“The Web Design Workshop” Decal – Spring 2014
Programming Handout 6

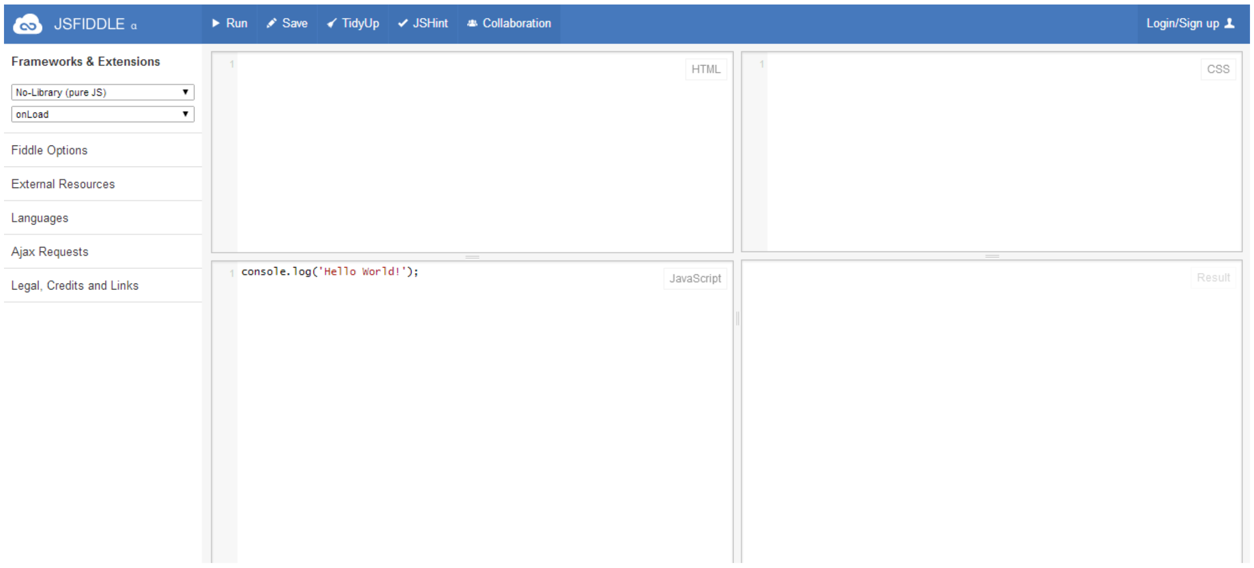
Goals:

1. Using console.log and variables
2. Functions and Loops
3. Arrays

1 – Using console.log and variables

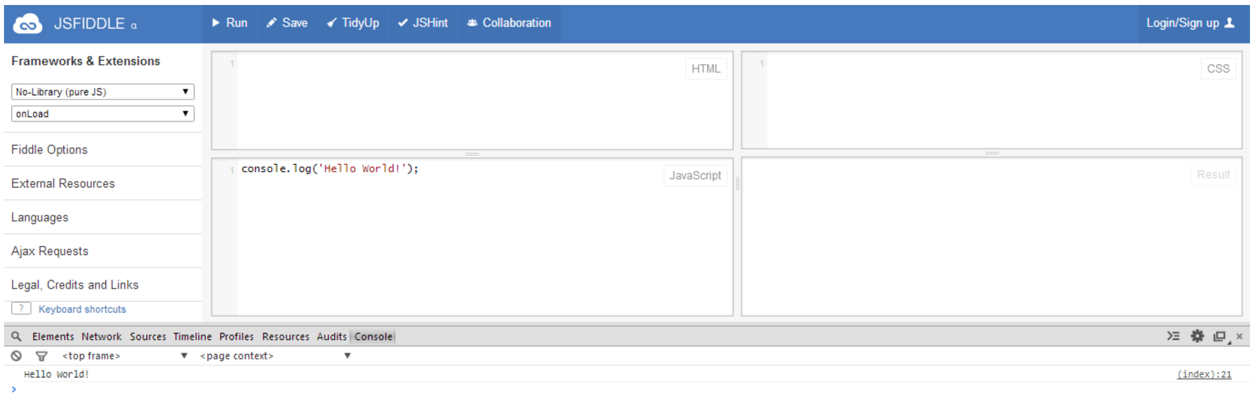
1. Go to http://jsfiddle.net on Chrome, and type a *console.log(“Hello World”);* in the JavaScript portion of the page.

