Functions

JavaScript scripts can easily end up being hundreds if not thousands of lines of code. Some of those codes are repeating codes that can be simplified to minimize the amount of lines of code in our program. To help structure our code and make common operations reusable, we create functions and objects.

Functions are mini programs inside our program scripts. They can be used to segment sections of our code to make it easier to manage, or to perform repeated operations, or both. Functions are wrapped around code blocks, which contain the actual declarations that will be executed, and usually include some combination of variable assignments, operations, and conditions.

Functions do one of two things: they create a result immediately, such as changing the content of an element on a web page, or they provide a response or output to be used by other functions, called a return value. In JavaScript, we work with three types of functions: named functions, which are executed when called by name, anonymous functions, which are normally executed once they are triggered by a specific event, and immediately invoked function expressions, which they run the moment the browser finds them.

All functions have the same general structure. They start with the word **`function`**, which tells the browser, here I am declaring a function, followed by its name, two parentheses, and then a pair of curly braces that wrap the block of code.

Definition　　　function name

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

To run a named function, we call it by name at a location in the script where we want it to run. That means we can define a function at the top of the script and choose to run it at the bottom of the script. In fact, this is the coding standard for functions.

```
// call for function sum. var z will store the return value add
var z = sum(3,-1);
```

Technically, it does not matter where a function is in the JavaScript script because the browser will load all the JavaScript's code first and then execute it. But to make it easier for humans to read, we normally place a function before it is called in the script. This provides an appearance of logical structure in our code when you read it from top to bottom.

Finally, functions can return values from where they were called using the return keyword.

Everything that is returned is not executed directly, but is captured in a variable or used immediately in another function.

Anonymous functions do not have a name, so the parentheses appears right after function. Immediately invoked function expressions are anonymous functions with another parenthesis pair at the end to trigger them, all wrapped inside parentheses. Every function comes with an argument's object, an array of possible arguments you can pass to the function when you call it. These arguments are separated by commas, and can be used inside the function itself based on their names in the function declaration.

```
function sum() {
 var a = 30-10;
 return a;
}

// call anonymous function sum()
sum( );
```

Creating a function: user defined functions

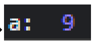
Named functions

Named functions are useful if we need to call a function many times to pass different values to it or just need to run it several times and it's also useful when functions just get really big and just clutter up the overall flow of the script. In that case, we create functions and put them above the main script to be called when needed. Named functions can have parameters and not, or with or without returning value.

Always remember that the order in which you place things in JavaScript really matters.

Example) function without parameters or returning value

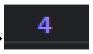
```
var a = 9;
var b = 3;
//
function x() {
      a>b? console.log("a: ", a) : console.log("b: ", b);
}
x( );
```

Result ➔ 

```
a: 9
```

Example) function without parameters but returning value

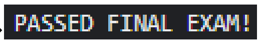
```
function message(){
  var x = 7;
  var y = -3;
  return x+y;
}

// call message()
console.log(message());
```

Result ➔ 

```
4
```

Example) function with parameters but no returning value

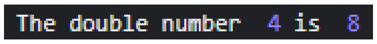
```
function upperMessage(message){
  var upper = message.toUpperCase();
  console.log(upper);
}

//call function upperMessage(message)
upperMessage('Passed Final Exam!')
```

Result ➔ 

```
PASSED FINAL EXAM!
```

Example) function with parameters and returning value

```
function doubleNumber(number){
  var x = number*2;
  return x;
}
// call function
var num = 4;
doubleNum=doubleNumber(num);
console.log('The double number ', num, 'is ', doubleNum);
```

result ➔ 

```
The double number  4 is  8
```

Functions are reusable. We can call a function as many times as necessary in the program.

```
var a = 5/9;
var b = 3;

function x(a, b) {
      var y = a+b;
      return y;
}
var c = 9;
var d = -5;

var num1 = x(a,b);
var num2 = x(10, 30);
var num3 = x(c, d);
```

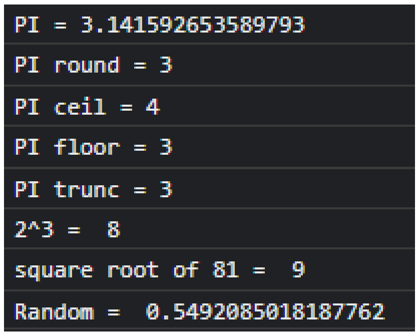
<u>Built-in function and methods. Math object</u>

JavaScript provides number of built-in functions that are global functions. Some of those built-in functions are math object function.

