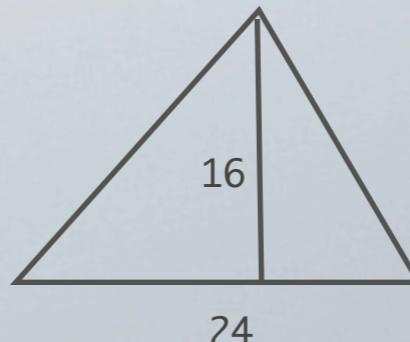
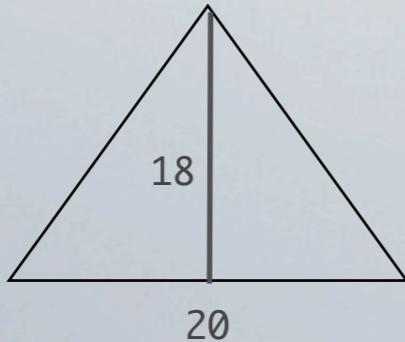
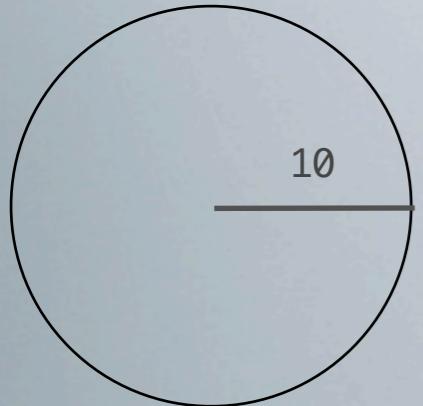
```
var circle_1 = {  
    hankei: 10  
}
```

```
// 底辺20、高さ18の三角形を表現
```

```
var triangle_1 = {  
    teihen: 20,  
    takasa: 18  
}
```

```
... // 上の要領で続きを書く
```

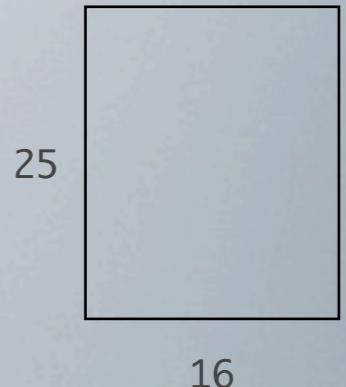
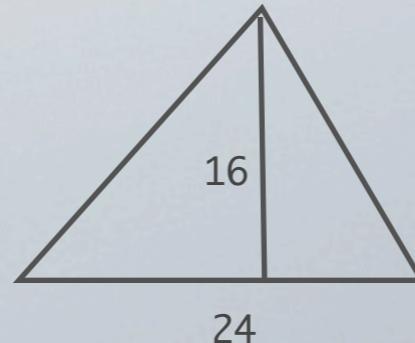
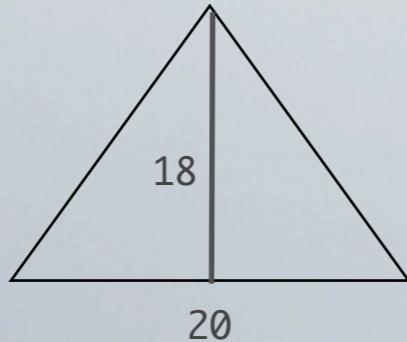
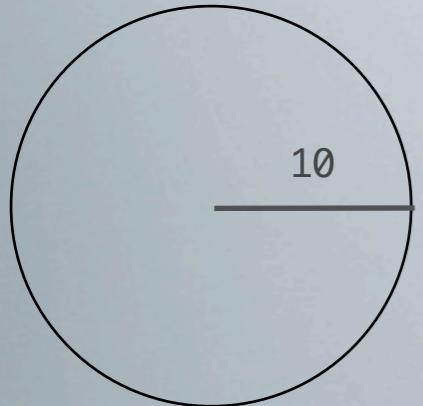
# Hint 2/16



1. 上記それぞれの図形をオブジェクト型で表現し、**shapes**という変数に配列として持たせる

```
var shapes = [circle_1, triangle_1, triangle_2, rectangle_1];
console.log(shapes);
/*
[ { hankei: 10 },
  { teihen: 20, takasa: 18 },
  { teihen: 24, takasa: 16 },
  { tate: 25, yoko: 16 } ]
```

# Hint 3/16

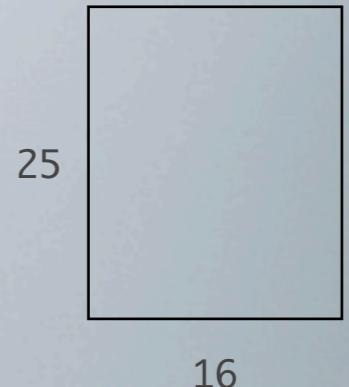
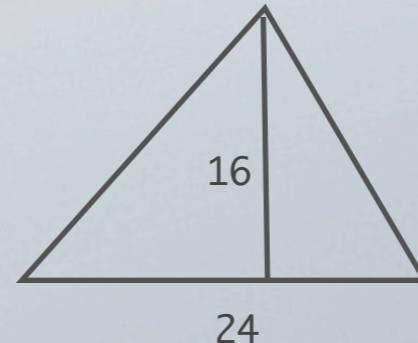
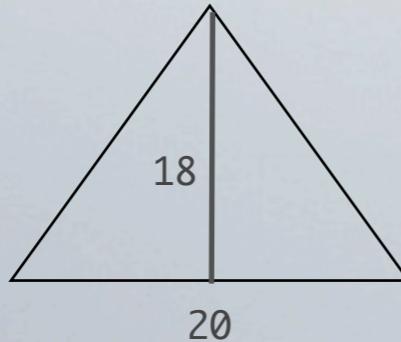
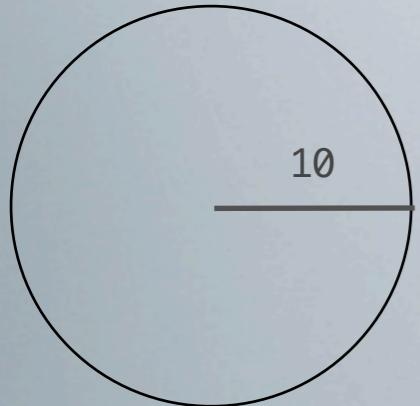


2. 円、三角形、四角形の面積を求める関数を作成する

```
// 円の面積を求める関数
function getCircleArea (circleObj/*{ hankei: 数値 }*/) {
    // 半径の値はcircleObj.hankeiでアクセス可能
    // ここに面積を求める処理を書く
}

// getCircleAreaがちゃんと動作するか確認
console.log(getCircleArea({ hankei: 10 }));
```

