Module 6 Programming II

Objective: Learn the basics of programming with functions and objects to build your first project.

At this point, most of the content about general programming concepts used to work with data has been covered.

However, a huge concept in programming is the concept of DRY programming.

DRY stands for Don’t Repeat Yourself.

So far we have built several small scripts. But if you think about all the applications you use, often the same action needs to be triggered over and over again. Right now as we have written code it only works on specific data we give it or something a user inputs, and it is only triggered once and can not be reused over and over.

The way we do this is through the use of Functions.

Why functions?

If we wanted to add two numbers.

We can say:

```
let num1 = 1
let num2 = 3
let added number = num1 + num2
//this works, but only for the specific numbers we gave it. If we wanted to
//reuse this we would need to re-write it again, which isn't DRY.
// Instead let's rewrite this as a function:
function add(num1, num2){ // here we are telling our function to expect 2
// inputs, which will be represented in the function as num1 and num2
      sum = num1 + num2
      return sum // the return statement ends the function and sends the
returned value to // be used as an input or saved as a variable.
}
// Just writing a function does not cause it to work tho it just defines it for
```

```
further use.
// to use functions we must call them:
add(1,3) // 4
add(3,4) //7
added = add(10,10)
added // 20
```

Please note in the above example we are writing out functions to take 2 arguments **num1** and **num2**. These act as placeholders for when we actually call the function and pass in actual numbers. Arguments are often part of functions but not always. A function can work without any arguments if written to do so see the below example:

```
function say_yer(){
      console.log("YERRRRR");
     return true
}
say_yer(); // "YERRRRR"
```

Functions:

Return Vs. Console.log()

So far working with viewing the output of a script we rely on **console.log** to give us our result on the screen.

Please note, Return in functions is not the same thing as a console log. It is not about displaying the info, but taking the output of a calculation so you can use it on something else, either to pass it into another function or to a variable for storage.

In Short:

Console.log SHOWS you something
Return GIVES you something

Functions as containers
It is important to understand that functions do not do anything alone, except create a label to reuse a chunk of code. Therefore, literally every script we have written thus far can be written as a function that can be reused.

Well written functions should have only one purpose.

Let's take the example from our loops section about going through an array of numbers and splitting them into even and odd numbers and returning an array.

The implementation is largely the same but there are a few key differences:

```
arr1 = [22,3,15,21,14,4,6,64,23,44]
even =[]
odd = []

for (let i = 0; i < arr1.length; i++){
     If (arr1[i] % 2 == 0){
           even.push(arr1[i])
      }
      else {
            odd.push(arr1[i])}
      }
```

The Function Version:

```
function even_odd(arr){
     let even = [ ]
     let odd = [ ]
     for (let i=0; i< arr1.length; i++){
          if (arr[i] %2 ==0){
               even.push[i]
          }
          else{
              odd.push[i]
             }

      }

      let results = []
      results.push(even)
      results.push(odd)
       //A function can only return one value, so to return two
//different arrays we just
 // we need to put them both in another array and just return that we
can then access
// even as results[0] and odd as results[1]
     return results

}

let arr1 = [22,3,15,21,14,4,6,64,23,44]

even_odd(arr1);
```

Lets functionalize another example we did previously:

```
let my_name = prompt("What’s your name?")
console.log("hello" + name)

Function say_hi(name){
     return "hello" + name
}

console.log(say_hi(my_name))
```

Function Scoping:

When declaring variables, it is important to understand the concept of scope. This is important because functions contain their own environment, so variables contained in functions may differ from ones outside the function, even if these variables with the same name:

See Below

```
let name = "Joe"; // this is global scope, as it exists outside of
any function

function namer(){
      //here is the function scope because we declaring this variable
     // w let inside the function
      let name = "Jim"
      console.log(name)//jim
       return name;
}

console.log(name)//Joe

name = namer();
console.log(name)//jim

//note: the name does not change until we reassign it. If we saved
//the result of the function to name2, for example, and
//console.logged name would have still been joe, while name 2 would
//have been Jim
```

```
Function Scope and Variable Scope Example:

Function Hello(){
  // do some job with local variables that should not be seen outside

  let message = "Hello"; // only visible in this block

  alert(message); // Hello
}

alert(message); // Error: message is not defined because it only exists
In the function

// Also the same variable can be used in different functions tied to
```

```
//different things. Remember functions are in their own bubble and
anything //defined in them only exists inside the function and can
overwrite it's //global(outside of the function) definition, but ONLY
within that //function.

Function Hello(){
  // show message
  let message = "Hello";
  alert(message);
}

Function Goodbye(){
  // show another message
  let message = "Goodbye";
  alert(message);
}
```

For more info on functions, scope check out this resource and examples

And one more
JavaScript Objects:

So far we have talked about almost all the major parts of objects, like most other programming languages.

We learned about the tools we use to store and alter information, make decisions based on the condition of the data, and automate the processes through loops.

We also learned how we can package different series of loops and conditions in a function, so we can group and reuse code blocks for a specific action.

However, what if we wanted to represent something more complex than a number or a word as data?

That is where the concept of objects and data structures come in.

So, if we represent a name as a piece of data, that is pretty straight forward

```
let name="Joe"
```

But what if we wanted to represent the entire person. We can do that as an object. What does an object contain?

Attributes: Data in the form of strings, bools, or numbers that are associated with the object.

Methods: As discussed earlier methods are just functions that are stored in objects.

