THE // FLATIRON SCHOOL

Functions & Scope

- Declaring
- Calling
- Parameters
- Returning Data
- Scope

Functions

Javascript functions

In programming, a function holds a set of actions that will only run when we call that function. This ultimately helps us control the flow of our program and allows us to easily repeat a set of actions multiple times.

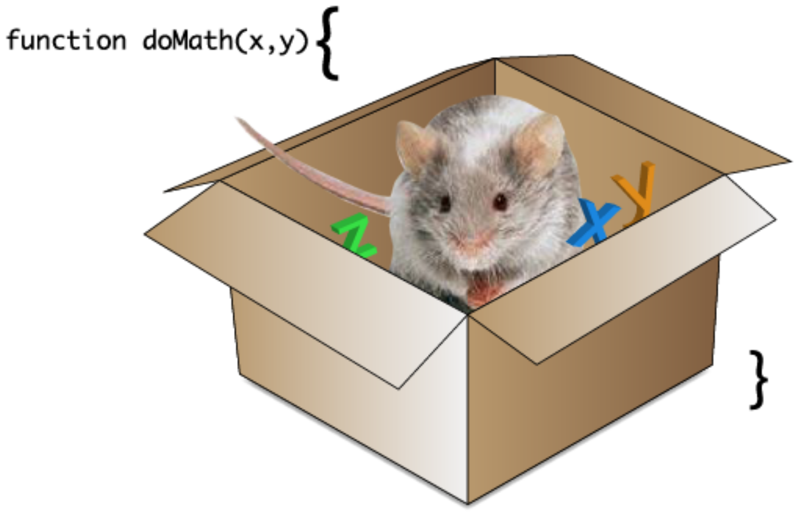
If you prefer a metaphor:

A function is kind of like placing a mouse in a box and that mouse will only perform the actions he was trained to do whenever you call his name.

DECLARING A FUNCTION

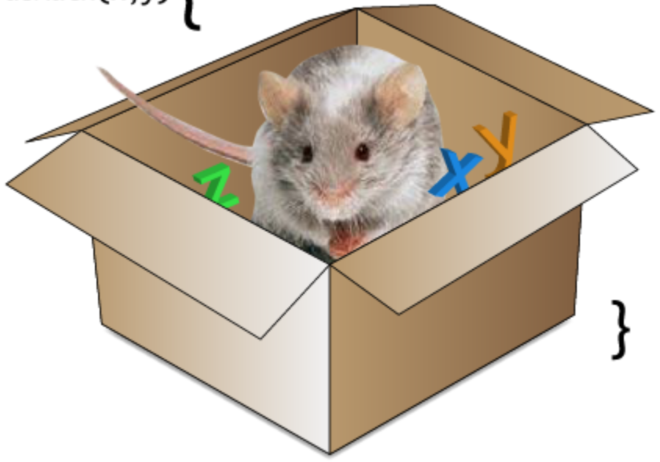
```
function doMath(x,y){



                                  }
```