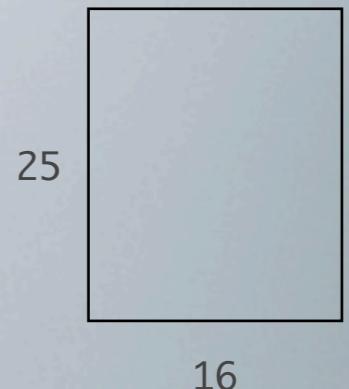
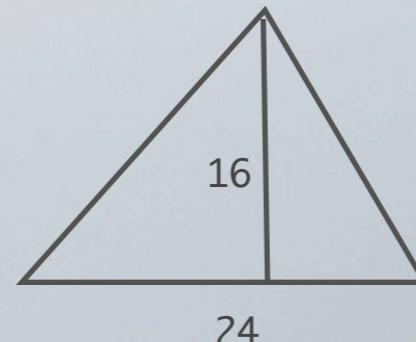
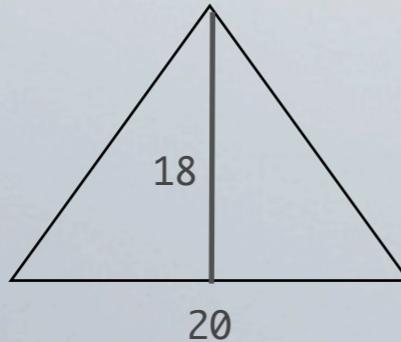
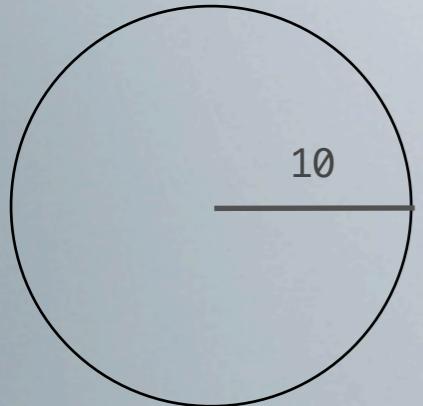
# Hint 4/16



## 2. 円、三角形、四角形の面積を求める関数を作成する

```
// 円の面積を求める関数  
function getCircleArea (circleObj/*{ hankei: 数値 }*/){  
    // 半径の値はcircleObj.hankeiでアクセス可能  
    var area = circleObj.hankei * circleObj.hankei * 3.14;  
    return area;  
}  
  
// getCircleAreaがちゃんと動作するか確認  
console.log(getCircleArea({ hankei: 10 })); // -> 314
```

# Hint 5/16

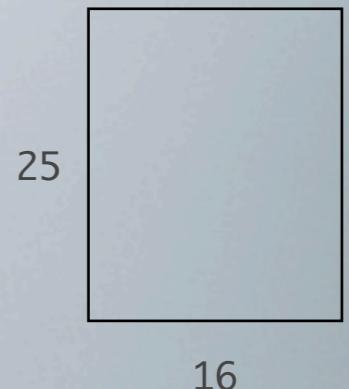
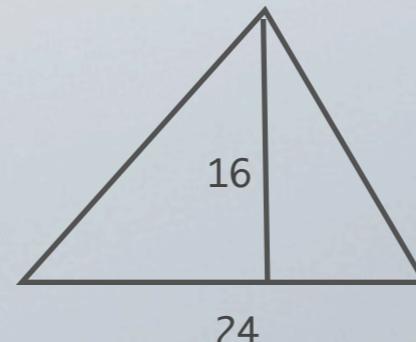
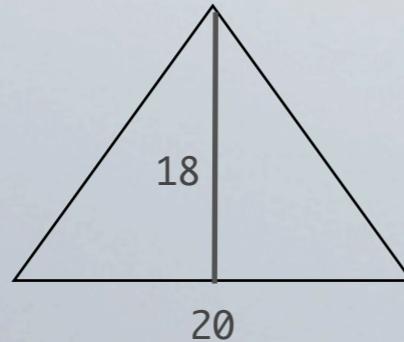
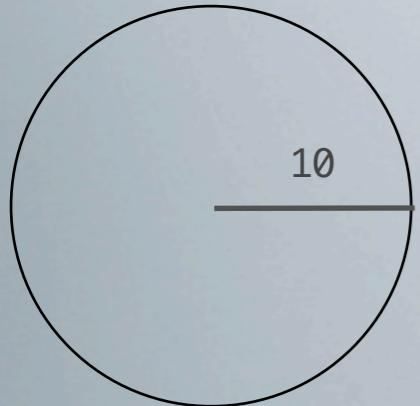


2. 円、三角形、四角形の面積を求める関数を作成する

```
// 三角形の面積を求める関数
function getTriangleArea (triangleObj/*{ teien: 数値, takasa: 数値 }*/) {
    // 三角形の面積は 底辺 × 高さ ÷ 2 で求める
    // ここに面積を求める処理を書く
}

// getTriangleAreaがちゃんと動作するか確認
console.log(getTriangleArea({ teien: 20, takasa: 18 }));
```

# Hint 6/16

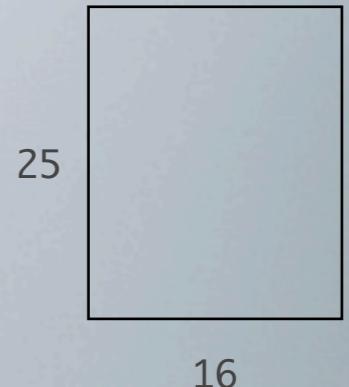
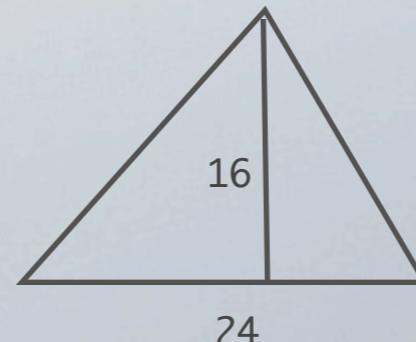
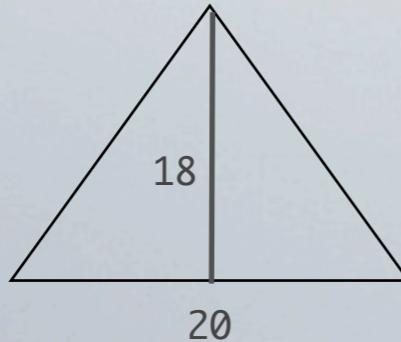
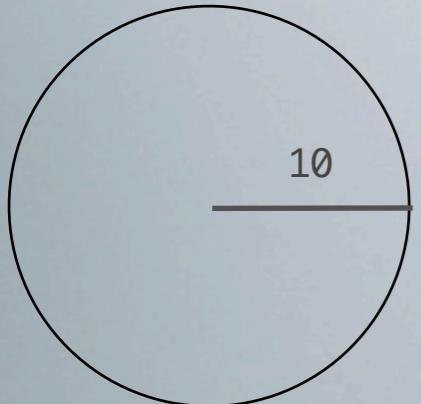


## 2. 円、三角形、四角形の面積を求める関数を作成する

```
// 三角形の面積を求める関数
function getTriangleArea (triangleObj/*{ teihen: 数値, takasa: 数値 }*/) {
    // 三角形の面積は 底辺 × 高さ ÷ 2 で求める
    var area = triangleObj.teihen * triangleObj.takasa / 2;
    return area;
}

// getTriangleAreaがちゃんと動作するか確認
console.log(getTriangleArea({ teihen: 20, takasa: 18 })); // -> 180
```

# Hint 7/16

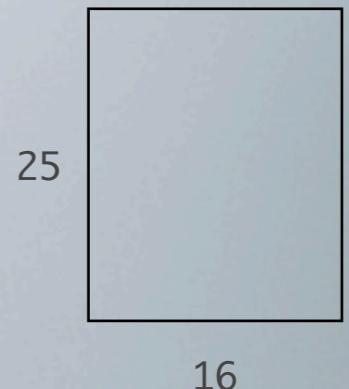
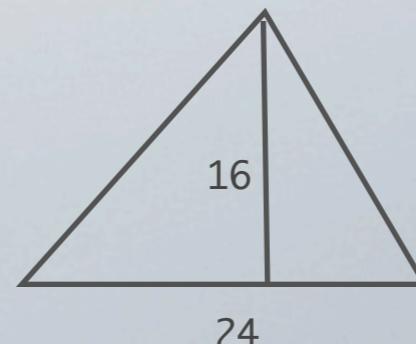
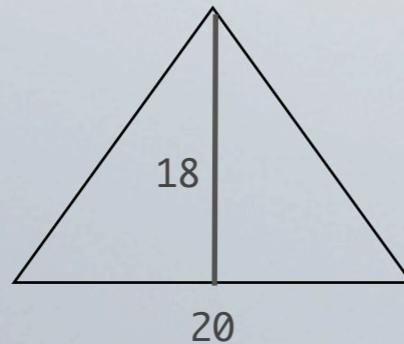
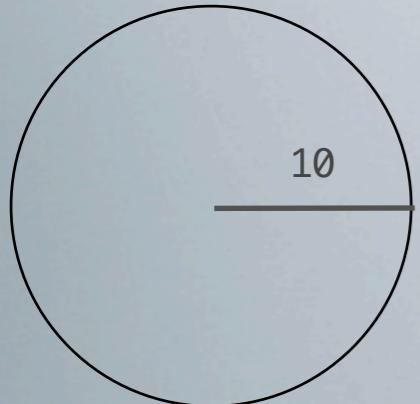


