Programming 1

Javascript
Lecture #2: Functions and variable scope

Functions – the story so far: a recap

- A function is
  - like a mini program within a program
  - a way of adding your own "commands" to a language
  - a way of defining a set of instructions up front when you're likely to need to do them several times within a larger program
- We use "functions" in everyday life
  - Q: “How do I make a sandwich?"
    - The answer might include statements like “slice bread, spread butter on slices”
    - What does *slice* mean? What does *spread* mean?
- ‘slice’ and ‘step’ are functions
  - We explain how we do these things before we give the "main“ set of instructions
  - We can then reference them in the instructions

Functions – the story so far: a recap

- Remember the `turnRight` function for Carol in Banana

```
function turnRight
   for count = 1 to 3
       call turnLeft
   endfor
end function
```

- Carol didn’t know how to turn right as part of her standard repetoire
  - So we gave the steps for turning right up front – we *defined* them as a function
  - We can then *call* this function whenever we need to turn right

Functions and parameters in Javascript

- To define a function, we specify brackets after the function name:

```
function printMessage()
{
   console.log("This is my message");
}
```

…then later in the program, we call it with…

```
printMessage();
```

Functions and parameters in Javascript

- When we define a function, we specify brackets after the function name:

```
function printMessage()
{
   console.log("This is my message");
}
```

brackets

…then later in the program, we call it with…

```
printMessage();
```

Functions and parameters in Javascript

- This would be the same as the Banana code

```
function printMessage
   display "This is my message"
endfunction
```

…then later in the program, we call it with…

```
call printmessage
```

Functions and parameters

- Parameters allow us to "communicate" with the function from the part of the code that calls the function
- Consider the sandwich example
- We might give an instruction like "Slice the bread"
- If *slice* was a function, then *the bread* is a parameter
- The parameter allows us to supply a value (or object) to the function
- The function can then use the parameter as part of what it does

Functions and parameters in Javascript

- We put *parameters* between the brackets of our *function declaration*

```
function printMessage(numberOfTimes)
{
    for (var count = 0; count < numberOfTimes; count++)
    {
        console.log("This is my message");
    }
}
```

…then later in the program, we call the function with something like..

```
printMessage(10);
```

Functions and parameters in Javascript

- The brackets are for any *parameters* the function might need

```
function printMessage(numberOfTimes)
{
    for (var count = 0; count < numberOfTimes; count++)
    {
        console.log("This is my message");
    }
}
```

...then later in the program, we call it with...

```
printMessage(10);
```

Functions and parameters in Javascript

- Multiple parameters can be specified

```
function betterPrintMessage(numberOfTimes,message)
{
    for (var count = 0; count < numberOfTimes; count++)
    {
        console.log(message);
    }
}
```

…then later in the program, we call it with…

```
betterPrintMessage(10,"Hello there");
```

Functions and parameters

- Multiple parameters can be specified

```
function betterPrintMessage(numberOfTimes, message)
{
    for (var count = 0; count < numberOfTimes; count++)
    {
        console.log(message);
    }
}
```

…then later in the program, we call it with…

```
betterPrintMessage(10, "Hello there");
```

"undefined"

- Unlike languages like Java, Javascript doesn't enforce function calls having the same number of parameters as the function declaration
- If you call a function and one of the parameters is missing, the special value *undefined* is placed into the parameter variable

```
function myTestFunction(myParam)
{
     console.log(myParam);
}
// prints "Good morning"
myTestFunction("Good morning");
// prints "undefined"
myTestFunction();
```

"undefined"

- Unlike languages like Java, Javascript doesn't enforce function calls having the same number of parameters as the function declaration
- If you call a function and one of the parameters is missing, the special value *undefined* is placed into the parameter variable

```
function myTestFunction(myParam)
{
     console.log(myParam);
}
// prints "Good morning"
myTestFunction("Good morning");
// prints "undefined"
myTestFunction();
```

parameter in function call is not defined, so the special value *undefined* gets passed to the function parameter

"undefined"

- You can detect for *undefined*

```
function myTestFunction(myParam)
{
     if (myParam != undefined)
     {
         console.log(myParam);
     }
     else
     {
         console.log("You forgot the parameter!");
     }
}
// prints "Good morning"
myTestFunction("Good morning");
// prints "You forgot the parameter!"
myTestFunction();
```

"undefined"