Both attributes and functions are called from functions using “dot notation”:

object.attribute_label
object.method_name()

Strings arrays and every data type container have both attributes associated with it, because they are also objects:

```
let name = "Joe";
name.length // 3 this is an attribute of the string object
name.toUpperCase() // This is a function which takes the string value
attribute of our //string object "Joe" and turns it into "JOE"
```

Let’s try to make our first object:

Let’s try to make an object to save attributes representing a graffiti artist. In this case, we will use SKUF YKK

We declare objects as a set of key-value pairs stored in curly braces.

```
let writer = {
      tag : "Skuf",
      crew : "YKK",
      hometown : "Brooklyn",
      speciality : "Street Bombing",
      status : "All City"
      affiliates : ["Kez", "Pear", "PGism"],
      shakes_cans : function(){
```

```
                              console.log("Click Clack");
                             return true
                        }

}
```

Ok, so that one was for me let’s do another more generic one too.

```
let dog = {
      name: "Snoop",
      breed: "Doberman",
      age: 7,
      legs: 4,
      bark: function(){alert("Woof")}
}

dog.name // "Snoop"
dog.bark()//"Woof"

//Adding a new method or attribute to a new function is easy, just
//create the dot notation

dog.color = "black"
dog.handshake(){return "Goodboy"}

dog.handshake() //"Goodboy"
dog.color // "black"
```

Not only can we create one-off objects, but we can also create templates from an object, so we can create new objects based on the template (called an **instance**) as needed. This is why our string objects have the same attributes, even though they are defined differently.

THIS keyword in Objects.
This is a keyword that refers to that object itself. This allows you to write objects that take other aspects of the model as part of the object:

```
let student = {
      name : "Mike",
      music: "Trap",
      tellUsAboutYourself: function(){
            console.log("My name is  "+this.name+" and my favorite
genre of music is " + this.music )
}
}

student.tellUsAboutYourself() // :My name is Mike and my favorite
genre of music is //Trap"
```

Getters & Setters

A getter is a method used to retrieve an attribute value from our object:

```
// Create an object:
let person = {
  firstName: "John",
  lastName : "Doe",
  language : "en",
  get lang() {
    return this.language;
  }
};

person.lang// "en"
```

A setter is a method used to set the value of an attribute of the object:

```
let person = {
  firstName: "John",
  lastName : "Doe",
  language : "",
  set lang(lang) {
    this.language = lang;
  }
};

// Set an object property using a setter:
person.lang = "en";
```

So let's do an example assignment:

Create three objects:

Car
Team(Sports)
Animal

All three should have a min of 4 attributes, a getter function, a setter function, and an action function.

They can all be included in one objects.js file in the Module 6 folder.

Objects pt 2: Constructors

Now that we have an idea about what objects are and how they work, let’s learn how to write code that generates copies and variations of an object using functions.

A Constructor is a function that uses arguments, which can be any data type, array or function, and generates a new object:

```
function Person(first_name, last_name, eye_color, height){
      this.first : first_name,
      this.last : last_name
      this.eye_color: eye_color,
      this.height: height,
       this.name:function(){return this.first + " "+this.last}
}

// above we created the blueprint for the person object, now using the
new // keyword we can generate as many people as possible:
```

```
myFriend = new Person('Jerelyn','Rodriguez', 'brown', '5.4')
myGuy = new Person('Joe', 'Knows', 'brown', '5.11')

myFriend.first_name // Jerelyn
myGuy.last_name // Knows
myGuy.name() // "Joe Knows"

//note that while i can do this:
myGuy.favMovie = "Paid in Full"
myGuy.hello = function(){return "Yooooo"}

//We can't do this:
Person.favMovie = "Paid in Full" // will throw error
Person.hello = function(){return "Yooooo"}// will throw error

To add stuff to the constructor we need to go into the actual
constructor function and add it there.
```

Prototype and Constructors:

Last note on object constructors

Although I said there is no way to add anything to an object constructor after the fact, that's not exactly true, it is just not done the way you would with a 1 off object:

The JavaScript `prototype` property allows you to add new properties to object constructors:

```
function Person(first, last, age, eyecolor) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eyecolor;
}
Person.nationality = “English”// WILL NOT WORK
Person.prototype.nationality = "English";//works
Person.nationality //"English"

//same for methods

Person.greet = function(){console.log("Hello")}//will not work
Person.prototype.greet = function(){console.log("Hello")}//works
```

Congratulations you have now been exposed to a majority of programming concepts in the javascript language.

Now that we have gone through these topics you have a task.

Using your new-found programming skills you need to choose and build out 3 of the following projects:

Build an HTML CSS JS Calculator following this guide:
https://freshman.tech/calculator/

A JS To do list:
https://data-flair.training/blogs/javascript-project-to-do-list/

Building a weather app:
https://dev.to/drewclem/building-a-weather-app-with-vanilla-javascript-19p9

Tip Calculator:
https://www.freecodecamp.org/news/how-to-build-a-tip-calculator-with-html-css-and-javascript/

Meditation App
https://www.youtube.com/watch?v=oMBXdZzYqEk&feature=youtu.be

Build a paper scissor rock game:
https://youtu.be/qWPtKtYEsN4

Submitting Module 6:

Create a “projects folder inside your module_6 folder.

Inside create a separate folder for each project, and place all project files in the corresponding folder.

Link Github repo on google classroom.