

Functions



Dr. Stephen Blythe
CSC I 0022, Fall 2019

The function Concept

- You have already called existing functions in JavaScript:
 - alert, prompt, document.write, Math.sqrt, ...
- Core idea: functions do a task
 - that is, they are made up of a sequence of JavaScript statements
- You do not need to know how the function works to use it
 - you just “call” the function, and it performs its task
- ... unless you are writing your own function ...
 - yes, that’s right - *you can create your own JavaScript functions!*

The `function` Statement ...

- `function` is used to create(define) a new function in JavaScript

- General form:

```
function <name> (<parameter_names>)  
{  
    <body>  
}
```

- `<name>` should be replaced with an identifier that is to be the name of the new function
- `<parameter_names>` is a (sometimes empty) list of input names
 - discussed later
- `<body>` is the code that performs the task for this function
 - *any* code you want: assignment, if, while, calls to other functions...

function Example

```
function printLongMessage()  
  
{  // when called, this prints out 3 lines of output.  
  
    document.write("An important message follows:<br>");  
  
    document.write(" You are reading this because I printed it!<br>");  
  
    document.write(" And because you cannot pass this class without");  
  
    document.write(" knowing this material ...<br>");  
  
}
```

- Once defined, you can then “call” this function from elsewhere:

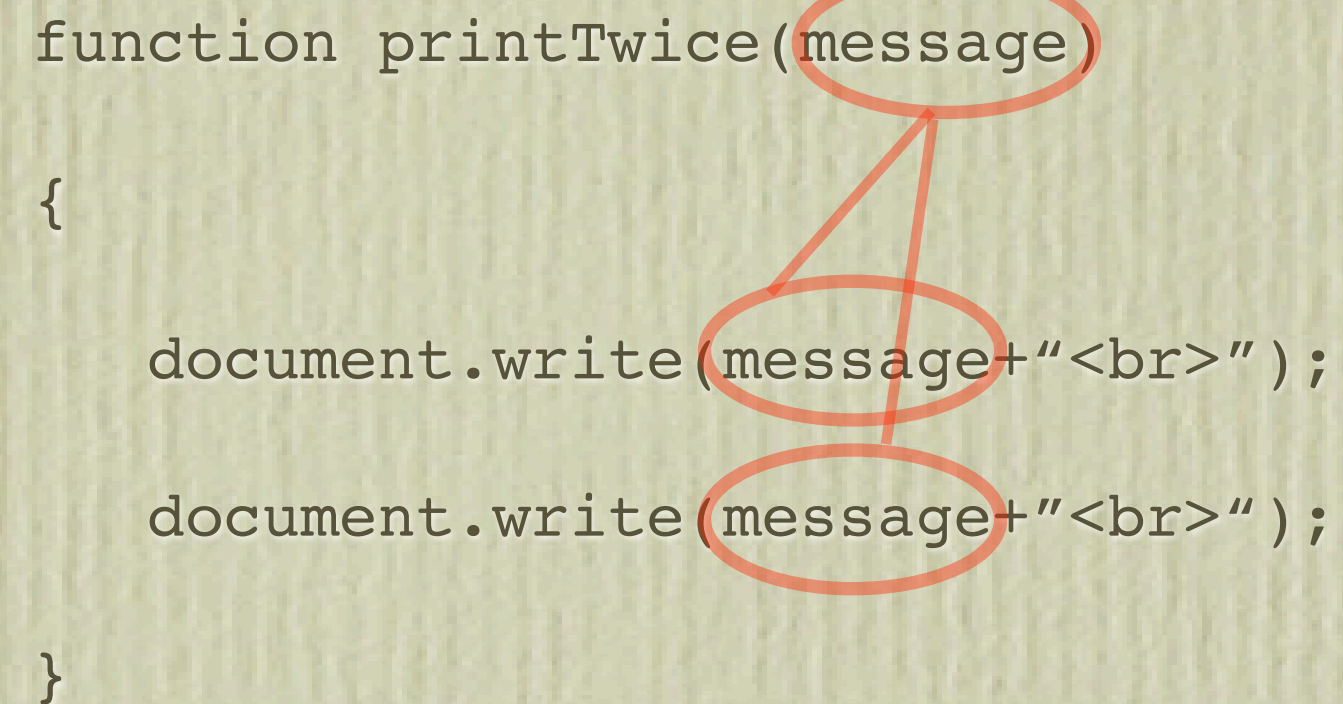
```
printLongMessage();  
  
printLongMessage();  
  
printLongMessage();
```

- Note that defining the function does NOT cause its code to run!
- Each call to a function causes the function’s body to be executed
 - Thus, the above prints 9 lines out to the document body. Why?