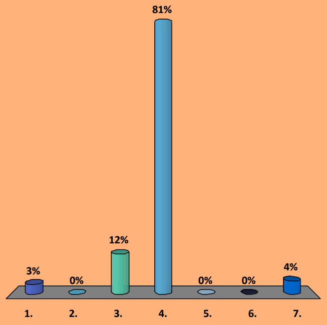
- or even

```
function myTestFunction(myParam)
{
    if (myParam)
    {
        console.log(myParam);
    }
    else
    {
        console.log("You forgot the parameter!");
    }
}
// prints "Good morning"
myTestFunction("Good morning");
// prints "You forgot the parameter!"
myTestFunction(false);
```

What would the program below do?

```
function countPrint(howMany)
{
    for (var count = 0; count < howMany; count++)
    {
        console.log(count);
    }
}
countPrint(9);
countPrint(3);
```

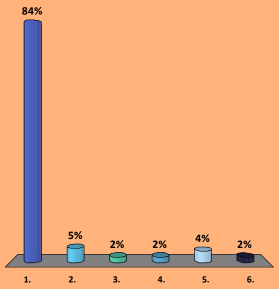
1. It would print the number 9 followed by the number 3
2. It would print the number 3 followed by the number 9
3. It would count from 0 to 9 then count from 0 to 3
4. It would count from 0 to 8 then count from 0 to 2
5. It would count from 0 to 12
6. It would count from 0 to 11
7. There would be an error



What would the program below display?

```
function chooseWord(index)
{
    var words = ["apple","orange","banana","pear"];
    console.log(words[index]);
}

chooseWord(1);
chooseWord(3);
chooseWord(5);
chooseWord(0);
chooseWord(-1);
```

1. orange/pear/undefined/apple/undefined
2. apple/banana/undefined/undefined/undefined
3. orange/pear/apple
4. apple/banana
5. 1/3/5/0/-1
6. There would be an error



Return values

- Functions can also *return* a value
- **THIS IS DIFFERENT TO PRINTING THE RESULT!**
- **THIS IS NOT THE SAME AS PRINTING THE RESULT!**
- **THIS IS DIFFERENT TO PRINTING THE RESULT!**
- (yes, I am laboring the point here…!)

Return values

- Consider the sandwich example
- Consider the previous excerpt, *slice the bread*
- What do we get back after we have sliced the bread?

Return values

- Consider the sandwich example
- Consider the previous excerpt, *slice the bread*
- What do we get back after we have sliced the bread?
  - So, the slices of bread that are the end result are like the return value of the function *slice*
    - We “called a function” on *bread* – i.e. we sliced it
    - What we got back was the slices – the result of running the function

Return values in Javascript

- We could write a function to **return** a specific word from an array:

```
function chooseWord(index)
{
    var words = ["apple","orange","banana","pear"];
    return(words[index]);
}
chooseWord(1);
var myWord = chooseWord(2);
console.log("My word was "+myWord);
```

Return values

- We could write a function to **return** a specific word from an array:

```
function chooseWord(index)
{
    var words = ["apple","orange","banana","pear"];
    return(word[index]);
}

~~chooseWord(1);~~ // does nothing (visible, at least)
var myWord = chooseWord(2);
console.log("My word was "+myWord);
```

Return values

- We could write a function to **return** a specific word from an array:

```
function chooseWord(index)
{
    var words = ["apple","orange","banana","pear"];
    return(word[index]);
}

~~chooseWord(1);~~ // does nothing (visible, at least)
~~var myWord = chooseWord(2);~~
console.log("My word was "+chooseWord(2));
```

Parameters AND return values

- Consider our sandwich example again
  - …”slice the bread”…
  - If *bread* is a parameter to a function *slice*
  - The resulting *slices of bread* are the what the function returns
  - Thus the *slice* function has both a *parameter* and a *return value*

Return values

- Functions can use parameters and return values to "communicate" with each other

```
function getRandomNumber(max)
{
   var num = Math.floor(Math.random()*(max+1));
   return num;
}

function getRandomWord()
{
   var words =["apple","orange","banana","pear","kiwi","peach","fig"];
   var randomNo = getRandomNumber(words.length-1);
   return words[randomNo];
}

for (var count = 0; count< 5; count++)
{
   console.log(getRandomWord());
}
```

Return values

- Functions can use parameters and return values to "communicate" with each other

```
function getRandomNumber(max)
{
   var num = Math.floor(Math.random()*(max+1));
   return num;
}

function getRandomWord()
{
   var words =["apple","orange","banana","pear","kiwi","peach","fig"];
   var randomNo = getRandomNumber(words.length-1);
   return words[randomNo];
}

for (var count = 0; count< 5; count++)
{
   console.log(getRandomWord());
}
```

first line that runs

Return values

- Functions can use parameters and return values to "communicate" with each other

```
function getRandomNumber(max)
{
   var num = Math.floor(Math.random()*(max+1));
   return num;
}

function getRandomWord()
{
   var words =["apple","orange","banana","pear","kiwi","peach","fig"];
   var randomNo = getRandomNumber(words.length-1);
   return words[randomNo];
}

for (var count = 0; count< 5; count++)
{
   console.log(getRandomWord());
}
```