DECLARING A FUNCTION



CALLING THE FUNCTION

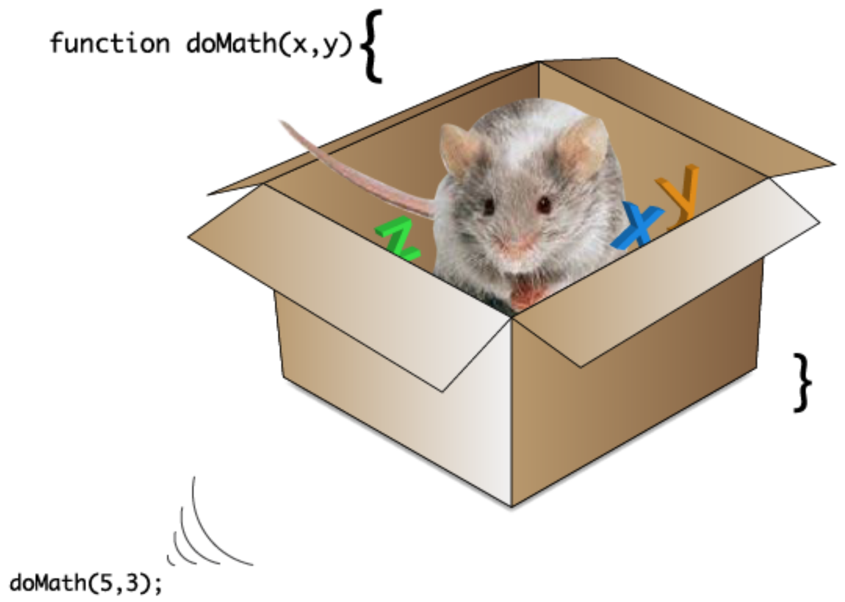
```
function doMath(x,y){
```



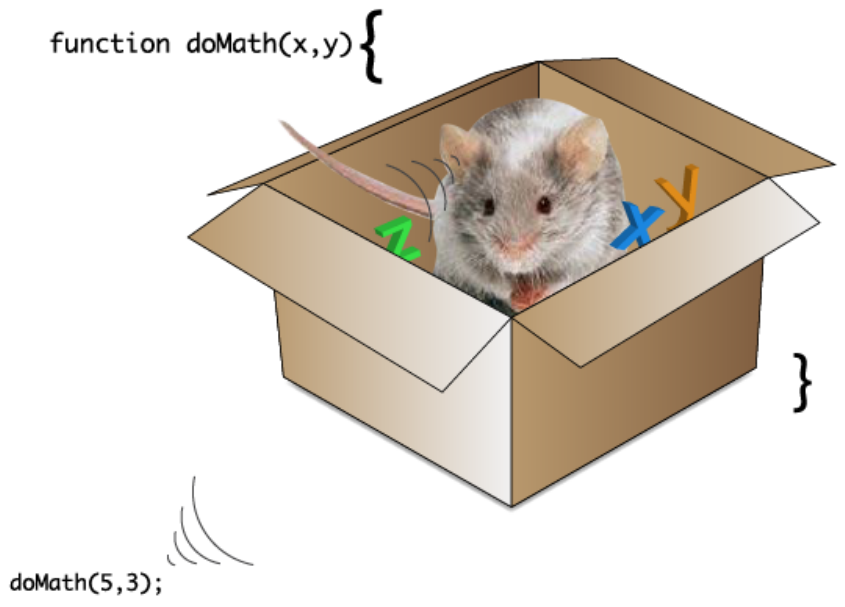
```
}
```

```
doMath(5,3);
```

CALLING THE FUNCTION

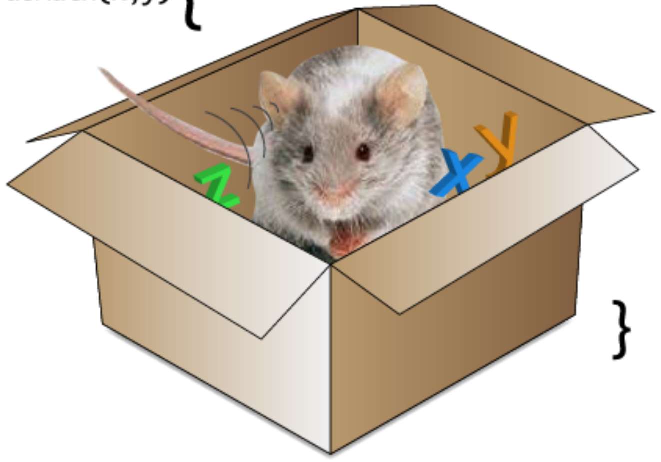


CALLING THE FUNCTION