Another function Example

- You can specify an input to a function definition:

```
function printTwice(message)
{
    document.write(message+"<br>");
    document.write(message+"<br>");
}
```

A diagram with red circles and lines. A red circle is around the parameter 'message' in the function definition 'function printTwice(message)'. Two red circles are around the 'message' parameter in the two function calls 'document.write(message+"
");'. Red lines connect the 'message' in the definition to the 'message' in each of the two calls.

```
printTwice("Hello World");

printTwice("GoodBye!");

var str = prompt("What do you want to see?");

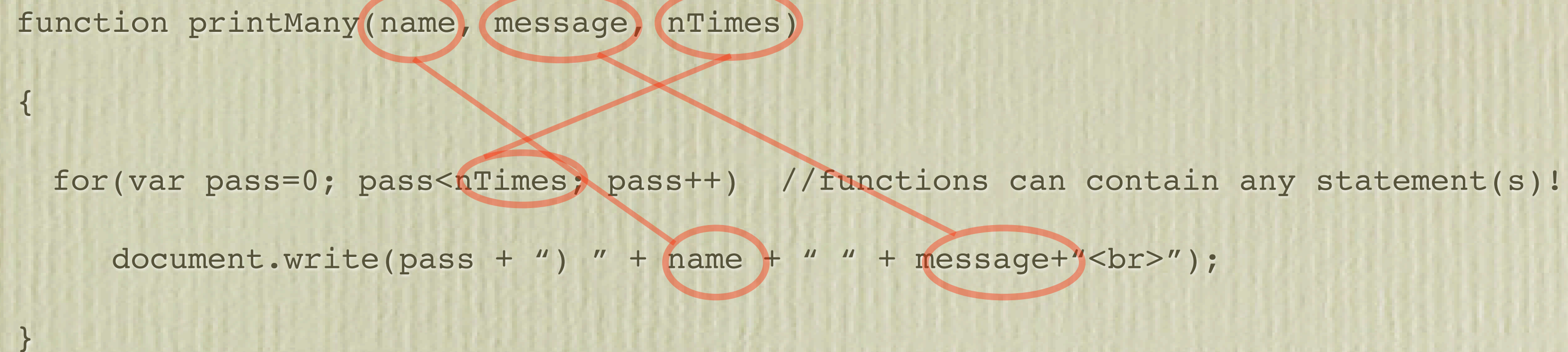
printTwice(str);
```

- Note that the parameter name/value in a call does NOT have to match the input name in the function definition!
- note the difference here: str vs. message

Another function Example

- You can specify several inputs to a function definition:

```
function printMany(name, message, nTimes)
{
    for(var pass=0; pass<nTimes; pass++) //functions can contain any statement(s)!
        document.write(pass + " " + name + " " + message + "<br>");
}
```



```
printMany("Dr. Blythe", "is totally evil", 25); // prints 25 lines

var msg = prompt("What do you want to see?");

var howMany = Number(prompt("How many times do you want to see '" + msg + "'?"));

printMany("Jimmy Fallon", msg, howMany );
```

- Note that anything goes in a function body
 - should not define another function inside of a function definition

Parameter Matching

```
function printThem(x, y)

{

    document.write("x=" + x + ", y=" + y + "<br>");

}
```

```
printThem(12.2, -7.1); // prints    x=12.2, y=-7.1

var x=1, y=22, a=3333, b=444;

printThem(x, y);        // prints    x=1, y=22

printThem(y, x);        // prints    x=22, y=1

printThem(b, a);        // prints    x=444, y=3333
```

- Note that the **ORDER** that the inputs are passed is what matters
- The names really don't tell us anything about how to align parameters!

Return Value

- A function can (optionally) return a value, using the `return` statement:

`return <value>;`

- `<value>` is replaced with the value to return, for example:

```
function validatedPrompt(m)

{

    var result = Number(prompt(m));

    while(result<=0)

        result = Number(prompt("Sorry, that was not positive ... " + m));

    return result;

}
```

- This is called as any other function might be:

```
var goodValue = validatedPrompt("Give me the first positive value");

var anotherGoodValue = validatedPrompt("Give me the second positive value");

document.write("The sum of your two positives is :"+ (goodValue+anotherGoodValue));
```


Variable “Scope”

- “Scope” refers to where a variable “exists”.
- Consider the following (simple) function:

```
function verySimple()  
  
{  
  
    var x=20;  
  
    document.write("inside the function, x="+x+"<br>");  
  
}
```

- The following sequence gives an “undefined variable” error for x:

```
verySimple();  
  
document.write("outside the function, x="+x+"<br>");
```

- ... which makes sense from the caller’s perspective, as x never was declared
- This is an example of function scope :
 - *variables declared inside a function only exist inside of the function!*

Variable “Scope” ...

- Consider the following (simple) function again:

```
function verySimple()  
  
{  
  
    var x=20;  
  
    document.write("inside the function, x="+x+"<br>");  
  
}
```

- This time, declare (and initialize) a variable x first:

```
var x=1;  
  
verySimple();  
  
document.write("outside the function, x="+x+"<br>");
```

- No error this time, the resulting output is:

inside the function, x=20

outside the function, x=1

Variable “Scope” ...

- Consider the following (similar) function:

```
function stillSimple()  
  
{  
  
    x=20; // no var here, so we must be referring to a “global” version of x  
  
    document.write(“inside the function, x="+x+"<br>”);  
  
}
```

- The absence of **var** above means x is not a “local” variable above ... so:

```
var x=1;  
  
stillSimple();  
  
document.write(“outside the function, x="+x+"<br>”);
```

- gives output of:

```
inside the function, x=20  
outside the function, x=20
```