   It should look something like this:



```
console.log('Hello World!');
```

If you right click your browser and go to **Inspect Element,** you will see the **Console** tab on the top-right. Click this to see all the output from your console.log(..);



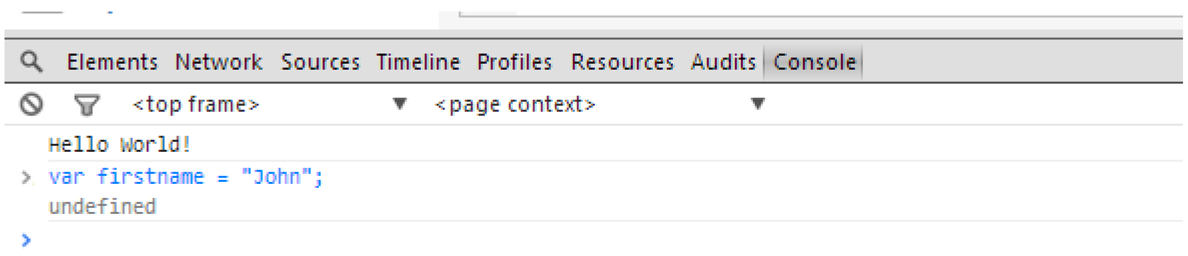
```
console.log('Hello World!');
```

```
Hello World!
>
```

console.log allows you to print, or log, content/variables into the console during development. This is extremely helpful when debugging, so keep this in mind!

While we are in the console, let's do some basic JavaScript.

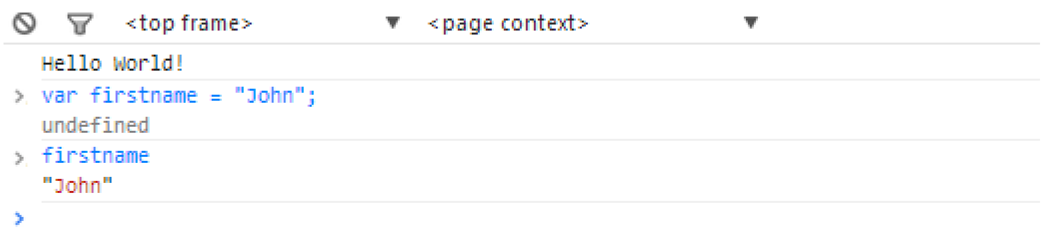
2. Click the space next to the blue arrow, right bellow "Hello World!" and let's type **var firstname = "John";**


Elements Network Sources Timeline Profiles Resources Audits Console
<top frame>
<page context>

```
Hello World!
> var firstname = "John";
undefined
>
```

Notice we have **undefined** below our firstname initialization. Ignore this, since it just means we are not outputting anything from our line of instruction.

Now, just type **firstname**. And you will get an output of whatever firstname is (John).