2. 円、三角形、四角形の面積を求める関数を作成する

```
// 四角形の面積を求める関数
function getRectangleArea (rectangleObj/*{ tate: 数値, yoko: 数値 }*/) {
    // 四角形の面積は 縦 × 横 で求める
    // ここに面積を求める処理を書く
}

// getTriangleAreaがちゃんと動作するか確認
console.log(getRectangleArea({ tate: 25, yoko: 16 }));
```

# Hint 8/16

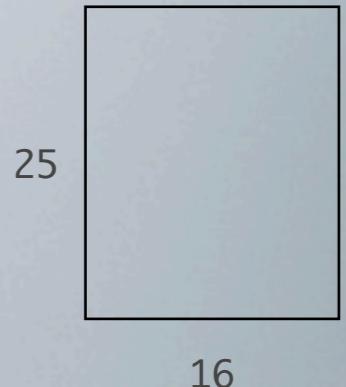
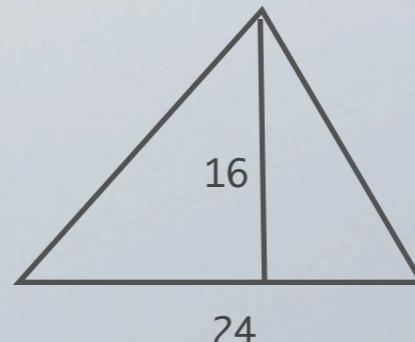
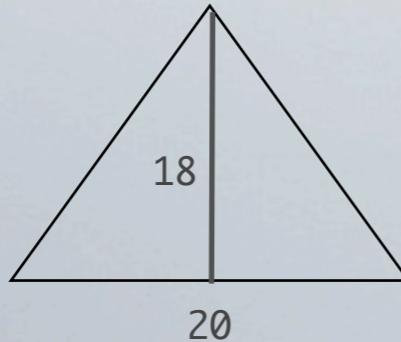
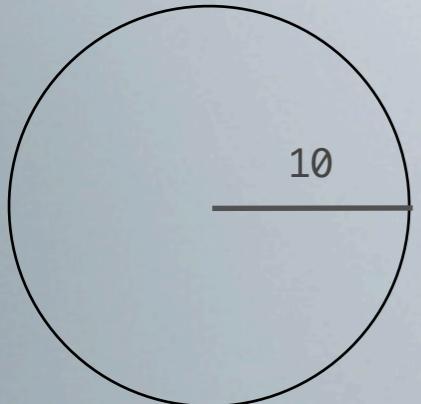


## 2. 円、三角形、四角形の面積を求める関数を作成する

```
// 四角形の面積を求める関数
function getRectangleArea (rectangleObj/*{ tate: 数値, yoko: 数値 }*/) {
    // 四角形の面積は 縦 × 横 で求める
    var area = rectangleObj.tate * rectangleObj.yoko;
    return area;
}

// getTriangleAreaがちゃんと動作するか確認
console.log(getRectangleArea({ tate: 25, yoko: 16 })); // -> 400
```

# Hint 9/16

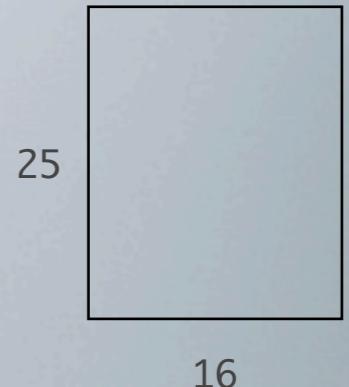
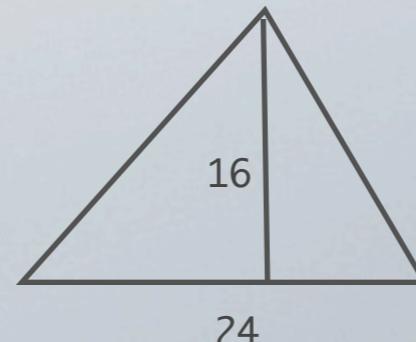
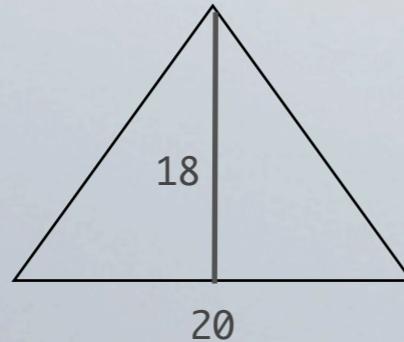
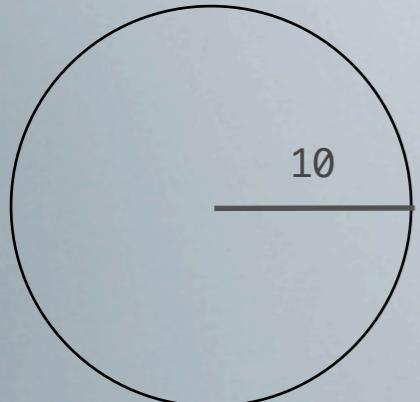


3. for文を使って 1で作ったshapesの各要素に対し、2で作った関数を用いて、area(面積)を追加する

```
var shapes = [circle_1, triangle_1, triangle_2, rectangle_1];

for (*初期化式; 条件式; 变化式*) {
    // 下みたいにfor文の中でshapesの要素にアクセスしたい
    console.log(shapes[i]);
}
```

# Hint 10/16

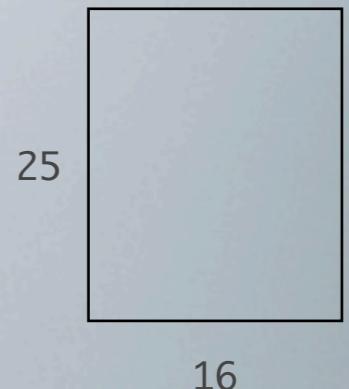
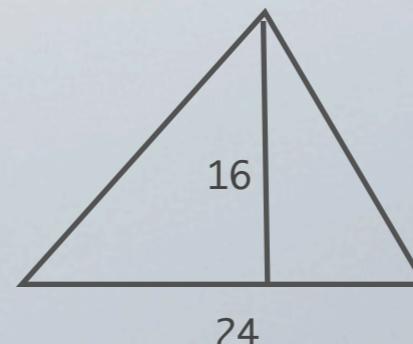
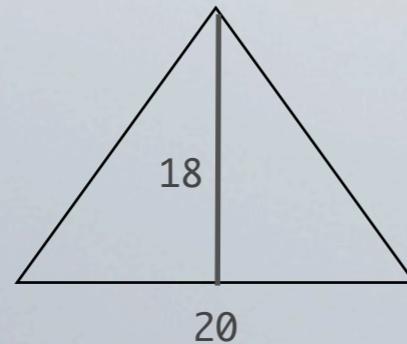
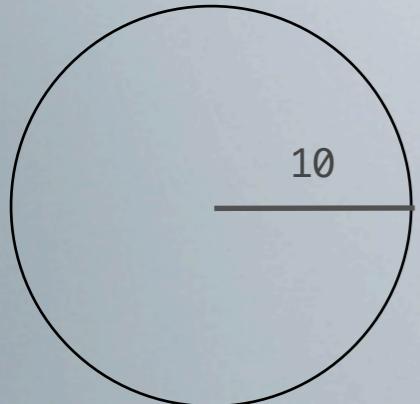