CALLING THE FUNCTION

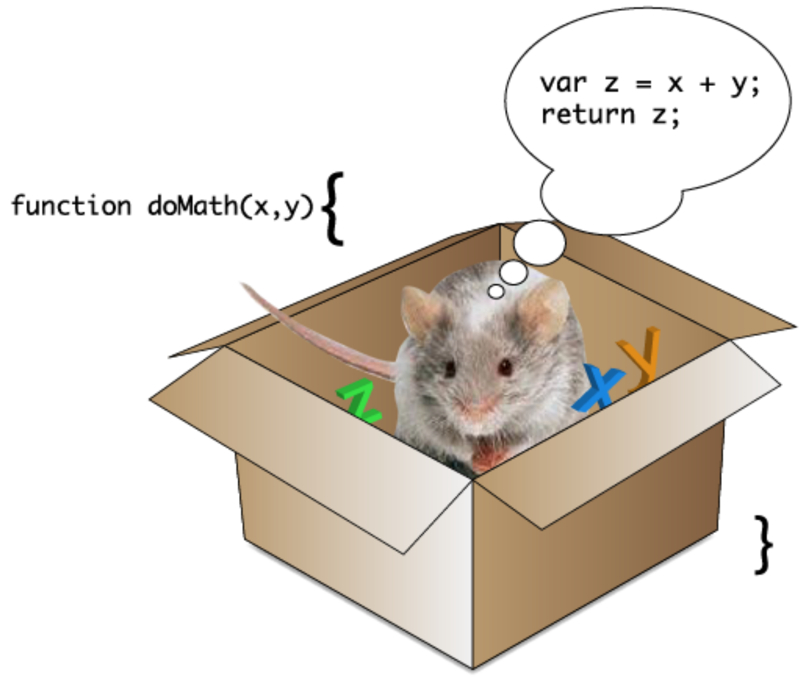
```
function doMath(x,y){
```



```
}
```

```
doMath(5,3);
```

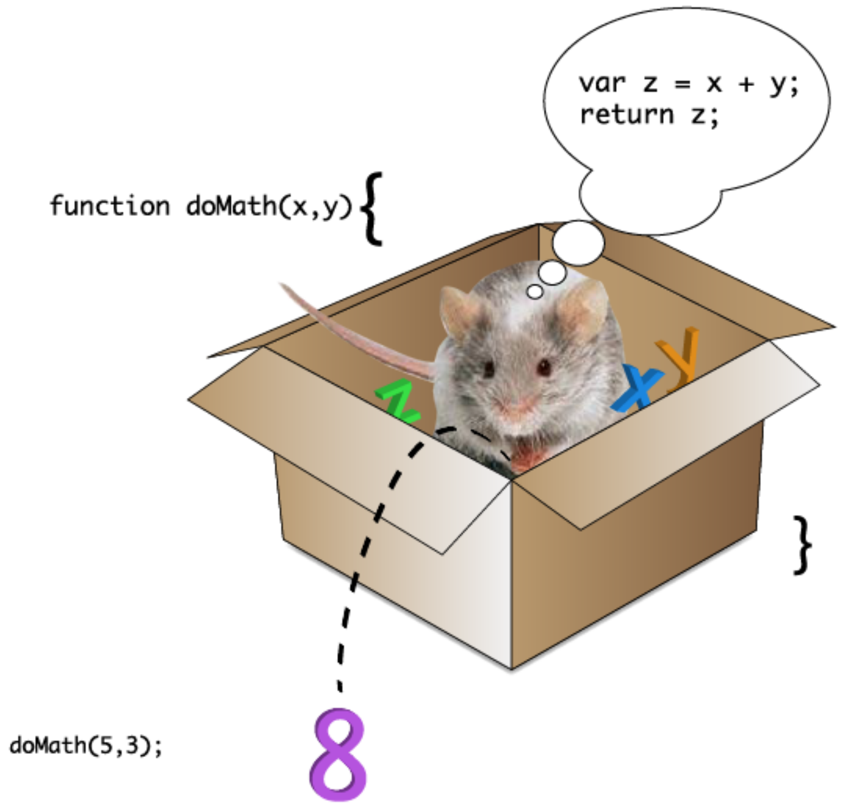
FUNCTION RUNNING FOR THE FIRST TIME



```
function doMath(x,y){
var z = x + y;
return z;
}
```

```
doMath(5,3);
```

