JAVASCRIPT FUNCTIONS

MICRO-REVIEW FROM PROGRAMMING LESSON

- What is a program?
- What are variables for?

WHAT IS A FUNCTION?

- In it's most basic form, ***a function is a reusable statement.***
- This avoids the need to rewrite the same statement over and over
- Functions enable the software developer to segment large, unwieldy applications into smaller, more manageable pieces.
- critical component of programming, functions address a key tenet of engineering: Don't Repeat Yourself, or DRY.

DRY

- We should always be striving to be creating DRY code.
- Our goal is to create programs with as little code as possible, while maintaining complete clarity.
- If you do notice that you are doing some piece of logic over and over again, you should be thinking “how do I make that into a function?”

HOW TO DECLARE A FUNCTION: FUNCTION EXPRESSION

- One way to create a function is with something we call a *function declaration*
- *A function declaration is where we declare a function to a variable name.*
- Like this:

```
var speak = function(words) {
  console.log(words)
};
```

HOW TO DECLARE A FUNCTION: FUNCTION EXPRESSION

```
var speak = function(words) {
  console.log(words);
};
```

- This looks like a normal variable, has the same “var” declaration, and the “=“ assignment
- The only difference is that we are using the keyword “function” to announce that instead of declaring the variable as a value, we are declaring it as a function.

HOW TO DECLARE A FUNCTION: FUNCTION EXPRESSION

```
var speak = function(words) {
  console.log(words);
};
```

- After the keyword “function” there are some clappers with a parameter inside (more on that in a few minutes)
- After the clappers, we put a curly bracket that contains the code that we want to run on the next lines.
- ***That code is indented inside by one tab or two spaces.***

THE ONE PROBLEM WITH FUNCTION EXPRESSIONS

- Function expression suffer from the same problem that variable declarations have.
- You can only use the function AFTER you declare it.

```
speak('hello, world!');

var speak = function (words) {
  console.log(words);
};

// results in an error
```

THE ONE PROBLEM WITH FUNCTION EXPRESSIONS

- The only way to use it is if you call the function after it is declared
- Not a huge deal, but something to be careful with.

```
var speak = function(words) {
  console.log(words)
};

speak('hello, world!’);
// returns ‘hello, world!’
```

HOW TO DECLARE A FUNCTION: FUNCTION DECLARATIONS

- Another way of declaring a function is with function declarations:

```
function speak(words) {
  console.log(words);
}
```

- In a function express you begin with the keyword function

HOW TO DECLARE A FUNCTION: FUNCTION DECLARATIONS

- Take a good hard look, it takes some getting used to:

```
function speak(words) {
  console.log(words);
}
```

- Note that with a function declaration, there is no equal sign assigning the value. You just begin with the keyword “function” and javascript expects the next value to be the name for it

HOW TO DECLARE A FUNCTION: FUNCTION DECLARATIONS

- Take a good hard look, it takes some getting used to:

```
function speak(words) {
  console.log(words);
}
```

- A function declaration always has the following:
- A name
- An optional list of parameters (i.e., the names of arguments to be "passed" into the function, or information the function will use); this is defined by the parenthesis before the opening curly brace
- Statements inside the function (the code executed every time the function is called)

FUNCTION DECLARATIONS: HOISTING

- One of the wonderful things about a function declaration is that javascript will read any function declarations into memory first. No matter what.
- This means that you don't necessarily need to declare the function before using it!

```
speak('hello, world!');
//it works!


function speak(words) {
  console.log(words)
}
```

FUNCTION DECLARATIONS: HOISTING

- This characteristic of function declarations is called “hoisting”
- This makes things a bit easier to work with. For this reason, function declarations are widely preferred over function expressions.

```
speak('hello, world!’);
//it works!


function speak(words) {
  console.log(words)
}
```

CALLING A FUNCTION – "INVOKING"

- Calling, or invoking, a function executes the code defined inside this function.
- But defining and calling a function is different. A function will not be called when it's defined.
- You call a function by using parenthesis after the function's name () (or as I call, "clappers" 👏👏)

```
function hello() {
  console.log("hello there!");
}

hello();
```

FUNCTION EFFECTIVENESS

- Now a function like that hello() function isn't very useful on it's own.
- Sure now we can do this:

```
function hello() {
  console.log("hello there!");
}

hello();
hello();
hello();
```

- Why is that a bad idea?

FUNCTION EFFECTIVENESS

- Now a function like that hello() function isn't very useful on it's own.
- Sure now we can do this:

```
function hello() {
  console.log("hello there!");
}

hello();
hello();
hello();
```

- Why is that a bad idea?