← repeats 5 times

Return values

- Functions can use parameters and return values to "communicate" with each other

```
function getRandomNumber(max)
{
   var num = Math.floor(Math.random()*(max+1));
   return num;
}

function getRandomWord()
{
   var words =["apple","orange","banana","pear","kiwi","peach","fig"];
   var randomNo = getRandomNumber(words.length-1);
   return words[randomNo];
}

for (var count = 0; count< 5; count++)
{
   console.log(getRandomWord());
}
```

this…
…calls this function

Return values

- Functions can use parameters and return values to "communicate" with each other

```
function getRandomNumber(max)
{
   var num = Math.floor(Math.random()*(max+1));
   return num;
}

function getRandomWord()
{
   var words =["apple","orange","banana","pear","kiwi","peach","fig"];
   var randomNo = getRandomNumber(words.length-1);
   return words[randomNo];
}

for (var count = 0; count< 5; count++)
{
   console.log(getRandomWord());
}
```

this…

…calls this function

Return values

- Functions can use parameters and return values to "communicate" with each other

```
function getRandomNumber(max)
{
   var num = Math.floor(Math.random()*(max+1));
   return num;
}

function getRandomWord()
{
   var words =["apple","orange","banana","pear","kiwi","peach","fig"];
   var randomNo = getRandomNumber(words.length-1);
   return words[randomNo];
}

for (var count = 0; count< 5; count++)
{
   console.log(getRandomWord());
}
```

this…
Is the result here

Return values

- Functions can use parameters and return values to "communicate" with each other

```
function getRandomNumber(max)
{
   var num = Math.floor(Math.random()*(max+1));
   return num;
}

function getRandomWord()
{
   var words =["apple","orange","banana","pear","kiwi","peach","fig"];
   var randomNo = getRandomNumber(words.length-1);
   return words[randomNo];
}

for (var count = 0; count< 5; count  )
{
   console.log(getRandomWord());
}
```

this…
Is the result of this

Variable scope

- Variables that are declared within functions are only visible within the function…

```
function chooseWord(index)
{
    var words = ["apple","orange","banana","pear"];
    var word = words[index];
}
chooseWord(2);
console.log(word);
```

Variable scope

- Variables that are declared within functions are only visible within the function…

```
function chooseWord(index)
{
    var words = ["apple","orange","banana","pear"];
    var word = words[index];
}
chooseWord(2);
console.log(word);
```

Will error here – the variable word cannot be seen outside the function it was declared in

Variable scope

- Variables that are declared within functions are only visible within the function…

```
var word; // declare the variable outside function
function chooseWord(index)
{
    var words = ["apple","orange","banana","pear"];
    word = words[index]; // note: no “var” keyword!
}
chooseWord(2);
console.log(word);
```

✓

Works fine – word is declared outside of the function, so is visible both inside and outside the function.

This is called a GLOBAL variable

Variable scope

- Global variables are frowned upon from a style perspective
- A return value would be a better way to do this…

```
function chooseWord(index)
{
    var words = ["apple","orange","banana","pear"];
    word = words[index]; // note: no “var” keyword!
    return word;
}
console.log(chooseWord(2));
```

✓

Works fine – word is never accessed directly outside the function – instead, we send it as our return value

“REMEMBER THE GRAMMAR”

Variable scope

- Global variables are frowned upon from a style perspective
- A return value would be a better way to do this…

```
function chooseWord(index)
{
    var words = ["apple","orange","banana","pear"];
    word = words[index]; // note: no "var" keyword!
    return word ;
}
console.log( chooseWord(2) );
```

✓

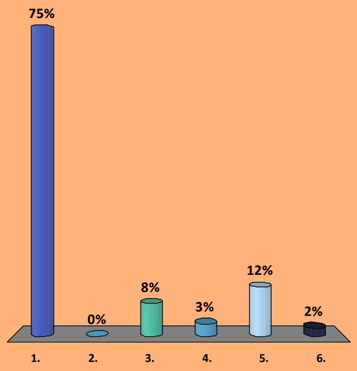
Works fine – word is never accessed directly outside the function – instead, we send it as our return value

"REMEMBER THE GRAMMAR"

What would the program below display?

```
function foo(bar)
{
   var woot = 10;
   var woot = bar * woot;
   return woot;
}

var woot = 27;
var bar = foo(woot);
console.log(bar);
```

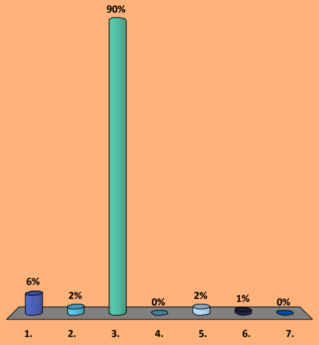
1. 270
2. 100
3. 27
4. 10
5. undefined
6. There would be an error



OK, what would this program display?