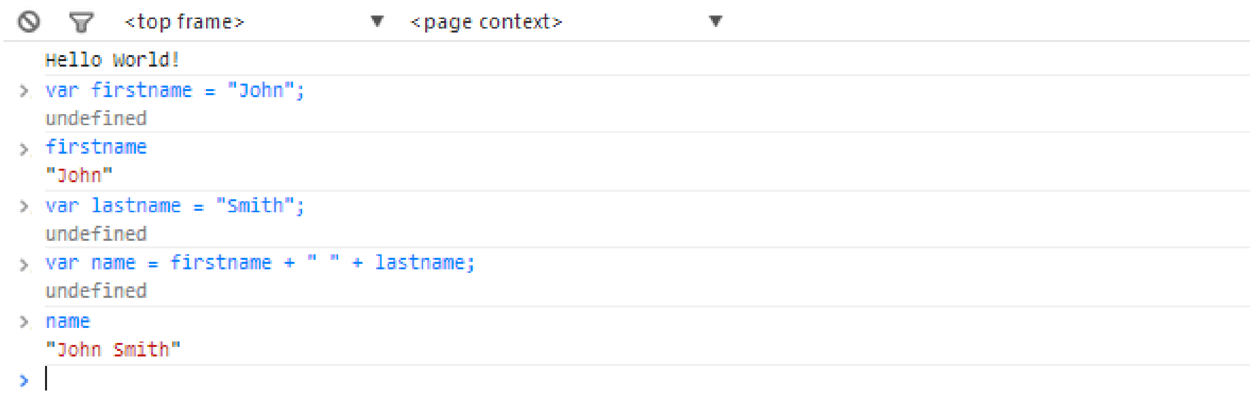


```
Hello World!
> var firstname = "John";
undefined
> firstname
"John"
>
```

3. Let’s add a lastname while we are at it. Type **var lastname = “Smith”;** and hit enter.

   Then, let’s **concatenate** (append both strings) together into one variable, **name.**

   So, set **var name = firstname + “ “ + lastname;** and hit enter.



```
Hello World!
> var firstname = "John";
undefined
> firstname
"John"
> var lastname = "Smith";
undefined
> var name = firstname + " " + lastname;
undefined
> name
"John Smith"
> |
```

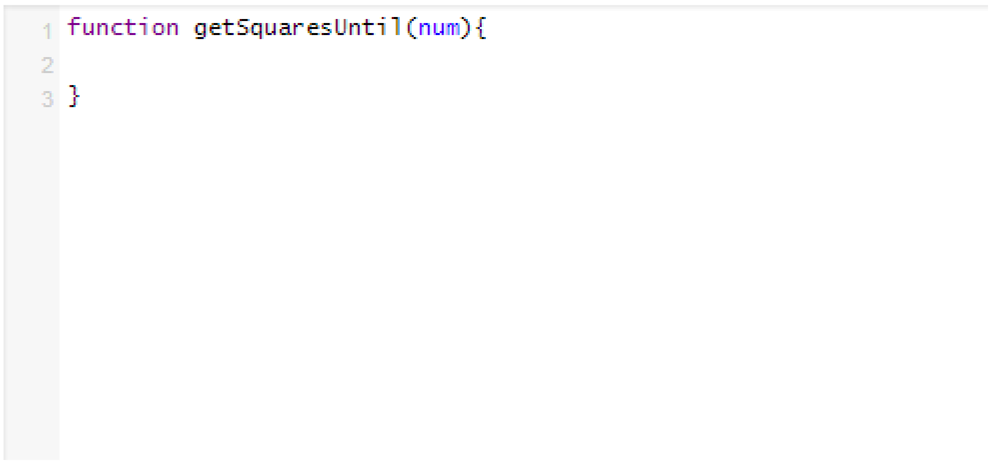
   You should get the screen above. Again, remember that **+** for strings concatenates, and for numbers they add. You can also **+** a string and a number, but the result will be the string and number appended into a new string.

2 – Functions and Loops

1. Good warmup! Now let’s do more powerful computations.

   Let’s write a function that prints every number’s squared value, starting with 1, until a number that we specifcy (if we input into the function 7, we should get 1*1, 2*2, 3*3… until 7*7).

   Go back to jsfiddle, clear out anything inside, and let’s define a simple **function**. Let’s call it **getSquaresUntil**. It will take in 1 argument, the number we want to stop at. We should get this:



```
function getSquaresUntil(num){

}
```

The **num** can be any name you want to give.