Returning a Value

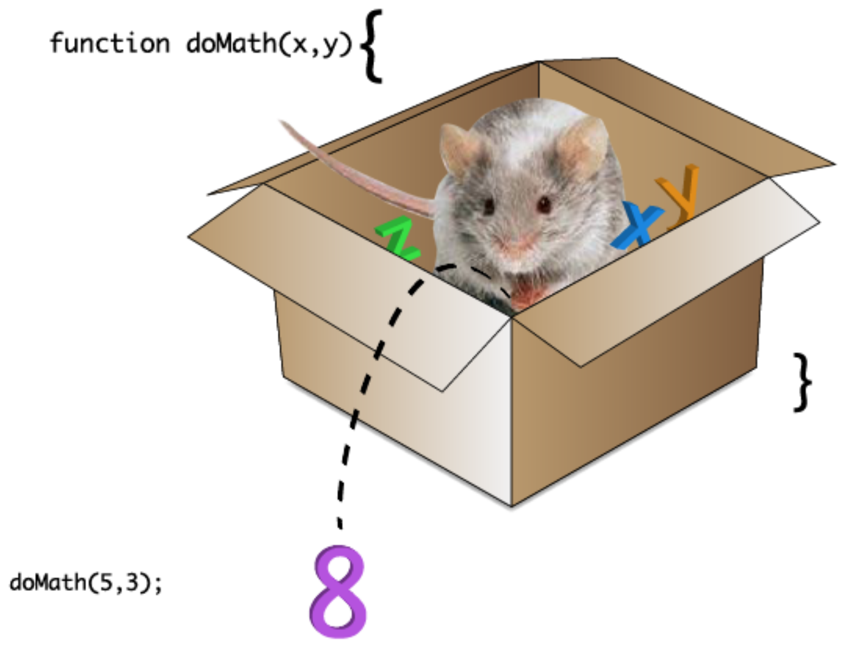


```
var z = x + y;
return z;
```

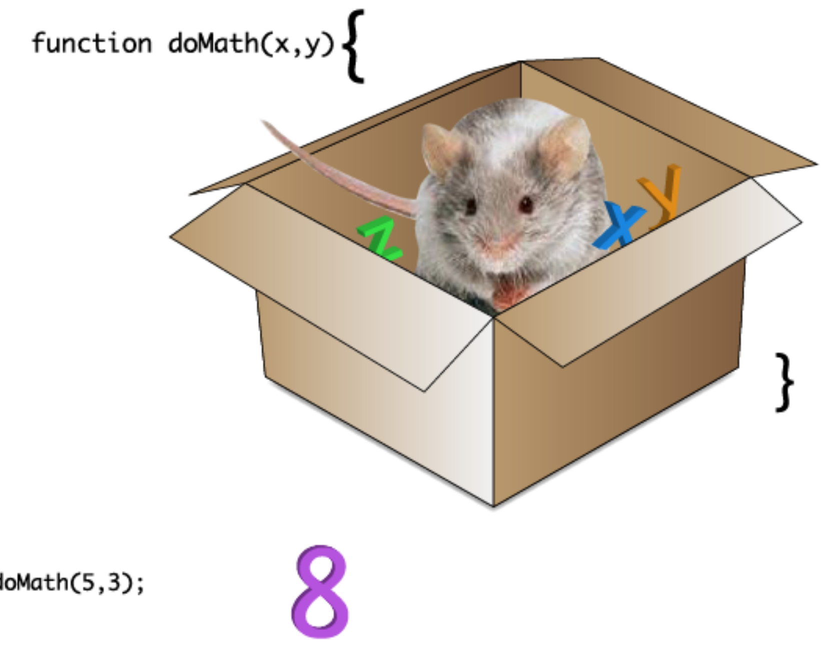
```
function doMath(x,y){
}
```

```
doMath(5,3);
```

