JavaScript

Functions

Functions

- Functions let us wrap bits of code up into REUSABLE packages. They are one of the building blocks of JS.

Declare a function

- In order to use a function, you must first declare it.

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

Calling a function.

- In order for a function to run, you must call it.
- To call a function, you write the name of it followed by parenthesis.

```
doSomething();
doSomething();
doSomething();
doSomething();
```

- In this example, we called the function *doSomething();* four times.

Twinkle Twinkle Little Star Example

- Suppose I want to write code to sing “Twinkle Twinkle Little Star”

```
console.log("Twinkle, twinkle, little star,");
console.log("How I wonder what you are!");
console.log("Up above the world so high,")
console.log("Like a diamond in the sky.");
```

- To sing it again, I have to rewrite all the code. This is not DRY!

```
console.log("Twinkle, twinkle, little star,");
console.log("How I wonder what you are!");
console.log("Up above the world so high,")
console.log("Like a diamond in the sky.");
```

Twinkle Twinkle cont.

- We can write a function to help us out.

```
function singSong() {
    console.log("Twinkle, twinkle, little star,");
    console.log("How I wonder what you are!");
    console.log("Up above the world so high,");
    console.log("Like a diamond in the sky.");
}
```

- To sing the song, we just need to call singSong();

```
// to sing the entire song 4 times
singSong();
singSong();
singSong();
singSong();
```

Arguments

- Often we want to write functions that take inputs.

```
function square(num) {
    console.log(num * num);
}
```

- Now when we call *square* we need to pass in a value.

```
square(10);  //prints 100
square(3);   //prints 9
square(4);   //prints 16
```

Arguments cont.

- Another example:

```
function sayHello(name) {
    console.log("Hello there " + name + "!");
}
```

- When we pass in *Blake,* it will now say “Hello there Blake!”

```
sayHello("Blake");
```

Arguments cont.

- Whatever you pass through to the argument will now hold the value to the variable.
- Functions can have as many arguments as needed.

```
function area(length, width) {
    console.log(length * width);
}

area(9, 2); //18

function greet(person1, person2, person3) {
    console.log("hi " + person1);
    console.log("hi " + person2);
    console.log("hi " + person3);
}

greet("Harry", "Ron", "Hermione");
```

The Return Keyword

- Often we want a function to send back an output value.
- We use the *return* keyword to output a value from a function
- This function capitalizes the first char in a string:

```
function capitalize(str) {
    return str.charAt(0).toUpperCase() + str.slice(1);
}

var city = "paris";              //paris
var capital = capitalize(city); //Paris
```

- We can capture the returned value in a variable.

The Return Keyword cont.

- The *return* keyword stops the execution of a function.

Assignment 8.1: Functions Problem Set

- Write a function *isEven()* which takes a single numeric argument and returns true if the number is even, and false otherwise.
- Write a function *factorial()* which takes a single numeric argument and returns the factorial of that number.
  - The factorial of 4 is 4 x 3 x 2 x 1
  - The factorial of 6 is 6 x 5 x 4 x 3 x 2 x 1
- Write a function *kebabToSnake()* which takes a single kebab-cased string argument and returns the snake_cased version.
  - Basically, replace “-”s with “_”s
  - kebabToSnake(“hello-world”); => “hello_world”
  - kebabToSnake(“dogs-are-awesome”); => “dogs_are_awesome”

JS Scope

- Scope is the **context** that code is executed in.
- A variable declared **inside** of a function can only be accessed within that function.
- A variable declared **outside (global scope)** of a function can be accessed inside of a function.
- Assigning a value of an already outside declared variable within a function without the var keyword will overwrite the value that was declared outside.