3. for文を使って 1で作ったshapesの各要素に対し、2で作った関数を用いて、area(面積)を追加する

```
var shapes = [circle_1, triangle_1, triangle_2, rectangle_1];

for (var i = 0, j = shapes.length; i < j; i = i + 1) {
    // 下みたいにfor文の中でshapesの要素にアクセスしたい
    console.log(shapes[i]);
}
```

# Hint 11/16



3. for文を使って 1で作ったshapesの各要素に対し、  
2で作った関数を用いて、area(面積)を追加する

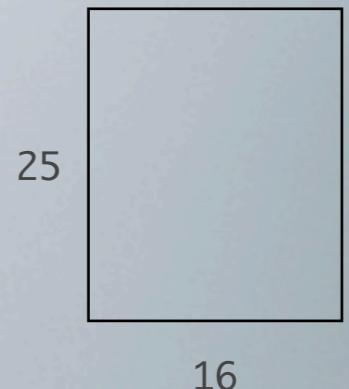
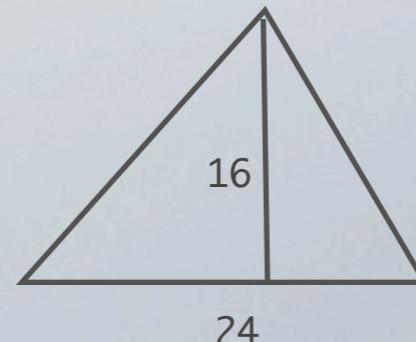
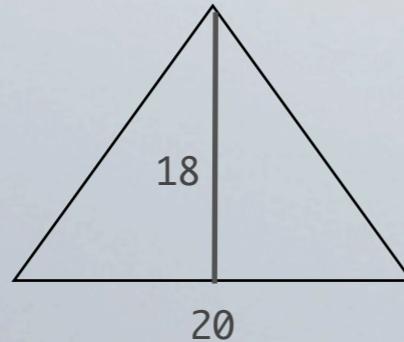
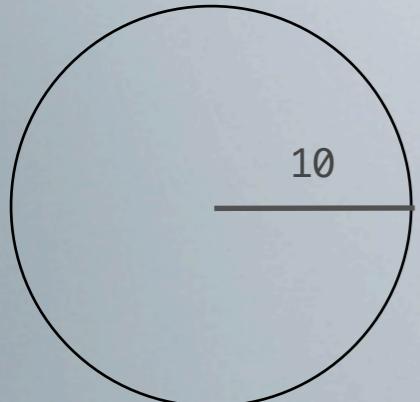
```
var shapes = [circle_1, triangle_1, triangle_2, rectangle_1];

for (var i = 0, j = shapes.length; i < j; i = i + 1) {
    // もしshapes[i]が円だったら
    shapes[i].area = getCircleArea(shapes[i]);

    // もしshapes[i]が三角形だったら
    shapes[i].area = getTriangleArea(shapes[i]);

    // もしshapes[i]が四角形だったら
    shapes[i].area = getRectangleArea(shapes[i]);
}
```

# Hint 12/16



3. for文を使って 1で作ったshapesの各要素に対し、  
2で作った関数を用いて、area(面積)を追加する

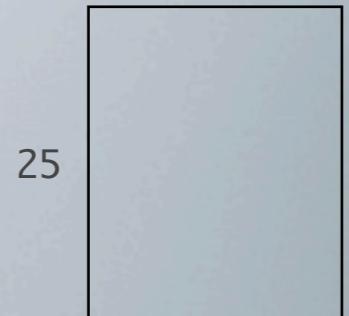
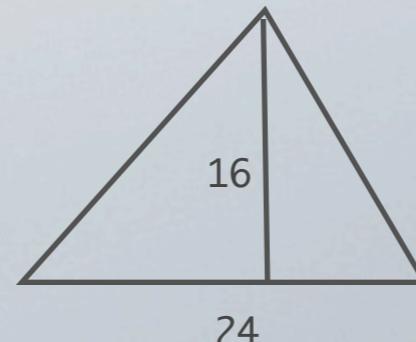
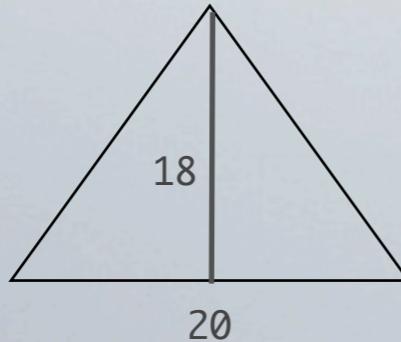
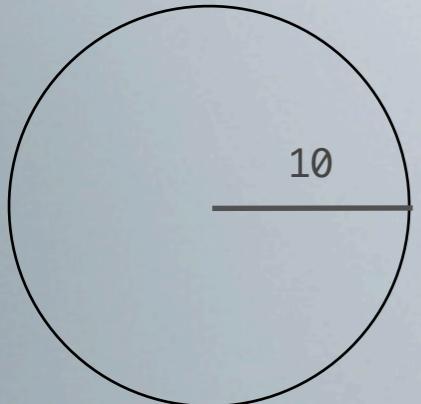
```
var shapes = [circle_1, triangle_1, triangle_2, rectangle_1];

// for文の中で、面積を求める関数に割り振るために
// それぞれのオブジェクトにshapeというプロパティをセットする

circle_1.shape = "circle";
triangle_1.shape = "triangle";
triangle_2.shape = "triangle";
rectangle_1.shape = "rectangle";

for (var i = 0, j = shapes.length; i < j; i = i + 1) {
    ...
}
```

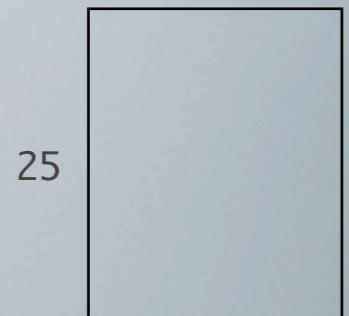
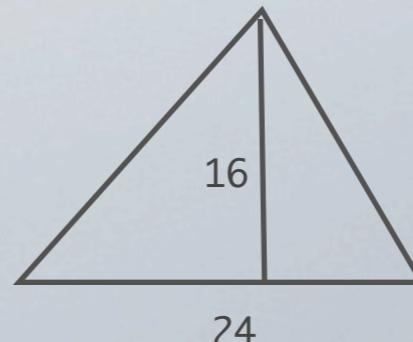
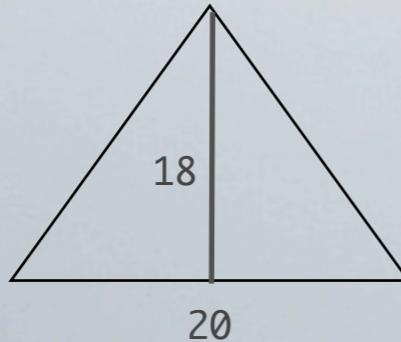
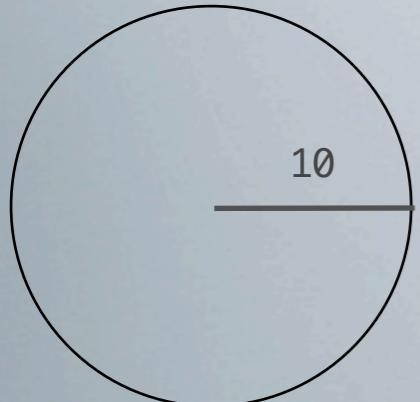
# Hint 13/16