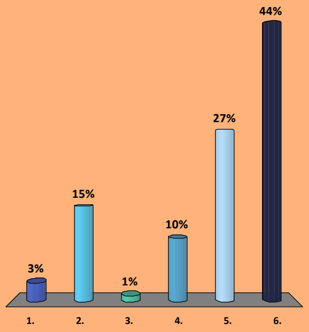
```
function foo(bar)
{
    var woot = kerplunk(bar)
    var woot = bar * woot;
    return woot;
}

function kerplunk(foo)`
{
    return foo+3;
}

var woot = 7;
var bar = foo(woot);
console.log(bar);
```

1. 10
2. 21
3. 70
4. 7
5. 3
6. Nothing
7. There would be an error



And this one? What would it display?

```
var woot = 12;

function foo(bar)
{
   var kerplunk = 24;
   var woot = bar * woot;
}

foo(5);
console.log(woot);
console.log(kerplunk);
```



1. 12 followed by 24
2. 60 followed by 24
3. 5, 12 and then 24
4. 60 followed by an error
5. 24 followed by an error
6. 12 followed by an error

Style choices for Javascript functions

- Functions can appear in any order throughout your program

```
function funcOne()
{
   console.log("This is the first function");
}

function funcTwo()
{
   console.log("This is the second function");
}

funcOne();
funcTwo();
```

Style choices for Javascript functions

- Functions can appear in any order throughout your program

```
funcOne();
funcTwo();

function funcOne()
{
   console.log("This is the first function");
}

function funcTwo()
{
   console.log("This is the second function");
}
```

Style choices for Javascript functions

- Functions can appear in any order throughout your program

```
function funcOne()
{
   console.log(“This is the first function”);
}

funcOne();
funcTwo();

function funcTwo()
{
   console.log(“This is the second function”);
}
```

Style choices for Javascript functions

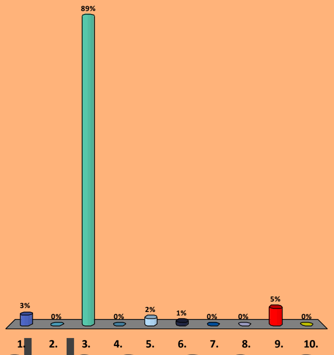
- Functions can appear in any order throughout your program

```
funcOne();

function funcOne()
{
   console.log("This is the first function");
}

funcTwo();

function funcTwo()
{
   console.log("This is the second function");
}
```

What would this program display?

```
function one()
{
   console.log("three");
}

function two()
{
   console.log("one");
}

function three()
{
   console.log("two");
}

function four()
{
   three();
   two()
   one();
}

four();
```

1. three/two/one
2. one/two/three
3. two/one/three
4. three/two/one/four
5. just four
6. just three
7. just two
8. just one
9. Nothing
10. There would be an error



Style choices for Javascript functions

- Functions can appear in any order throughout your program
- As a best practice style choice, use the following approach:
  - Global variables
  - Functions
  - Then the main body of your code

Summary

- A function is like
  - A mini program within a program
  - A way of extending the capabilities of a programming language by providing a means of describing how to do new “commands”
  - A way of avoiding repetition within a program
- Function can accept parameters
  - A parameter is a value that is sent to the function from the calling code
    - “slice bread with knife” – bread and knife are parameters
- Functions can have return values
  - A return value goes back to the code that calls a function
  - THIS IS NOT THE SAME AS PRINTING SOMETHING TO THE SCREEN
  - The calling code can then use the return value
    - “slice bread with knife” – the return value would be the bread slices that resulted

Summary

- In Javascript, variable scope means that variables can only be "seen" from within the function in which they are declared
- Remember that when you use the `var` keyword, this declares a variable
  - If you want to use a variable throughout a program, declare it as a global variable – i.e. not within a specific function
- Functions in Javascript can appear at any point within a program
  - But try to keep functions declarations at the top of your program – this will help avoid confusion
  - Don't fall into the trap of having little loose bits of main body code floating around function declarations…