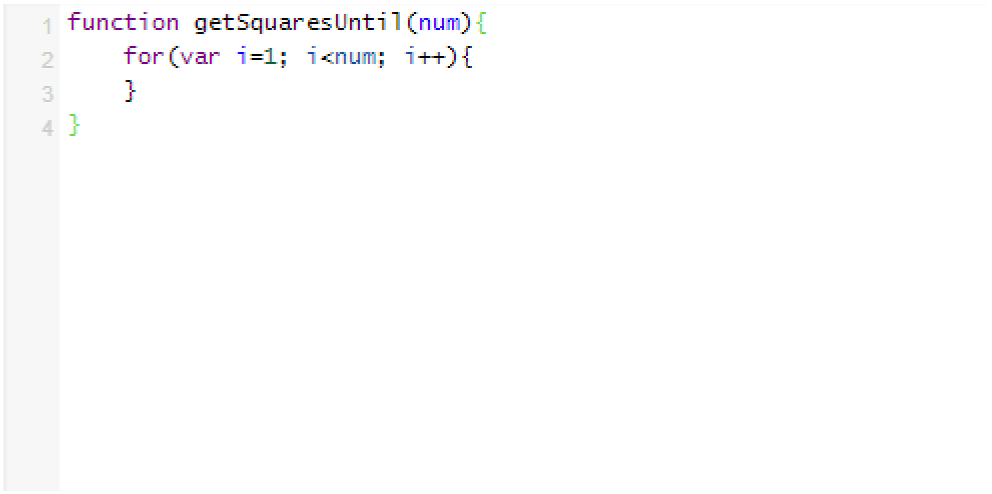
2. Now, we need a loop to loop through the our values from 1 until we hit num. There are 2 ways we can do this: use a for loop or use a while loop.

   Let's use a for loop for this example. Type in:

   for(var i=1; i< num; i++){
   }

   Again, you can use another name instead of i if you'd like. Remember, we are doing 3 things in this one line: initializing i to 1, incrementing it by 1 every loop, and it continues until i is no longer less than num (so in this case, when i equals num).

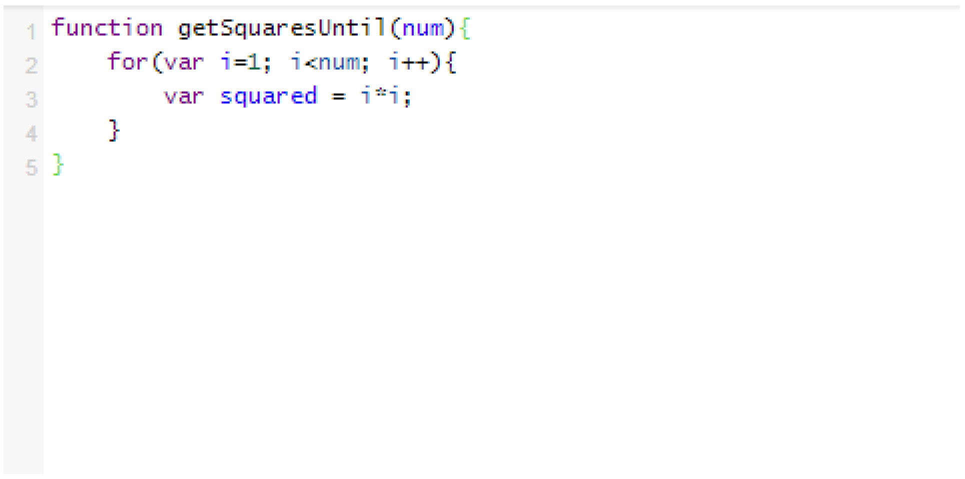
   We now get this:



```
function getSquaresUntil(num){
    for(var i=1; i<num; i++){
    }
}
```