3. for文を使って 1で作ったshapesの各要素に対し、  
**2で作った関数を用いて、area(面積)を追加する**

```
for (var i = 0, j = shapes.length; i < j; i = i + 1) {  
    // もしshapes[i]が円だったら  
    shapes[i].area = getCircleArea(shapes[i]);  
}
```

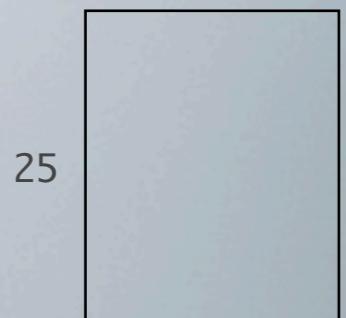
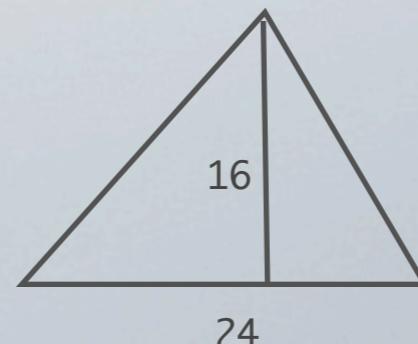
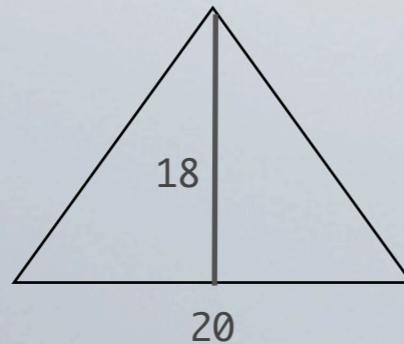
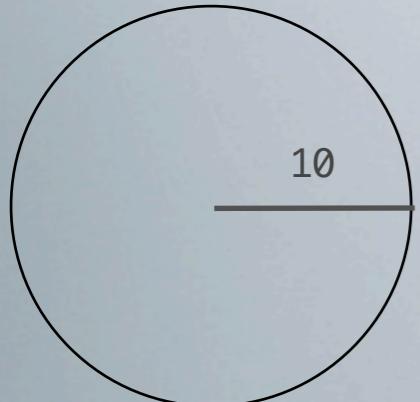
# Hint 14/16



3. for文を使って 1で作ったshapesの各要素に対し、  
**2で作った関数を用いて、area(面積)を追加する**

```
for (var i = 0, j = shapes.length; i < j; i = i + 1) {  
    // もしshapes[i]が円だったら  
    if (shapes[i].shape === "circle") {  
        shapes[i].area = getCircleArea(shapes[i]);  
    }  
    // もしshapes[i]が三角形だったら  
    shapes[i].area = getTriangleArea(shapes[i]);  
}
```

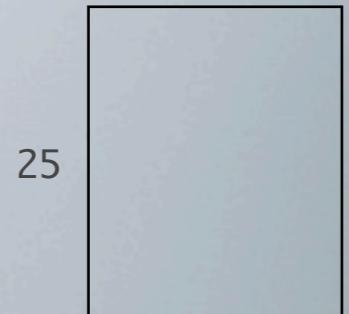
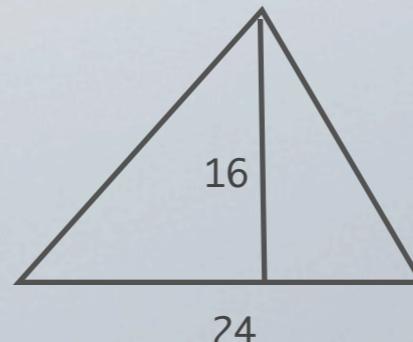
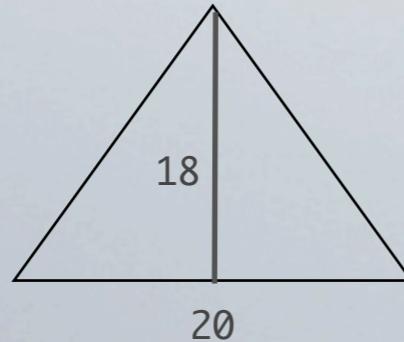
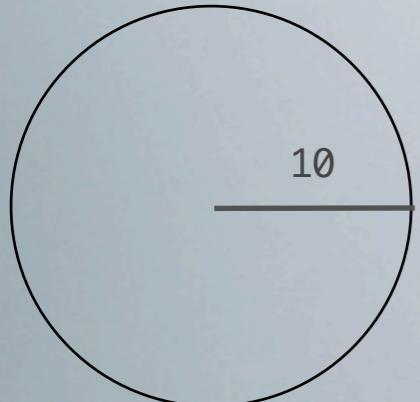
# Hint 15/16



3. for文を使って 1で作ったshapesの各要素に対し、  
**2で作った関数を用いて、area(面積)を追加する**

```
for (var i = 0, j = shapes.length; i < j; i = i + 1) {  
    // もしshapes[i]が円だったら  
    if (shapes[i].shape === "circle") {  
        shapes[i].area = getCircleArea(shapes[i]);  
    }  
    // もしshapes[i]が三角形だったら  
    if (shapes[i].shape === "triangle") {  
        shapes[i].area = getTriangleArea(shapes[i]);  
    }  
    // もしshapes[i]が四角形だったら  
    shapes[i].area = getRectangleArea(shapes[i]);  
}
```

# Hint 16/16



3. for文を使って 1で作ったshapesの各要素に対し、<sup>16</sup>2で作った関数を用いて、area(面積)を追加する

```
for (var i = 0, j = shapes.length; i < j; i = i + 1) {  
    // もしshapes[i]が円だったら  
    if (shapes[i].shape === "circle") {  
        shapes[i].area = getCircleArea(shapes[i]);  
    }  
    // もしshapes[i]が三角形だったら  
    if (shapes[i].shape === "triangle") {  
        shapes[i].area = getTriangleArea(shapes[i]);  
    }  
    // もしshapes[i]が四角形だったら  
    if (shapes[i].shape === "rectangle") {  
        shapes[i].area = getRectangleArea(shapes[i]);  
    }  
}
```

# Answer 1/3

```
// 半径が10の円を表現
var circle_1 = {
  hankei: 10
}

// 底辺20、高さ18の三角形を表現
var triangle_1 = {
  teihen: 20,
  takasa: 18
}

// 底辺24、高さ16の三角形を表現
var triangle_2 = {
  teihen: 24,
  takasa: 16
}

// 縦25、横16の四角形を表現
var rectangle_1 = {
  tate: 25,
  yoko: 16
}

var shapes = [circle_1, triangle_1, triangle_2, rectangle_1];
```

# Answer 2/3

```
// 円の面積を求める関数
function getCircleArea (circleObj/*{ hankei: 数値 }*/) {
    // 半径の値はcircleObj.hankeiでアクセス可能
    var area = circleObj.hankei * circleObj.hankei * 3.14;
    return area;
}

// 三角形の面積を求める関数
function getTriangleArea (triangleObj/*{ teihen: 数値, takasa: 数値 }*/) {
    // 三角形の面積は 底辺 × 高さ ÷ 2 で求める
    var area = triangleObj.teihen * triangleObj.takasa / 2;
    return area;
}

// 四角形の面積を求める関数
function getRectangleArea (rectangleObj/*{ tate: 数値, yoko: 数値 }*/) {
    // 四角形の面積は 縦 × 横 で求める
    var area = rectangleObj.tate * rectangleObj.yoko;
    return area;
}
```

