For Loop

For Loop

1. Looping in programming languages is a feature which facilitates the execution of a set of instructions/functions repeatedly while some condition evaluates to true.
2. The for statement creates a loop that is executed as long as a condition is true.
3. It will only stop when the condition becomes false.

For Loop

```
for (initialization; condition; expression) {
    // code to be executed
}
```

1. Initialization is done (one time) before the execution of the code block.
2. Condition for executing the code block and exit loop
3. Expression is executed (every time) after the code block has been executed.

For Loop

```
for (var i = 0; i < 3; i++){
    console.log(i);
}
```

Output:

0

1

2

For Loop

```
for (var i = 5; i <= 8; i++){
    console.log(i);
}
```

Output:

5

6

7

8

Infinite Loop

1. All 3 statement in loop are options, in that case it will be infinite loop
2. Also if you do not provide condition in loop it will make loop infinite

```
for ( ; ; ){
    console.log("Hello");
}
```

For Loop

1. Printing table of 3 will require hard coded statements

```
console.log("3 x 1 = 3");
console.log("3 x 2 = 6");
console.log("3 x 3 = 9");
console.log("3 x 4 = 12");
console.log("3 x 5 = 15");
```

For Loop

1. With for loop it will be dynamic

```
var num = 3;
for (var i=1; i<=10; i++){
    console.log(num+" x "+i+" = "+(num*i));
}
```

Output:

3 x 1 = 3

….

3 x 10 = 30

Break

```
for (var i = 0; i < 8; i++){
    if(i == 4) {
        break;
    }
    console.log("I = "+i);
}
```

Output:

I = 0

I = 1

I = 2

I = 3

Continue

```
for (var i = 0; i < 8; i++){
    if(i == 4) {
        continue;
    }
    console.log("I = "+i);
}
```

Output:

I = 0

I = 1

I = 2

I = 3

I = 5

I = 6

I = 7

Nested Loops

1. If a loop exists inside the body of another loop, it's called nested loop.

```
for (var i = 0; i < 3; i++){
    for(var j = 0; j < 2; j++) {
        console.log("I = "+i+" J = "+j);
    }
}
```

Output:

I = 0 J = 0
I = 0 J = 1
I = 1 J = 0
I = 1 J = 1
I = 2 J = 0
I = 2 J = 1

Task

1. Find out if number is prime number or not
2. Prime number is divisible only by itself and 1 (e.g. 2, 3, 5, 7, 11).

Task

1. Generate triangle output like below, Hint: nested loop required

```
    *
   **
  ***
 ****
*****
******
*******
```

Arrays

Arrays

1. JavaScript arrays are used to store multiple values in a single variable.
2. An array is used to store a collection of data
3. It is an ordered collection which store elements in sequence
4. We can use array to store list of something like:
    a. Students
    b. Cars
    c. Food items

Arrays

1. If you want to store temperature of last 7 days in variable, then you have to create 7 variables

```
var mondayTemprature = 23;
var tuesdayTemprature = 12;
var wednesdayTemprature = 35;
var thursdayTemprature = 30;
var fridayTemprature = 27;
var saturdayTemprature = 19;
var sundayTemprature = 22;
```

Arrays

1. Now what if you want to store temperature of morning and evening for 7 days. That will be 14 variables
2. How about temperature for a year morning and evening that 730
3. It will be very difficult to manage and assign names to these variables

Arrays

1. If you want to find out all days temperature was above 30 then it will be quite difficult
2. If you want to sort temperature in ascending or descending order that will not be possible with separate multiple variables

Arrays

1. With Arrays you can create single variable and hold all temprature in it.
2. With Array you will be able to find and sort temperature easily

```
var temperatures = [34,12,27,65,34,28,
19];
```

Creating an Arrays

1. Creating an Array using array literal

```
var food = ["Pizza",   "Burger",   "Snacks"];
```

2. Creating an Array using *new* Keyword

```
var foods = new Array("Pizza", "Burger",
"Snacks");
```

3. Both are same, first one more recommended way to create array

Creating an Arrays

1. Array can be created for all datatypes or you can mix them in single array

```
var arr1 = ["Hello", "World",  "Bye"];
var arr2 = [29,  38, 16, 22];
var arr3 = [true,false,true,false,false];
var arr4 = [23.2,45.8,  98.12];
var arr5 = [{name: "John"}, {name: "Jason"}];
var arr6 = [74,  "Hello",  true, {name:
           "John"}];
```

Accessing Array Elements

1. You access an array element by referring to the index number.
2. To access element you provided number in square brackets

```
var foods = ["Pizza", "Burger", "Snacks"];
foods[0];  // Pizza
foods[1];  // Burger
foods[2];  // Snacks
```

Accessing Array Elements

1. Store result in variable or show output directly

```
var foods = ["Pizza", "Burger", "Snacks"];
var a = foods[0];      // Pizza
var b = foods[1];      // Burger
var c = foods[2];      // Snacks
alert(a);              // Pizza
alert(foods[2]);       // Snacks
```

Accessing Array Elements

1. Range of array index is from 0 to Array's length - 1
2. First element is on index 0
3. Second element on index 1
4. Third on index 2
5. Last element will be on array's length -1 e.g
    a. Array has 5 element then last element in on 4th index
    b. Array has 8 element then last element in on 7th index