JavaScript math object allows users to perform mathematical tasks on numbers such as to square, round off, or randomly generate a number.

```
var num = Math.PI;
```

```
console.log('PI = '+ num);
console.log('PI round = '+ Math.round(num)); //returns the nearest integer
console.log('PI ceil = '+ Math.ceil(num));// returns the value rounded up to its nearest
integer
console.log('PI floor = '+ Math.floor(num)); //returns the value rounded down to its
nearest integer
console.log('PI trunc = '+ Math.trunc(num)); //returns the integer part of x:

console.log('2^3 =  '+ Math.pow(2,3)); //Math.pow(x, y), returns the value of x to the
power of y
console.log('square root of 81 =  '+ Math.sqrt(81)); //returns the square root of a
number
console.log('Random =  '+ Math.random());//returns a random number between 0 (inclusive),
and 1 (exclusive)
```



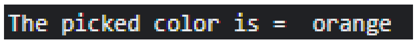
```
PI = 3.141592653589793
PI round = 3
PI ceil = 4
PI floor = 3
PI trunc = 3
2^3 =  8
square root of 81 =  9
Random =  0.5492085018187762
```

Example) create a function that randomly generate a number between 10 and 20 inclusive

```
function pickNum(){
  pick = 10+Math.round(Math.random()*20);
  return pick;
}
console.log(pickNum());
```

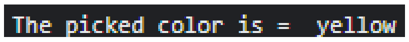
Example) Create a function that randomly pick a color from an array.

```
var colors = ['red','green','blue','yellow','orange'];
function colorPick(colorL){
  pick = Math.random()*colorL;
  return pick;
}
pickedIndex = Math.round(colorPick(colors.length-1));
console.log('The picked color is = ',colors[pickedIndex]);
```



```
The picked color is =  orange
```

Refresh the browser



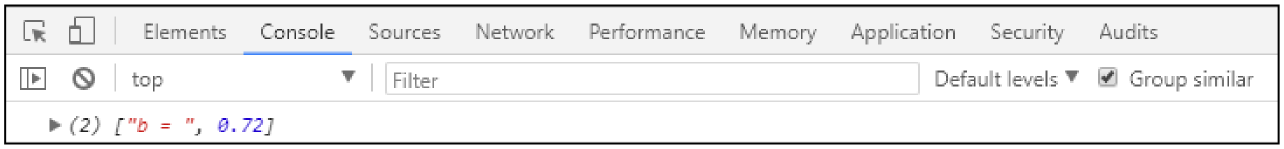
```
The picked color is =  yellow
```

Anonymous functions

Anonymous functions do not have names, so they need to be tied to something, a variable, or an event, or something similar to run. Anonymous functions are used mainly when the function is not called often in different part on the script.

Example) Anonymous function without argument and return value

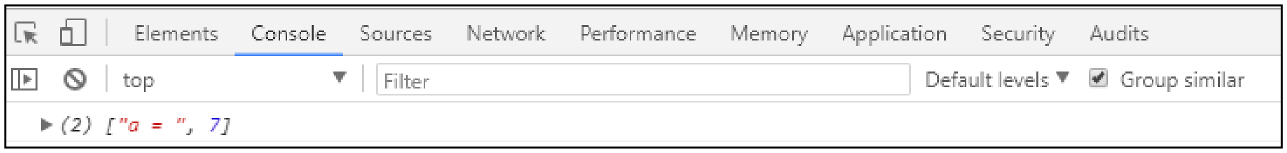
```
var a =5/7;
b = 18/25;
var x = function(){
  var result;
  a>b? result =["a = ",a] : result = ["b = ",b];
  console.log(result);
}

x( );
```



```
▶ (2) ["b = ", 0.72]
```