# Answer 3/3

```
// for文の中で、面積を求める関数に割り振るために  
// それぞれのオブジェクトにshapeというプロパティをセットする  
  
circle_1.shape = "circle";  
triangle_1.shape = "triangle";  
triangle_2.shape = "triangle";  
rectangle_1.shape = "rectangle";  
  
for (var i = 0, j = shapes.length; i < j; i = i + 1) {  
    // もしshapes[i]が円だったら  
    if (shapes[i].shape === "circle") {  
        shapes[i].area = getCircleArea(shapes[i]);  
    }  
  
    // もしshapes[i]が三角形だったら  
    if (shapes[i].shape === "triangle") {  
        shapes[i].area = getTriangleArea(shapes[i]);  
    }  
  
    // もしshapes[i]が四角形だったら  
    if (shapes[i].shape === "rectangle") {  
        shapes[i].area = getRectangleArea(shapes[i]);  
    }  
}  
  
console.log(shapes);
```

# Function Quiz

```
console.log(sum(10, 20)); // -> ?  
console.log(sum2(15, 25)); // -> ?
```

```
function sum () {  
    return arguments[0] + arguments[1];  
}
```

```
var sum2 = function () {  
    return arguments[0] + arguments[1];  
}
```

# Call by reference

```
var obj = {  
  a: 1  
};  
  
var copy = obj;  
copy.a = 20;  
  
console.log(obj.a); // -> 20
```

# Call by reference 2

```
var obj_1 = {  
  a: 123  
};
```

```
var obj_2 = {  
  a: 123  
};
```

```
console.log(obj_1 == obj_2); // -> false
```

# Call by reference 3

```
var arr = [1, 2, 3, 4, 5],  
    dummy = arr;  
  
function dummy_pop (dum) {  
    dum.pop();  
    dum.pop();  
    dum.pop();  
}  
  
dummy_pop(dummy);  
console.log(arr); // -> [1, 2]
```

# Quiz

```
function replace ( input ) {  
    input = {a: 100};  
}  
  
var obj = {a: 1};  
replace(obj);  
  
console.log(obj.a) // -> ?
```

# 2nd Contents

1. About Syntax
2. About DOM & Event
3. About jQuery
4. About jQuery UI

# と、その前に

Functionについて

JavaScriptにおいてFunctionは死ぬほど大切で  
かつ、少し癖があって難しいポイント

なのでもう少し詳しく解説します。

# Function

関数は変数に代入できる

```
function sum (a, b) {  
    return a + b;  
}  
  
var result = sum;  
console.log( result(10, 20) ); // -> 30;
```

# Function

関数は変数に代入できる

```
function sum (a, b) {  
    return a + b;  
}  
  
var result = sum;  
console.log( result(10, 20) );
```

---

sum	function (a, b)

---

# Function

関数は変数に代入できる

```
function sum (a, b) {  
    return a + b;  
}  
  
var result = sum;  
console.log( result(10, 20) );
```

sum	function (a, b)
result	sum

# Function

関数は変数に代入できる

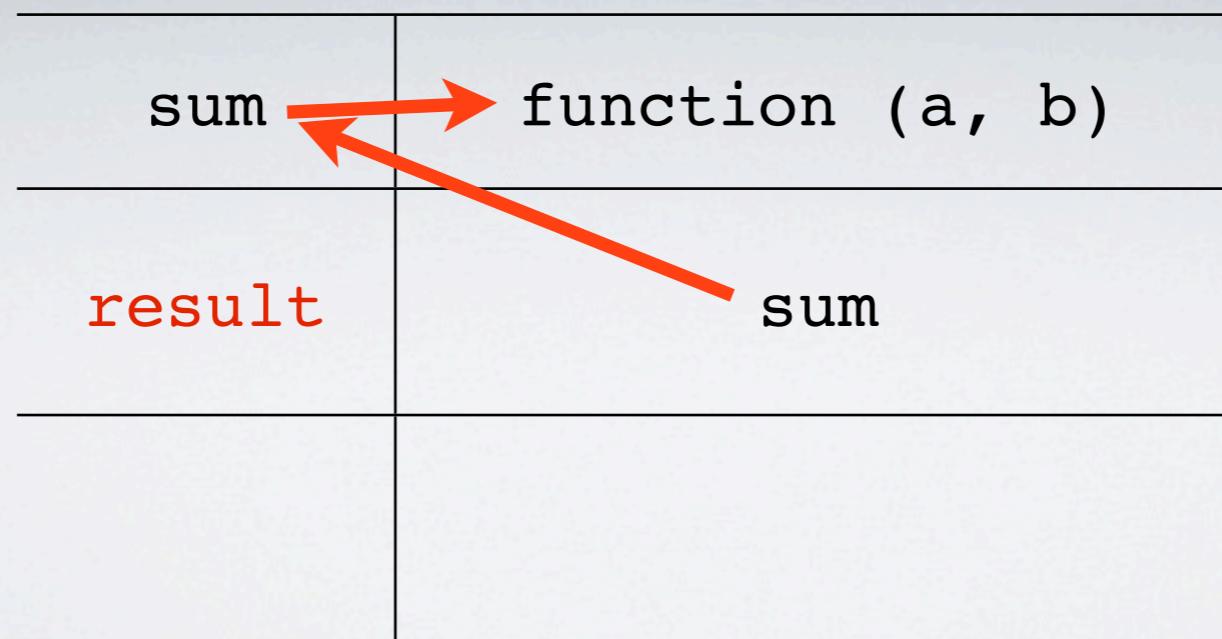
```
function sum (a, b) {  
    return a + b;  
}  
  
var result = sum;  
console.log( result(10, 20) );
```

sum	function (a, b)
result	sum

# Function

関数は変数に代入できる

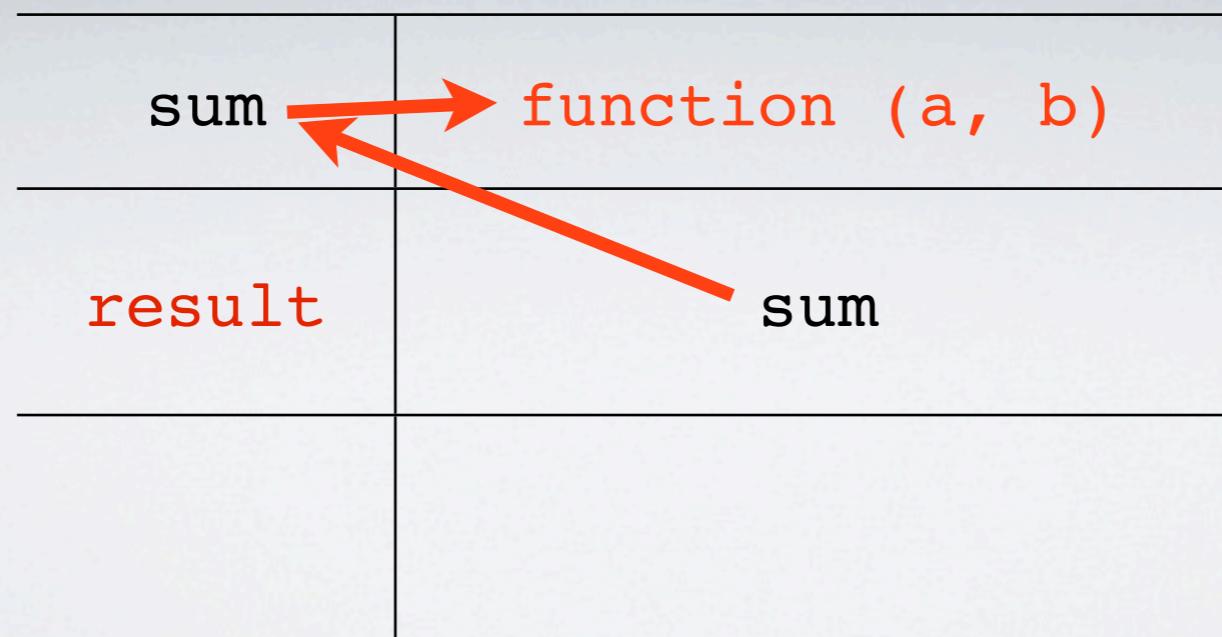
```
function sum (a, b) {  
    return a + b;  
}  
  
var result = sum;  
console.log( result(10, 20) );
```



