

Functions



Dr. Stephen Blythe
CSC I 0022, Fall 2020

The function Concept

- You have already called existing functions in JavaScript:
 - alert, prompt, console.log, Math.atan
- 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 for you
- ... 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, calls to other functions...

function Example

```
function printLongMessage()  
  
{  // when called, this prints out 4 lines of output.  
  
    console.log("An important message follows:");  
  
    console.log(" You are reading this because I logged it!");  
  
    console.log(" And because you cannot pass this class without");  
  
    console.log(" knowing this material ...");  
  
}
```

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

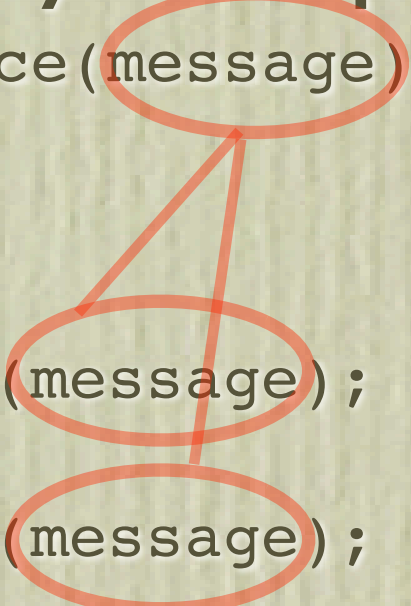
```
printLongMessage();  
  
printLongMessage();  
  
printLongMessage();
```

- 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 **12** lines out to the console. Why?

Another function Example

- You can specify an input to a function definition:

```
function printTwice(message)
{
    console.log(message);
    console.log(message);
}
```

A diagram consisting of three red ovals. The top oval is around the parameter 'message' in the function definition. Two lines extend from the bottom of this oval to two other ovals below it. The first oval is around the 'message' argument in the first 'console.log' call, and the second oval is around the 'message' argument in the second 'console.log' call.

```
printTwice("Hello World");

printTwice("GoodBye!");

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

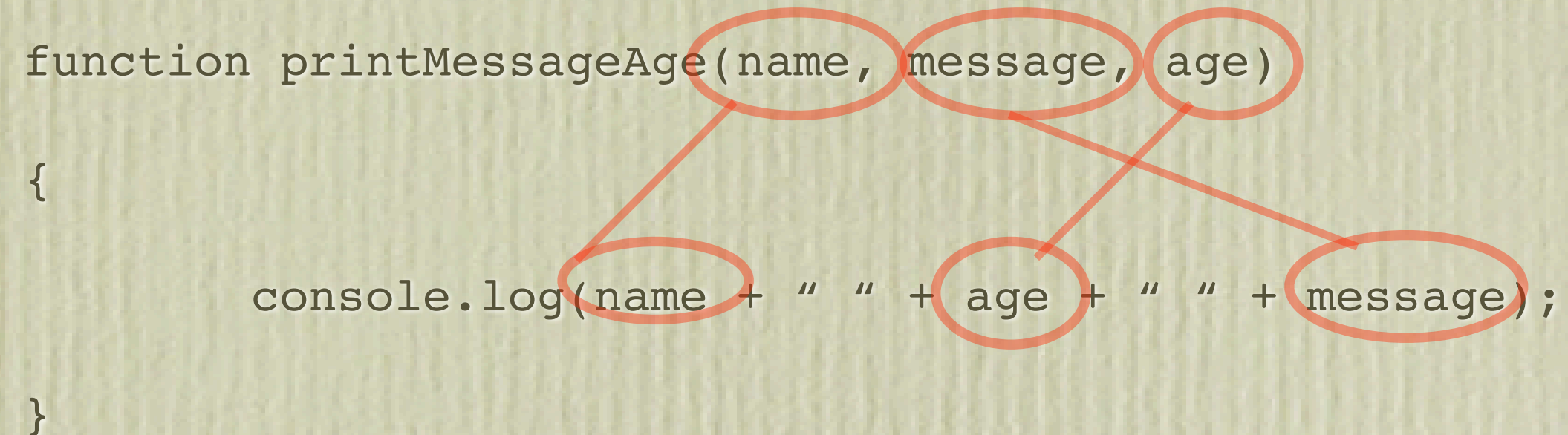
printTwice(str);
```

- 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 printMessageAge(name, message, age)
{
    console.log(name + " " + age + " " + message);
}
```



A diagram illustrating argument passing. Red circles highlight the arguments 'name', 'message', and 'age' in the function definition 'function printMessageAge(name, message, age)'. Another set of red circles highlights the arguments 'name', 'age', and 'message' in the function call 'console.log(name + " " + age + " " + message);'. Red lines connect the circles in the function call to the corresponding circles in the function definition: 'name' to 'name', 'age' to 'age', and 'message' to 'message'.

```
printMessageAge("Dr. Blythe", "years old in his dreams", 25);

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

var howMany = Number(prompt("How old is Jimmy Fallon ?") );

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)

{

    console.log("x=" + x + ", y=" + y );

}
```

```
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 numberPrompt(m)
{
    var result = Number(prompt(m));
    return result;
}
```

- This is called as any other function might be ...
 - ... but now we can store the value returned

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

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

console.log("The sum of your two numbers is :" + (goodValue+anotherGoodValue));
```


Variable “Scope”

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

```
function verySimple()  
  
{  
  
    var x=20;  
  
    console.log("inside the function, x="+x);  
  
}
```

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

```
verySimple();  
  
console.log("outside the function, x="+x);
```

- ... 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;  
  
    console.log("inside the function, x="+x);  
  
}
```

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

```
var x=1;  
  
verySimple();  
  
console.log("outside the function, x="+x);
```

- 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  
  
    console.log(“inside the function, x="+x);  
  
}
```

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

```
var x=1;  
  
stillSimple();  
  
console.log(“outside the function, x="+x);
```

- gives output of:
 inside the function, x=20
 outside the function, x=20

Alternate Function Definition

- Consider the following variable declaration:

```
var oddlyWritten = function()  
  
{  
  
    x=20; // no var here, so we must be referring to a "global" version of x  
  
    console.log("inside the function, x="+x);  
  
}
```

- it's an alternative way to declare a function.

```
var x=1;  
  
oddlyWritten();  
  
console.log("outside the function, x="+x);
```

- It's still called the same way, though!
- And does exactly the same thing!