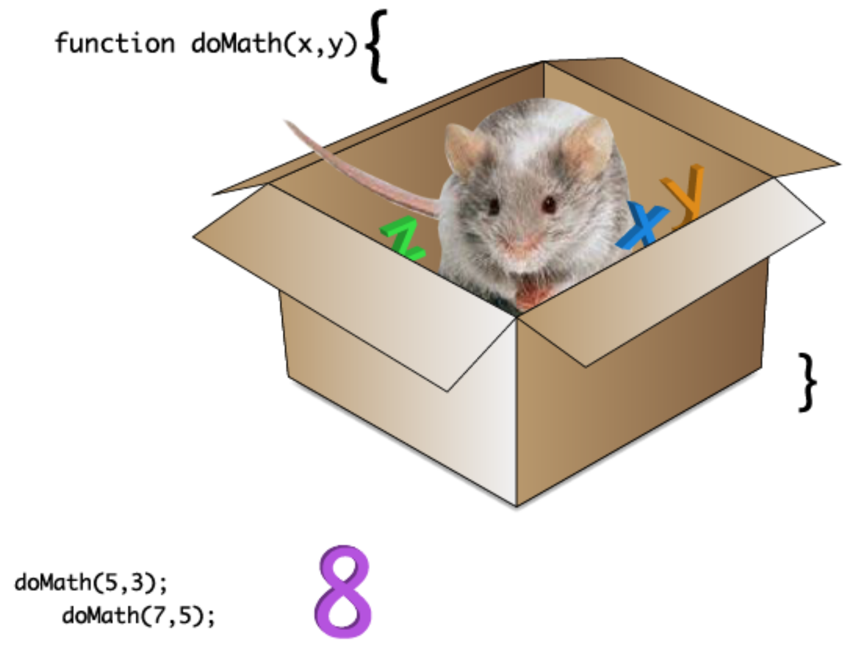
Returning a Value



Returning a Value



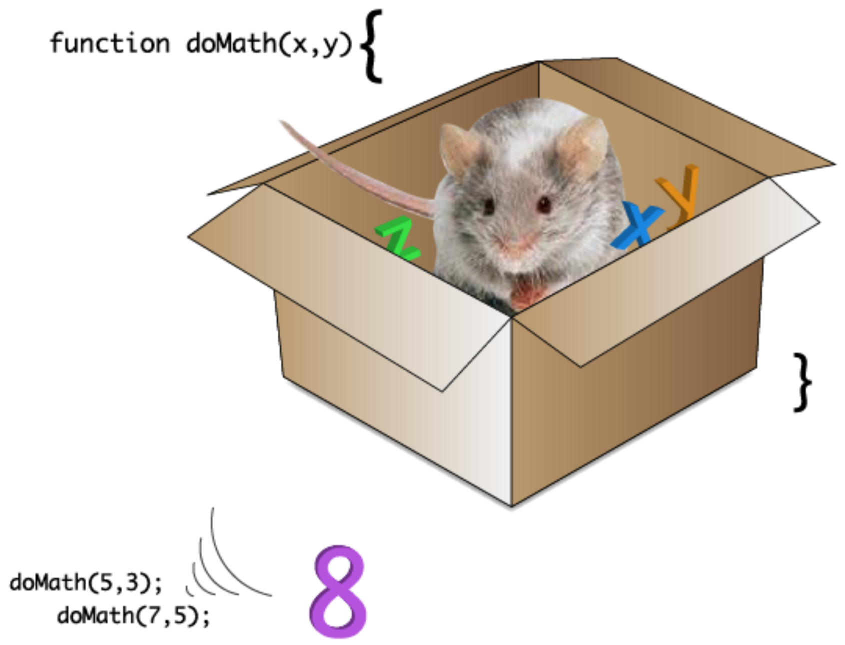
CALLING THE FUNCTION AGAIN With Different Parameters