# Function

関数は変数に代入できる

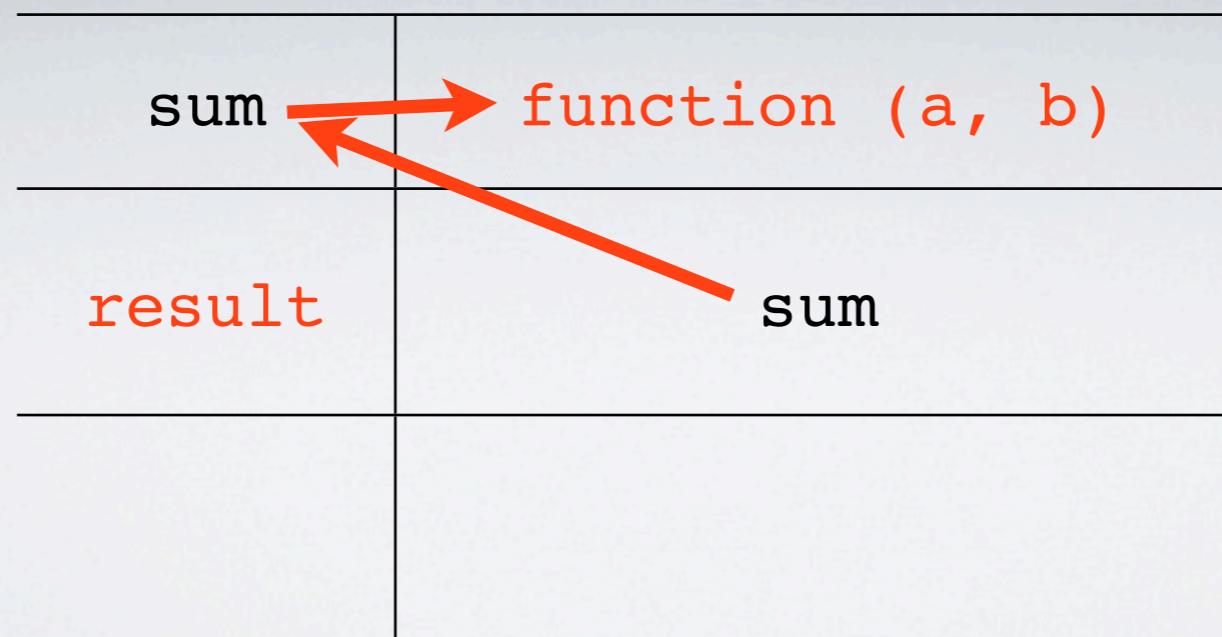
```
function sum (a, b) {  
    return a + b;  
}  
  
var result = sum;  
console.log( result(10, 20) );
```



# Function

関数は変数に代入できる

```
function sum (a, b) {  
    return a + b;  
}  
  
var result = sum;  
console.log( result(10, 20) ); // -> 30;
```



# Function

関数をオブジェクトの値とする事ができる

```
var circle_1 = {  
    hankei: 10,  
    getArea: function () {  
        return this.hankei * this.hankei * 3.14;  
    }  
}  
  
console.log(circle_1.getArea());
```

# Function

関数をオブジェクトの値とする事ができる

```
var circle_1 = {  
    hankei: 10,  
    getArea: function () {  
        return this.hankei * this.hankei * 3.14;  
    }  
}  
  
console.log(circle_1.getArea());
```

circle_1	hankei	10
	getArea	function()

# Function

関数をオブジェクトの値とする事ができる

```
var circle_1 = {  
    hankei: 10,  
    getArea: function () {  
        return this.hankei * this.hankei * 3.14;  
    }  
}  
  
console.log(circle_1.getArea());
```

circle_1	hankei	10
	getArea	function()

# Function

関数をオブジェクトの値とする事ができる

```
var circle_1 = {  
    hankei: 10,  
    getArea: function () {  
        return this.hankei * this.hankei * 3.14;  
    }  
}  
  
console.log(circle_1.getArea());
```

	hankei	10
circle_1	getArea	function()

# Function

関数をオブジェクトの値とする事ができる

```
var circle_1 = {  
hankei: 10,  
getArea: function () {  
    return this.hankei * this.hankei * 3.14;  
}  
}  
  
console.log(circle_1.getArea());
```

this は "自身" を指すキーワード  
ここでは this が含まれる circle\_1 自身を指す

# Function

例えばconsole.logとともに実は・・・

```
var console = {  
  log: function (data) {  
    // dataをコンソールに表示する  
  },  
  dir: function (data) {  
    // dataの詳細をコンソールに表示する  
  },  
  ...  
}
```

※ 本当は裏でもっと複雑なコードが書かれています

# First-class Function

JavaScriptのFunctionは第一級オブジェクトである

つまり、変数に格納ができたり、  
引数として渡すことも出来る

```
var result = function (callback) {  
    return callback(10, 20);  
};  
  
console.log(  
    result(function (a, b) {  
        return a + b;  
    })  
) // -> 30
```

# First-class Function

JavaScriptのFunctionは第一級オブジェクトである

つまり、変数に格納ができたり、  
引数として渡すことも出来る

```
var result = function (callback) {  
    return callback(10, 20);  
};  
  
console.log(  
    result(function (a, b) {  
        return a + b;  
    })  
) // -> 30
```

# First-class Function

当然こんな風に、  
「呼び出されたら関数を返す」関数なんかも書ける

```
var x_plus_y = function (x) {  
    return function (y) {  
        console.log(x + y);  
    };  
};  
  
x_plus_y(10)(20); // -> 30
```

こと関数を返す関数に関しては、  
面白い挙動を確認することが出来る

# Closure

```
function counter () {  
  var i = 0;  
  return function () {  
    console.log(i);  
    i++;  
  }  
}
```

```
cou = counter();  
cou(); //-> 0  
cou(); //-> 1  
cou(); //-> 2  
cou(); //-> 3
```

# Closure Quiz

```
(function () {
  for (var i = 0, j = 10; i < j; i++) {
    setTimeout(function () {
      console.log(i);
    }, 100);
  }
})();
```

上記の出力が0 1 2 3 4 ... となるよう、  
setTimeoutの中を書き換える

ヒント: setTimeoutは非同期なので100ms待っている間にfor文のループが進んでしまって現状の出力になっている

# 2nd Contents

1. About Syntax
2. About DOM & Event
3. About jQuery
4. About jQuery UI

# JavaScriptでしたい事

HTMLは文書と文書構造を定義する

CSSは文書の見せ方を定義する

JavaScriptは振る舞いを定義する

極論をいうと、JavaScriptはなくても良い

ないといけないページは作るべきでない

あくまで、ユーザビリティを向上するために

用いるのが吉

# そもそもDOMとは？

DOM(Document Object Model)