3. Next, we need to do our computation.

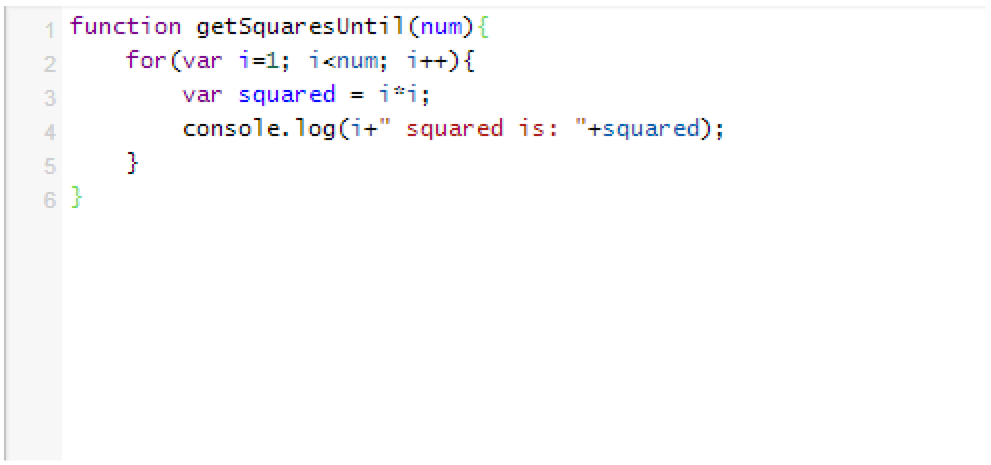
Inside the **for** loop, let’s create a variable called **squared**, and set it equal to **i*i.** So for every iteration of the loop, i will be a different value, and same goes for squared.



```
function getSquaresUntil(num){
    for(var i=1; i<num; i++){
        var squared = i*i;
    }
}
```

Some people may initialize variables outside a loop (say var squared; above the for loop, and simply do squared = i*i; inside). This is perfectly fine. Some people may argue one way is optimally better, but often times the performance will be the same. One reason you may want to initialize variables inside a loop is for scoping purposes (so **squared** can only be accessed inside the loop, not outside).

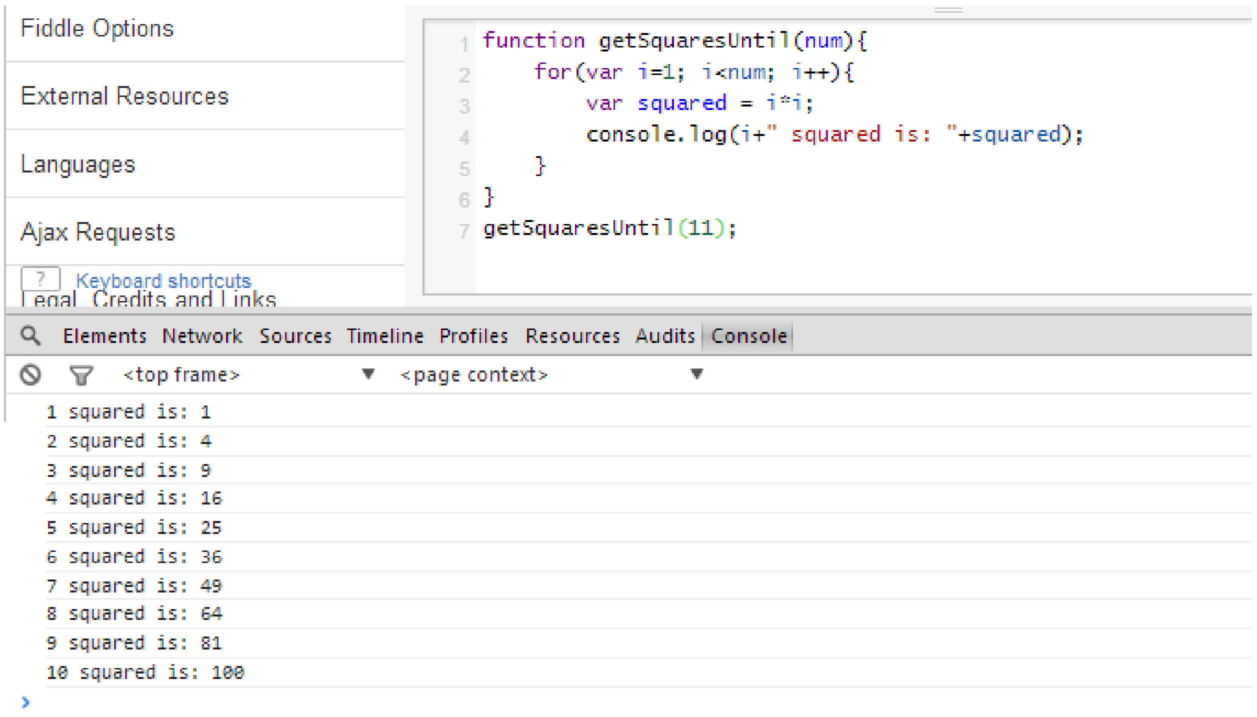
4. Next, let’s output **squared** into our **console**. Right after setting the squared value, let’s do a **console.log** that tells us what the value is at that moment in the loop.



```
function getSquaresUntil(num){
    for(var i=1; i<num; i++){
        var squared = i*i;
        console.log(i+" squared is: "+squared);
    }
}
```