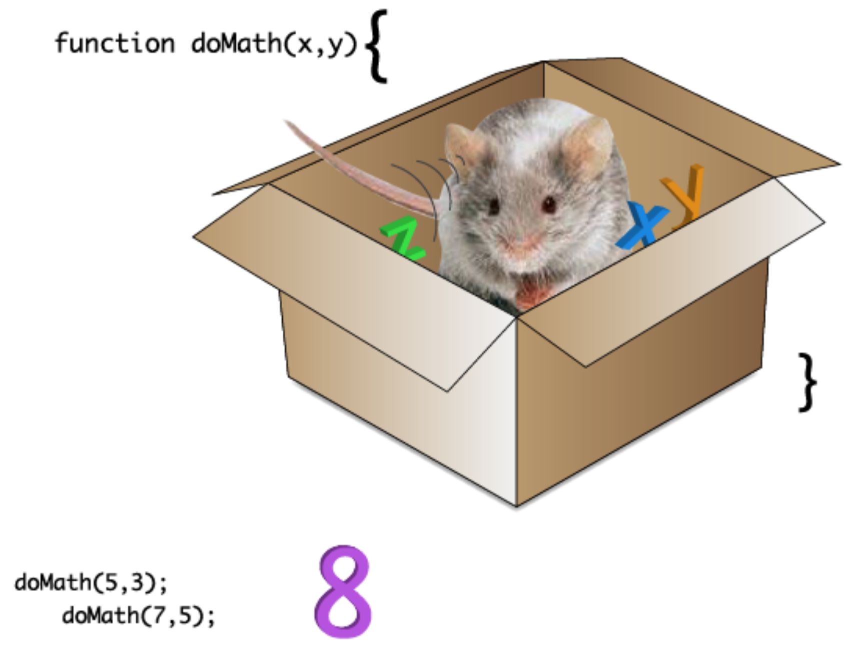
```
function doMath(x,y){
}
```

```
doMath(5,3);
    doMath(7,5);
```

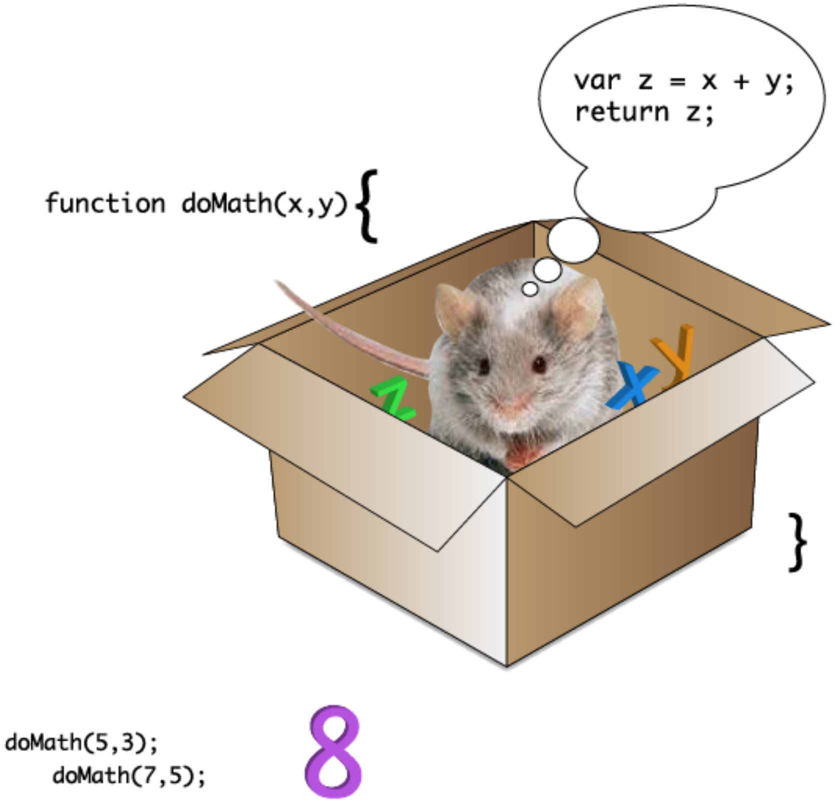
CALLING THE FUNCTION AGAIN With Different Parameters



