FUNCTIONS

FUNCTIONS

Functions allow us to reuse parts of our code. They are like the verbs of JavaScript.

```
function beAwesome() {
  // go on being awesome
}

beAwesome();
```

FUNCTIONS

A function lets us write some code, not to run now, but to run later...

... as many time as we want.

... or maybe we'll never run it.

THE BREAKDOWN

1. We begin declaring functions by using the `function` keyword.

```
function
```

THE BREAKDOWN

2. Optionally, we can name our functions.

```
function nameOfFunction
```

THE BREAKDOWN

3. The declaration of the function must include a set of parentheses.

```
function nameOfFunction()
```

THE BREAKDOWN

4. The code we want to be able to use in multiple places goes inside a set of curly braces.

```
function nameOfFunction() {
  // things the function does
}
```

THE BREAKDOWN

5. Executing, or *calling*, the function is done by typing the name of the function followed by parentheses.

```
function nameOfFunction() {
  // things the function does
}

nameOfFunction();
```

THE BREAKDOWN

6. Optionally, we can give the function one or more parameters. We can then pass arguments to the function to change the way it works.

```
function nameOfFunction(someParameter) {
  // things the function does, using someParameter
}

nameOfFunction(someArgument);
```

SEEING IS BELIEVING

Let's try some functions in jsbin.com or repl.it.

WHAT IF WE RUN THIS?

```
function sayHi() {
  console.log("Hello World!");
}
```

WHAT IF WE RUN THIS?

It does nothing! If we don't *call* the function, it will never run.

```
function sayHi() {
  console.log("Hello World!");
}
sayHi();
```

What if we want to output "Hello World" twice?

PARAMETER

When a function needs information to complete its task, *parameters*, are defined.

Parameters act as variables in a function.

We give our parameters names that we can reference within the function.

ARGUMENTS

When we call or use our function, we can pass in values or arguments.

These values will be assigned to the parameter names for use within our function.

IT'S LIKE MAD-LIBS

Parameters are the blank spaces.

place: _____, adj: _____, noun: _____

On his way to __place__, Lucas stumbled upon a __adj__ __noun__.

IT'S LIKE MAD-LIBS

Arguments are the values you fill in those spaces.

place: _Nebraska_, adj: _tender_, noun: _lantern_

On his way to _Nebraska_, Lucas stumbled upon a _tender_ _lantern_.

IT'S LIKE MAD-LIBS

You can tell the story again and again but with different words.

place: _Mars_, adj: _lucky_, noun: _caterpillar_

On his way to Mars, Lucas stumbled upon a lucky caterpillar.

ARGUMENTS

```
function sayHi(name) {
  console.log("Hi, " + name + "!");
}

sayHi("Jeseekia");
// >"Hi, Jeseekia!"
```

ARGUMENTS

We can have more than one argument.

We must also consider the data type when passing in values.

ARGUMENTS

```
function sayHi(name,age) {
  console.log("Hi, " + name + "! It's so cool that you're " + age + "!");
}

sayHi("Jeseekia",25);
// > "Hi, Jeseekia! It's so cool that you're 25!"
```

ARGUMENTS

```
function sayHi(name,age) {
  console.log("Hi, " + name + "! It's so cool that you're " + age + "!");
}

sayHi(25, "Jeseekia");
```

What if we switch the order of the arguments?

PARAMETERS

When calling our function, we are not limited to passing in values.

We can also pass in variables with predefined values.

PARAMETERS

```
function sayHi(name,age) {
  console.log("Hi, " + name + "! It's so cool that you're " + age + "!");
}

var myName = "Jeseekia";
var myAge = 25;
sayHi(myName,myAge);
// > "Hi, Jeseekia! It's so cool that you're 25!"
```

THINK ABOUT THIS...

this

MINI CODE CHALLENGE

Write a function named `printWith`.

- It takes two parameters, `phrase1` and `phrase2`.
- Within the function, it outputs the both phrases combined with the word "with". e.g. "a hamburger" and "cheese" -> "a hamburger with cheese".
- Call the function several times with different arguments.

RETURNING VALUES

We can get output from our function by returning values.

This allows us to use output from one function in another.

RETURNING VALUES

The *return* keyword allows a function to return a value to the code that called the function.

PARAMETERS

```
function getArea(length,width) {
  return length * width;
}
var area = getArea(10,6);
console.log(area);
// > 60
```

RETURNING VALUES

Return causes the function to immediately leave the function and return to the statement that called it.

Any code written after a return statement will not be executed, so the return statement should always be last in the function.

PARAMETERS

```
function getArea(length,width) {
  return length * width;
  console.log("You entered Lenth: " + length + ", Width: " + width);
}
getArea(10,6);
// Will return 60, but will not display the console.log statement
```

TRY THIS

Basic JS Exercise 8

Basic JS Exercise 9

Then try

Basic JS Exercise 7

QUESTIONS?

RECAP

You should understand and be able to use:

- Declaring Functions
- Calling Functions
- Passing Arguments to Parameters
- Returning Values back

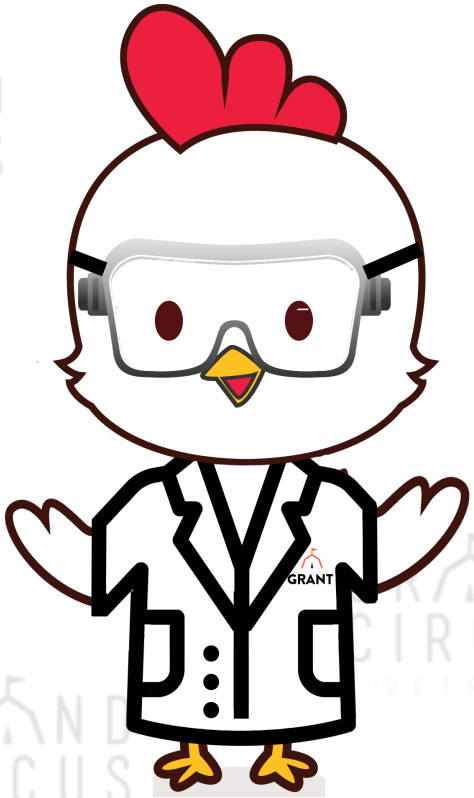
HOMEWORK

From JavaScript & jQuery:

- Chapter 2: 70-73,
- Chapter 3: 100-144

LAB 6

TEMP CONVERTER FUNCTION



INSTRUCTIONS

Rewrite your temperature converter code from Lab 5 into a function.

1. Write a function that converts temperature.
2. Your function should accept two arguments. A temperature and a string indicating what unit to convert to (ex. "C" or "F").
3. Call the function with the following arguments to verify your outputs.

```
convertTemp(212, "C"); // > 100
convertTemp(32, "C");  // > 0
convertTemp(65, "C"); // > 18.333
convertTemp(0, "F"); // > 32
```

PRE-GAME

Solve the problem, then write the code

FIGURE IT OUT

Write a function that assigns a random integer between 1 and 10 to a variable. Prompt the user to input a guess number. If the user input matches with the random number, print a "Good Work" message. Otherwise, display a "Not a Match" message and prompt the user to guess again.

BONUS: If the user's guess is not a match. Give a hint like "Nope, higher" or "Lower" based on the user's guess in relation to the target number.