5. Alright, that should do it! You have a function!

   Now, to actually see any results, you must call the function outside of it. Say we want squares until 10, what should we type?



```
function getSquaresUntil(num){
    for(var i=1; i<num; i++){
        var squared = i*i;
        console.log(i+" squared is: "+squared);
    }
}
getSquaresUntil(11);
```

```
1 squared is: 1
2 squared is: 4
3 squared is: 9
4 squared is: 16
5 squared is: 25
6 squared is: 36
7 squared is: 49
8 squared is: 64
9 squared is: 81
10 squared is: 100
```

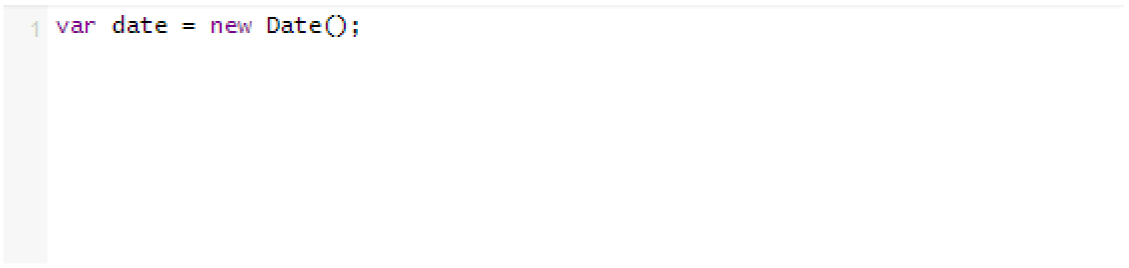
We called the function with **getSquaresUntil(11)**. Why? It's because in our for loop, we set the loop condition to i<num. We could have set it to i<=num, and called with **getSquaresUntil(10).** Either will work.

3 – Arrays and Conditionals

1. Now that you have a better understanding of functions, let's write another one! This time, we will get the current day of the week (e.g. Monday, Tuesday, etc.)

   Clear everything in your jsfiddle and let's start from scratch.

   First, enter in this line: **var date = new Date();**



```
var date = new Date();
```

This is something new! We have not covered **objects** yet, but what this line does is it creates a **Date Object** that we save into our variable, date.

What are objects? They are essentially their own entity, with their own functions and attributes. For example, a **Date** object has functions that allows you to get the current day, month or year, and even allows you to set it to anything you like!

We will use this Date object, now stored in **var date** to get the current day. However, one problem! The Date object will return a *number representation* of the current day of the week rather than the *literal representation* (0 will be Sunday, 1 will be Monday, 2 will be Tuesday, and so forth).