CALLING THE FUNCTION AGAIN With Different Parameters



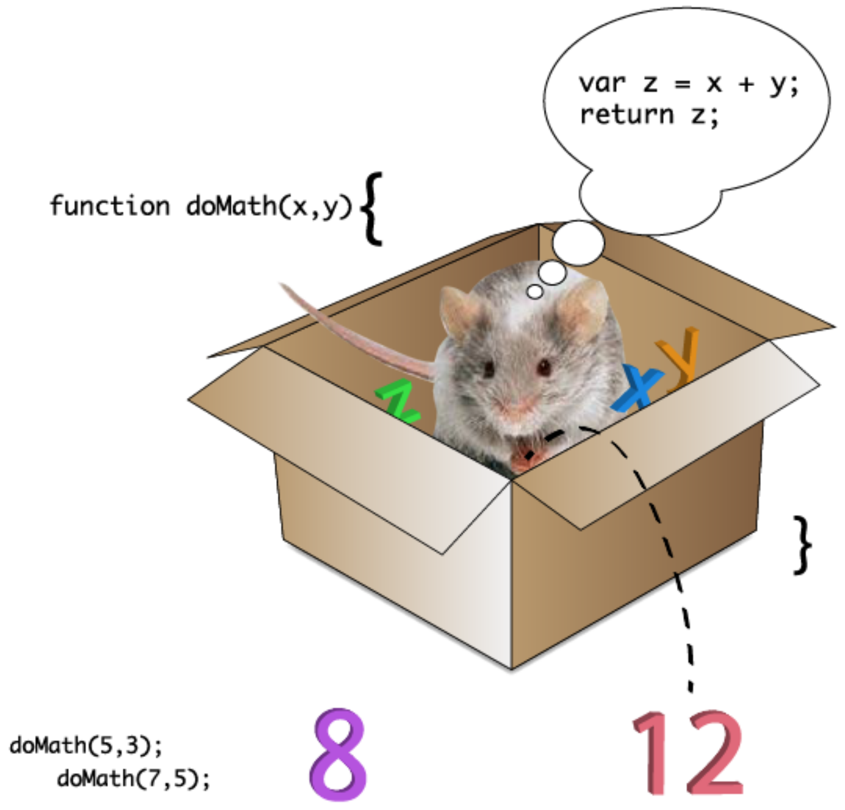
FUNCTION RUNNING A SECOND TIME



```
function doMath(x,y){
    var z = x + y;
    return z;
}
```

```
doMath(5,3);
    doMath(7,5);
```

Returning a NEW Value

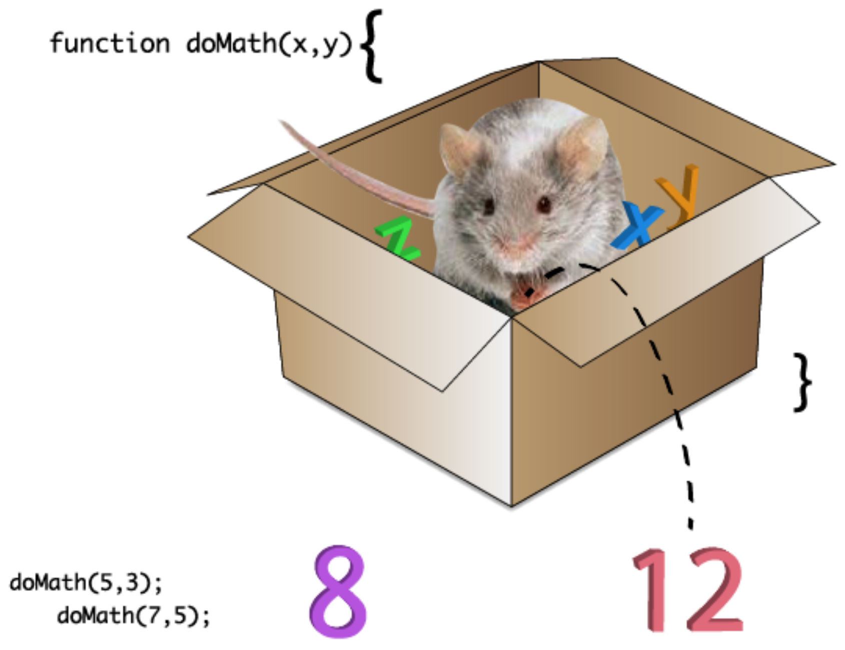


```
var z = x + y;
return z;
```

```
function doMath(x,y){
}
```