Example) Return the value in an anonymous function

```
var a =7;
b = -18;
var x = function(){
  var result;
  a>b? result =["a = ",a] : result = ["b = ",b];
  return result;
}

  console.log(x());
```



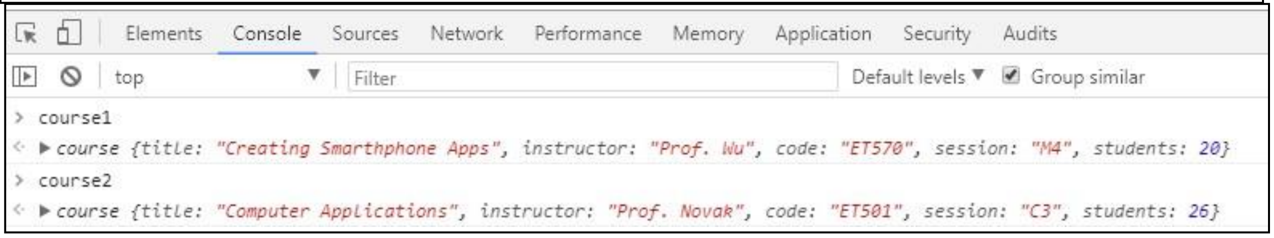
```
▶ (2) ["a = ", 7]
```

Object constructor using functions

Object Constructors are templates for creating objects that we define once and then we can use those templates again and again.

To make a constructor for any object we start with function, then we give our function a name and here we capitalize the first letter to signify this is an object.

```
// Create an object function named course
function course(title, instructor, code, session, students){
  this.title = title;
  this.instructor = instructor;
  this.code = code;
  this.session= session;
  this.students = students;
};
// call for object function
var course1 = new course("Creating Smarthphone Apps", "Prof. Wu", "ET570","M4",20);
var course2 = new course("Computer Applications", "Prof. Novak", "ET501", "C3",
26);
```

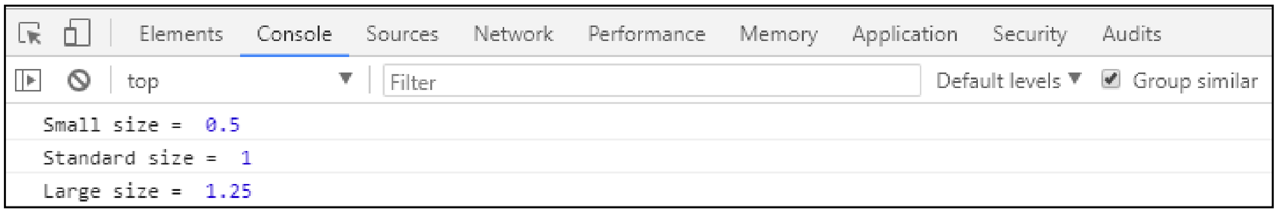


```
> course1
< ▶course {title: "Creating Smarthphone Apps", instructor: "Prof. Wu", code: "ET570", session: "M4", students: 20}
> course2
< ▶course {title: "Computer Applications", instructor: "Prof. Novak", code: "ET501", session: "C3", students: 26}
```

Closure

A closure is a function inside a function that relies on variables in the outside function to work

```
function giveMeEms(pixels){
  var baseValue = 16;
  // function inside the function giveMeEms
  function DoMath(){
      return pixels/baseValue;
  }
  return DoMath;
}
var smallSize =  giveMeEms(8);
var standardSize = giveMeEms(16);
var largeSize = giveMeEms(20);
// Display the different sizes
console.log("Small size = ", smallSize());
console.log("Standard size = ", standardSize());
console.log("Large size = ", largeSize());
```



```
Small size =  0.5
Standard size =  1
Large size =  1.25
```