FUNCTION EFFECTIVENESS

- It's just not practical! We could get the same thing done with less code like this:

```
console.log("hello there!");
console.log("hello there!");
```

FUNCTION EFFECTIVENESS

- It's just not practical! We could get the same thing done with less code like this:

```
console.log("hello there!”);
console.log("hello there!”);
console.log("hello there!”);
```

- If we were to make functions for everything, we would have a messy, inefficient ugly bit of code.
- ***Functions have one purpose: take an input, perform logic you'd want to do multiple times, output the result.***

FUNCTION PARAMETERS

- So how do we allow different inputs into our functions?
- This is remedied by “parameters”. Some also call them “arguments” interchangeably, and honestly no one really cares. But teeeechnically speaking:
  - Parameters are used to define a function;
    - As in “I created a function with these parameters”
  - Arguments are used to invoke a function.
    - As in “I passed these arguments into the function”
- Cool that’s totally not that important, and to be honest, 99% of developers don’t know the difference. Welcome to the 1% you knowledge-wizards you. Anyways, moving on…

FUNCTION PARAMETERS

- So how do we allow different inputs into our functions?
- This is remedied by “parameters”. Some also call them “arguments” interchangeably, and honestly no one really cares. But teeeechnically speaking:
  - Parameters are used to define a function;
    - As in “I created a function with these parameters”
  - Arguments are used to invoke a function.
    - As in “I passed these arguments into the function”
- Cool that’s totally not that important, and to be honest, 99% of developers don’t know the difference. Welcome to the 1% you knowledge-wizards you. Anyways, moving on…

USING FUNCTION PARAMETERS

- As we were saying “parameters” are how we allow functions to take in a variety of inputs to perform a small program on them.

```
function sayHello(name) {
  console.log('Hello ' + name);
}

sayHello('Mark');
=> 'Hello Mark'

sayHello('Obama');
=> 'Hello Obama'
```

USING FUNCTION PARAMETERS

```
function sayHello(name) {
  console.log('Hello ' + name);
}
```

- Here the parameter is called “name”.
- The parameter is declared inside of the clappers after the keyword ‘function’ and after the name of the function is declared.
- This parameter can be though of as a variable that is special to that function. This special variable is created during the function declaration! No “var” or anything required.

USING FUNCTION PARAMETERS

```
function sayHello(name) {
  console.log('Hello ' + name);
}

sayHello('Trevor')
=> 'Hello Trevor'
```

- Then when you invoke the function, whatever you put in the clappers will replace where the parameter is found inside of the function.

USING FUNCTION PARAMETERS

- Functions can have any many parameters as you want. You just separate them with commas.
- When you invoke the function, the order of the parameter is respected by the order of the arguments you put into it.

```
function sum(x, y, z) {
  console.log(x + y + z)
}


sum(1, 2, 3);
=> 6
// x = 1, y = 2, z = 3.  In the
same order as the parameters!
```

USING FUNCTION PARAMETERS

- Functions can have any many parameters as you want. You just separate them with commas.
- When you invoke the function, the order of the parameter is respected by the order of the arguments you put into it.

```
function sum(x, y, z) {
  console.log(x + y + z)
}


sum(1, 2, 3);
=> 6
// x = 1, y = 2, z = 3.  In the
same order as the parameters!
```

USING FUNCTION PARAMETERS

- Parameters can take in any data type, strings, numbers, booleans, even arrays and objects.

```
function printAnimals(animalArray) {
  for(var i=0; i < animalArray.length; i++) {
    console.log('I like ' + animalArray[i]);
  }
}

printAnimals(['chicken', 'penguins', 'velociraptor']);
```

THE RETURN STATEMENT

- So far we've been doing console.logs for us to see values in the console. This is and will always be a very useful development tool. Keep using it.
- That said, it doesn't actually MEAN anything. It's just printing information for us as we develop.
- Sometimes we don't want to just print to the console, we might instead want to update a variable, use the output from a function, or event call another function.
- This requires a 'return' statement

THE RETURN STATEMENT

- When we return something, it ends the function's execution and "spits out" what we are returning. We can then store this returned value in another variable...

```
function sum(x, y) {
  return x + y;
}

var z = sum(3, 4);
=> 7
```

- Using console.log(x + y) here would not have allowed us to assign the result to a variable name

THE RETURN STATEMENT

- Note that the return statement will completely stop a function's execution. Any statements following the return statement will not be called:

```
function speak(words) {
  return words;

  // The following statements will not run:
  var x = 1;
  var y = 2;
  console.log(x + y)
}
```

THE RETURN STATEMENT

- By default, JavaScript functions will return an undefined value. To test this, use Node to define and run a function without a return value. A return value "overwrites" this default value.

CODEALONG

- The value of a return statement gets more evident as we get into more realistic code examples.

- Let’s create a new main.js in a new directory in the ‘labs’ folder for this class

LAB

Let's practice