Our goal then is to take these numbers and give them meaning! (return their string counterpart)

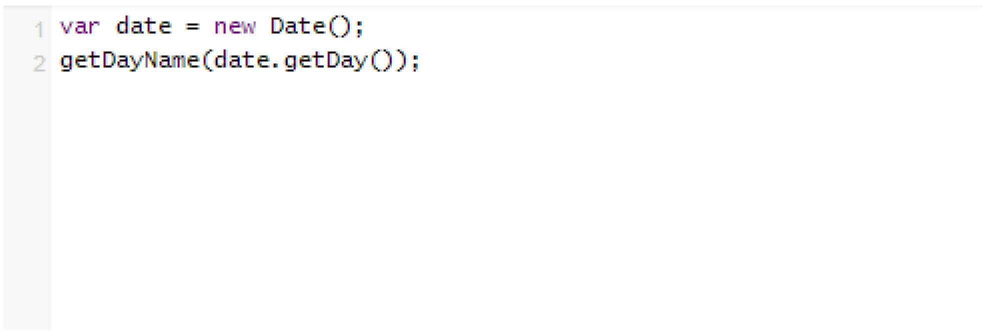
2. Right below our **var date = new Date()** line, let's add the following line:

   getDayName();

   We are going to call a function (that we will write momentarily) called **getDayName** that will take our number representation of the day, and output to the Console the literal representation.

   So, let's pass in an *argument* to our **getDayName** function. To pass in the day, type inside the parentheses of getDayName: **date.getDay();**

   You should have:



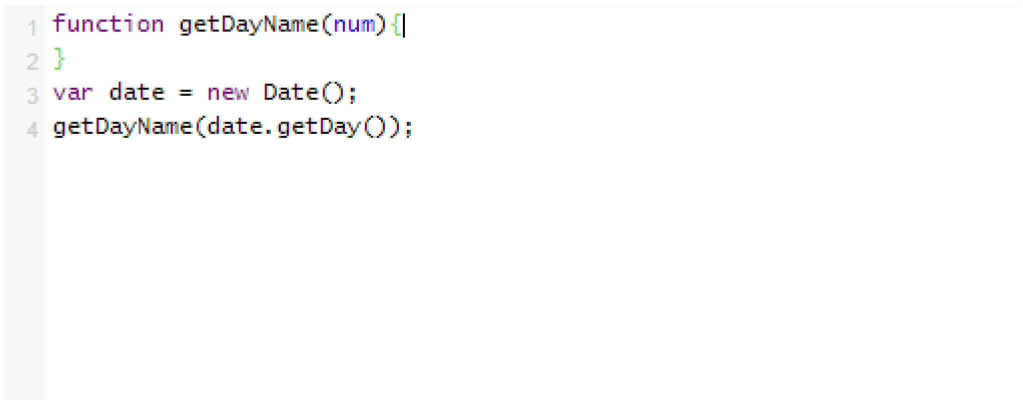
```
var date = new Date();
getDayName(date.getDay());
```

   Again, don't worry too much about objects, but **date.getDay()** is just telling our date variable to get the numeric day representation from our **Date** object, and pass it to our **getDayName** function.

3. Now, let's write our function!

   Write the function *above* our code (in order to call functions, they must already exist above the call, since JavaScript executes top down).

   Call the function **getDayName**, and give it an argument, let's call it **num** (for the numeric representation of the day).



```
function getDayName(num){
}
var date = new Date();
getDayName(date.getDay());
```

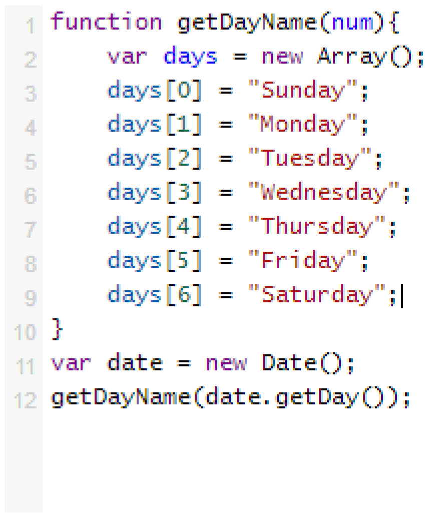
4. Next, let’s think about how to get the literal string for the day.

