Program Flow: Functions

Making Our Programs Modular

We'll cover the following

Now that you have an understanding of the basic data types and operations, we should try to make our programs a little more interesting. We will do this by manipulating the way in which the Javascript program executes. One way we can accomplish this is through the use of **functions**.

Functions

Functions allow us to repeat tasks that involve a similar sequence of steps. You've already seen the `console.log()` **function**, which allows us to predictably *log some output* to the console.

Let's look at an example of a function definition that finds the sum of two numbers:

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

We start by creating a variable, which we then define as a *function* by using the `function(){}` syntax.

Within the parentheses are a set of **arguments**, which are values that are used or manipulated by the function in some manner. If you have *multiple* arguments, they are each separated by a comma. A function can also have *no arguments*.

The code to be executed by the function lies within the curly braces (`{}`). If you expect the function to *give back* some value, it should include a **return statement** (which is done by using the keyword `return`), followed by the value you want to be returned. If you don't expect your Javascript function to return a value, you *do not* have to include a return statement.

You may notice if you run the code above, nothing happens. In order to make use of our function, we must **invoke** or **call** the function.

A function is **called** like so:

```
functionName();
```

The parentheses together, `()`, are an *operator* that **initiates a function call**. If you have arguments for the function to make use of, you must include those within the parenthesis.

Let's make use of the `sum` function we just created:

```
var onePlusTwo = sum(1, 2);
var twoPlusTwenty = sum(2, 20);

console.log(onePlusTwo, twoPlusTwenty);
```

In this example, we make use of two different functions:

- the `sum()` function, where we pass **two numbers** as arguments
- the `console.log()` function, where we supply **the newly created variables** as arguments

In both cases where we use `sum()`, numbers are passed, so the value *returned* will also be a number. This number is then placed into the console using `console.log()`.

Why are functions important?

The most compelling reason to use functions is that they allow us to **reuse code**

and create **modules** to perform procedures we plan on using repeatedly. Though

the example above isn't much more useful than just using the `+` operator, you will see that functions are very useful as your code starts to get more complex.

Check your Understanding

1

What will the `sum()` function return if we **invoke** it like so?

```
sum(30, -30);
```

2

What will the returned value be from the following function call?

```
sum("some", "string");
```

Retake Quiz

Exercise

Write a function named `subtract`, that takes two values as arguments and **returns** the result of subtracting those two values.

```
//write a function definition that finds the result of subtracting two values.
var subtract = function(){

}
```

Hint 1 of 2

Pass two parameters in the function header

Now that you have learned about functions in javascript, let's learn about conditionals in the next lesson.