XMLやHTMLを操作するためのAPI  
仕様はW3Cにより策定されている

本来JavaScriptとは別物ですが、  
ブラウザ上のJavaScript環境では  
それらの基本的なAPI(関数)がサポートされている

要は、Webページの要素を  
JavaScriptで扱えるように表現したのがDOMってこと

# 1\_1.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>JavaScript</title>
  </head>
  <body>
    <h1>Color List</h1>
    <div>
      <ul>
        <li>Red</li>
        <li>Green</li>
        <li>Blue</li>
      </ul>
    </div>
    <script type="text/javascript">
// このscriptタグの間に書いてね！

    </script>
  </body>
</html>
```

# 要素アクセスAPI

JavaScriptでHTMLを操作するためには、  
まず、操作したい要素をオブジェクトとして取得する  
必要がある

```
var elem = document.querySelector("h1");
alert(elem.nodeName);

var rgb_list = document.querySelectorAll("div ul");
alert(rgb_list[0].nodeName);

alert(rgb_list[0].firstElementChild.textContent);
```

# 要素アクセスAPI

JavaScriptでHTMLを操作するためには、  
まず、操作したい要素をオブジェクトとして取得する  
必要がある

```
var elem = document.querySelector("h1");
alert(elem.nodeName);

var rgb_list = document.querySelectorAll("div ul");
alert(rgb_list[0].nodeName);

alert(rgb_list[0].firstElementChild.textContent);
```

# 1\_1.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>JavaScript</title>
  </head>
  <body>
    <h1>Color List</h1>
    <div>
      <ul>
        <li>Red</li>
        <li>Green</li>
        <li>Blue</li>
      </ul>
    </div>
    <script type="text/javascript">
// このscriptタグの間に書いてね！
      </script>
  </body>
</html>
```

```
var elem = document.querySelector("h1");
alert(elem.nodeName);

var rgb_list = document.querySelectorAll("div ul");
alert(rgb_list[0].nodeName);

alert(rgb_list[0].firstElementChild.textContent);
```

# 1\_1.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>JavaScript</title>
  </head>
  <body>
    <h1>Color List</h1>
    <div>
      <ul>
        <li>Red</li>
        <li>Green</li>
        <li>Blue</li>
      </ul>
    </div>
    <script type="text/javascript">
// このscriptタグの間に書いてね！
      </script>
    </body>
  </html>
```

```
var elem = document.querySelector("h1");
alert(elem.nodeName);

var rgb_list = document.querySelectorAll("div ul");
alert(rgb_list[0].nodeName);

alert(rgb_list[0].firstElementChild.textContent);
```



# 1\_1.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>JavaScript</title>
  </head>
  <body>
    <h1>Color List</h1>
    <div>
      <ul>
        <li>Red</li>
        <li>Green</li>
        <li>Blue</li>
      </ul>
    </div>
    <script type="text/javascript">
// このscriptタグの間に書いてね！
      </script>
    </body>
  </html>
```



rgb\_list[0].firstElementChild

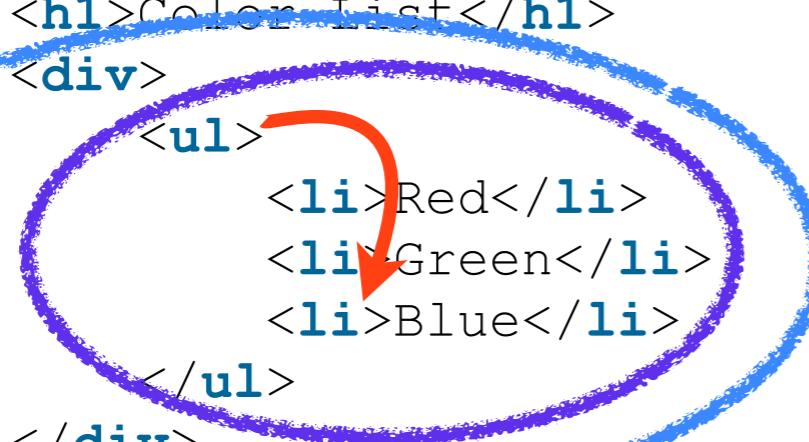
```
var elem = document.querySelector("h1");
alert(elem.nodeName);

var rgb_list = document.querySelectorAll("div ul");
alert(rgb_list[0].nodeName);

alert(rgb_list[0].firstElementChild.textContent);
```

# 1\_1.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>JavaScript</title>
  </head>
  <body>
    <h1>Color List</h1>
    <div>
      <ul>
        <li>Red</li>
        <li>Green</li>
        <li>Blue</li>
      </ul>
    </div>
    <script type="text/javascript">
// このscriptタグの間に書いてね！
      </script>
    </body>
  </html>
```



rgb\_list[0].lastElementChild

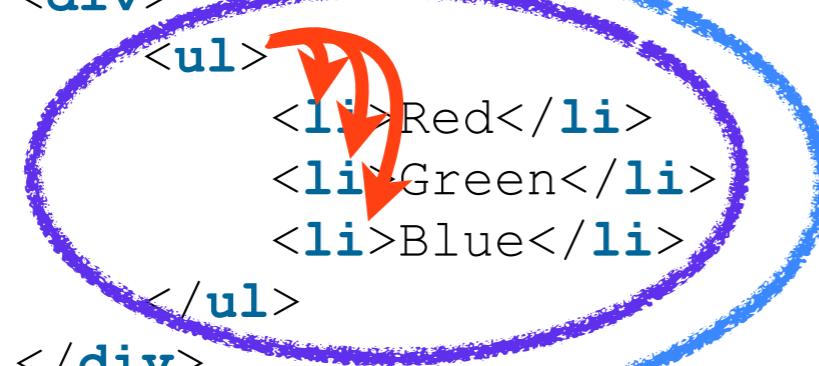
```
var elem = document.querySelector("h1");
alert(elem.nodeName);

var rgb_list = document.querySelectorAll("div ul");
alert(rgb_list[0].nodeName);

alert(rgb_list[0].firstElementChild.textContent);
```

# 1\_1.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>JavaScript</title>
  </head>
  <body>
    <h1>Color List</h1>
    <div>
      <ul>
        <li>Red</li>
        <li>Green</li>
        <li>Blue</li>
      </ul>
    </div>
    <script type="text/javascript">
// このscriptタグの間に書いてね！
      </script>
    </body>
  </html>
```



rgb\_list[0].children

```
var elem = document.querySelector("h1");
alert(elem.nodeName);

var rgb_list = document.querySelectorAll("div ul");
alert(rgb_list[0].nodeName);

alert(rgb_list[0].firstElementChild.textContent);
```

# 便利なデバッグ

JavaScriptを書く上で、  
現在変数の中にはどんな値が入っているのか？や、  
返ってきたオブジェクトはどんなプロパティ、メソッド  
を持っているのか等を確認するには、  
console.logやconsole.dirが便利！

確認はデバッガで行える

```
var elem = document.querySelector("#colorList");
console.log(elem.nodeName); //-> 'DIV'

var rgb_list = document.querySelectorAll(".RGB");
console.dir(rgb_list[0]);
```

# 便利なデバッグ

JavaScriptを書く上で、  
現在変数の中にはどんな値が入っているのか？や、  
返ってきたオブジェクトはどんなプロパティ、メソッド  
を持っているのか等を確認するには、  
console.logやconsole.dirが便利！

確認はデバッガで行える

```
var elem = document.querySelector("#colorList");
console.log(elem.nodeName); //-> 'DIV'

var rgb_list = document.querySelectorAll(".RGB");
console.dir(rgb_list[0]);
```

# 要素アクセスlesson

1. querySelectorAllを用いてli要素を配列で取得し、for文を用いて各要素のテキストをコンソールに出力しよう
2. querySelectorでbody要素を取得し、firstElementChildやchildrenプロパティを使って、<li>Green</li>のGreenをコンソールに出力しよう