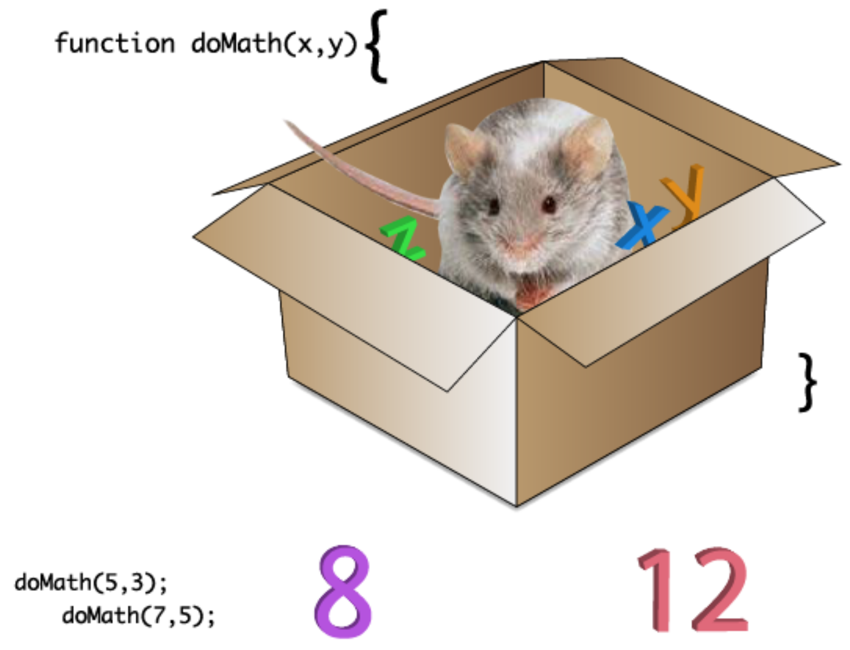
```
doMath(5,3);
    doMath(7,5);
```

Returning a NEW Value



```
doMath(5,3);
    doMath(7,5);
```

Returning a NEW Value



```
function doMath(x,y){
}
doMath(5,3);
    doMath(7,5);
```

Javascript functions

//functions are very useful for setting aside a set of actions that we would like to run multiple times and at specific moments. They will only run when we ask them.

```
function takeOutTrash() {
    var garbage = 'empty';
    console.log(garbage);
    //Things in here will not run until we call the function by name.
}
```

//notice there is no need for a semi-colon after a functions curly braces.

Javascript functions

```
//call the function when we want to run the actions inside it.
takeOutTrash();

//functions can also be stored inside of variables.
var x = function() {
    console.log('Hello there!');
}

//call the function
x();

//sometimes functions do not have names, such as the one inside of the
var x in the example above, these are called an anonymous functions.
```

Javascript functions

```
//You can pass values into a function using parameters
function doMath(x, y) {
    var z = x + y;
    console.log(z);
}

doMath(4, 5); //This will print 9

//parameters can accept boolean, numbers, or strings, however be
careful. You can get unexpected results if you pass strings into the
doMath function...
```

Javascript functions

```
//Values created inside the function can't be used unless we return the
value outside the function.

function doMoreMath(x, y) {
    var z = x / y;
}

doMoreMath(4, 2);
//console.log(z); //see console error then comment this out.
```

Javascript functions

```
//now lets return the value of z inside the function.

function doEvenMoreMath(x, y) {
    var z = x * y;
    return z;
}

console.log(doEvenMoreMath(5,3)); //prints 15.
```

Scope

SCOPE

- Variables have limited scope within functions.

- A variable declared **outside** a given function has **global** scope---meaning that variable can be seen and used outside of as well as inside of any functions in the program.
- A variable declared **inside** a function has **local** scope---meaning that variable can only be seen, operated upon within the scope of that function.

SCOPE examples

//The variables declared inside of functions are called local varibales because they only have 'local' scope meaning they can't be seen outside the function unless we return them as in the example above. One good thing about this is that it protects variables so we can't ovewrite them accidentally outside the function.

```
function milkTheCow() {
     var bessyCow = 'milked'; //local var
     return bessyCow;
}

console.log(bessyCow); //see error
```

Brain Teaser

```
//try to figure out this functions result:

var tiger = 2,
    lion = 3;

function countAnimals(giraffe) {
    tiger = tiger - 1;
    var total = lion + tiger + giraffe;
    return total;
}

countAnimals(3);
```