   You *could* have 7 if/else statements such as **if(num == 0){ console.log(“Sunday”) }else if…**, but let’s be a little more efficient!

   Remember that arrays can contain several strings, and remember that the first element in an array is at position **0.** Remember also that **date.getDay()** will give us numbers that correspond to days (so Sunday is 0, Monday is 1, Tuesday is 2, etc.).

   We can make an array for the 7 days, and save this array as a variable called **var days**, we can access Sunday simply with **days[0]**, and Monday as **days[1]**.

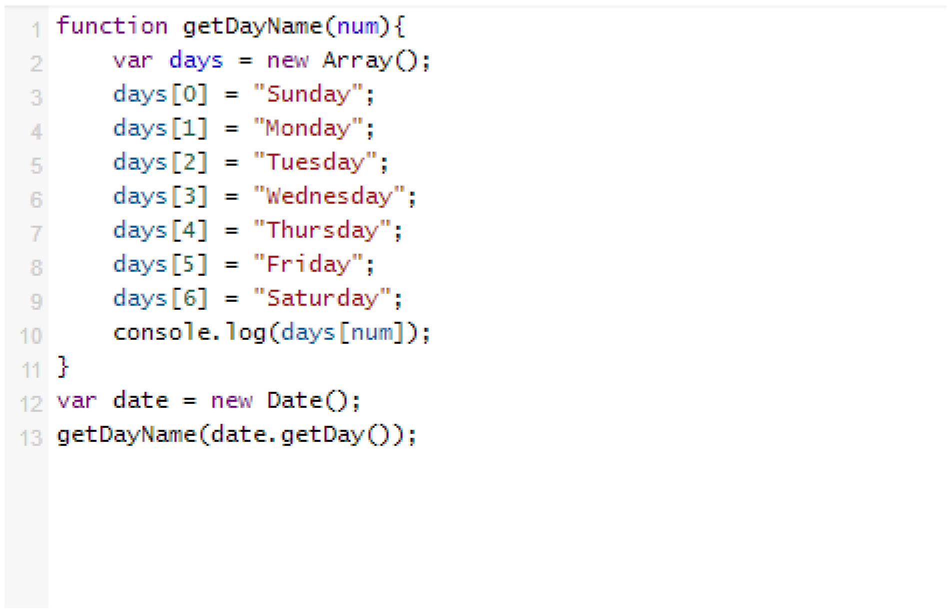
   If you’re lost, this is what we will do:



```
function getDayName(num){
    var days = new Array();
    days[0] = "Sunday";
    days[1] = "Monday";
    days[2] = "Tuesday";
    days[3] = "Wednesday";
    days[4] = "Thursday";
    days[5] = "Friday";
    days[6] = "Saturday";
}
var date = new Date();
getDayName(date.getDay());
```

   We will create a **new Array()** (hey! Array is an object, what we just talked about!), save it to **var days**, and for each element from 0 to 6, save the corresponding string representation of the day.

5. Ok! How do we access the array and print the day into the Console?



```javascript
function getDayName(num){
    var days = new Array();
    days[0] = "Sunday";
    days[1] = "Monday";
    days[2] = "Tuesday";
    days[3] = "Wednesday";
    days[4] = "Thursday";
    days[5] = "Friday";
    days[6] = "Saturday";
    console.log(days[num]);
}
var date = new Date();
getDayName(date.getDay());
```

With the single line **console.log(days[num]);**

We used console above, for our squares function. Same thing, we will use it here, and output **days[num]**, which is taking the **num** (passed in as **date.getDay()**), and using our array *days*, finding the corresponding day represented as a String!

Check your Console, and you should see today's date!!

Awesome! ☺

Congratulations! Let's review what we learned:

- We reviewed how to use console.log to debug
- We reviewed how to use variables and concatenation
- We reviewed how to use a for loop
- We reviewed how to write functions, and calling them
- We reviewed how to use arrays