Accessing full Array

1. The full array can be accessed by referring to the array name

```
var foods = ["Pizza", "Burger", "Snacks"];
console.log(foods); // Pizza, Burger, Snacks
```

Accessing Index that does not exists

1. If you create array with 3 elements and try to access 4th element, it will return *undefined*
2. There will be no error in accessing index that does not exists

```
var foods = ["Pizza", "Burger", "Snacks"];

console.log(foods[2]); // Snacks
console.log(foods[3]); // undefined
console.log(foods[8]); // undefined
```

Add/Update Element using index

1. You can use array index to add or update elements in array

```
var foods = [];

foods[0] = "Pizza";
foods[1] = "Burger"

foods[2] = "Snacks";
console.log(foods[0]); //Pizza

console.log(foods[2]); //Snacks
```

Add/Update Element using index

```
var foods = ["Pizza", "Burger", "Snacks"];

console.log(foods[1]); // Burger
foods[1] = "Sandwich"; // Updating existing element

console.log(foods[1]); // Sandwich
foods[3] = "French Fries"; // Adding 1 more element

console.log(foods[3]); // French Fries
```

Length property

1. You can find out number of elements in an array by length property

```
var foods = ["Pizza", "Burger", "Snacks"];
console.log(foods.length); // 3


var arr = [];
console.log(arr.length); // 0
```

Push function

1. Push function lets you add element in array without worrying about index
2. You don’t need to remember last index used to add in element, just call push function on array

```
var foods = [];
foods.push("Pizza");
foods.push("Burger");
foods.push("Snacks");
```

Push function

```
var foods = [];
foods.push("Pizza");
foods.push("Burger");
foods.push("Snacks");

alert(foods[0]);     // Pizza
alert(foods[1]);     // Burger
alert(foods[2]);     // Snacks
```

Push function -- Multiple input

```
var foods = [];
foods.push("Pizza");
foods.push("Burger", "Snacks");// Will add in sequence
foods.push("Sandwich");

alert(foods[0]);     // Pizza
alert(foods[1]);     // Burger
alert(foods[2]);     // Snacks
alert(foods[3]);     // Sandwich
```

Array Data Structure

1. A data structure is a specialized format for organizing, processing, retrieving and storing data
2. It enables efficient access and modification of data.
3. You can use same array syntax as:
    a. Random Access
    b. Stack (Last in First out)
    c. Queue (First in First out)

Random Access

1. You can access array elements from any index and update them

```
var foods = ["Pizza", "Burger", "Snacks"];

console.log(foods[1]); // Burger
foods[1] = "Sandwich"; // Updating existing element
console.log(foods[1]); // Sandwich
```

Iterating array with Loops

1. With loops you can utilize arrays in much efficient way.
2. If you want to find element in array, you can use loop to iterate over array in check if particular value exists in array
3. If you want to sort elements in array, you can use loop to iterate over each element and swap elements

Task

1. Create an array and fill it with numbers
2. Ask input from user
3. Find element in array that is provided by user

Splice function

1. Add add element in array we have used
    a. Push function -- add element in last
    b. Unshift function -- add element in start
    c. Index -- add element in last or replace existing element
2. If we want to add element in middle of array or any index other than first/last then we can use splice function

Splice function

1. Splice function can add one or more element on particular index in array
2. Splice function can replace one or more element on particular index in array
3. Splice function returns elements which removed from array, if no element removed then returns empty array

Splice function

```
var foods = ["Pizza", "Burger", "Snacks"];
console.log(foods);// "Pizza","Burger","Snacks"
foods.splice(1,0,"Sandwich");
console.log(foods);

//"Pizza","Sandwich", "Burger","Snacks"
```

This will add 1 element on index 1 and move all elements one index forward

Splice function

```
var foods = ["Pizza", "Burger", "Snacks"];
console.log(foods);// "Pizza","Burger","Snacks"
foods.splice(1,0,"Sandwich", "Fries");
console.log(foods);

//"Pizza","Sandwich","Fries","Burger","Snacks"
```

This will add 2 element on index 1 and 2 and move all elements two index forward

Splice function

```
var foods = ["Pizza", "Burger", "Snacks"];
console.log(foods);// "Pizza","Burger","Snacks"
foods.splice(1,2,"Sandwich");
console.log(foods);

//"Pizza","Sandwich"
```

This will remove 2 element from index 1 and will add 1 element on index 1

Slice function

1. To create array from element of existing array you can use slice function
2. We can create subset of array from existing array
3. Slice takes start and end index of array to create new array
4. Slice Syntax:
    a. slice(index of array, end index)
    b. End index is exclusive, if you say 4 that means 3rd index

Slice function

```
var foods = ["Pizza", "Burger", "Snacks", "Sandwich",
"Fries"];
console.log(foods);//
"Pizza","Burger","Snacks","Sandwich","Fries"

var arr = foods.slice(1,3);
console.log(foods);// output same as above
console.log(arr); //"Burger","Snacks"
```

Slice function

```
var foods = ["Pizza", "Burger", "Snacks", "Sandwich",
"Fries"];
console.log(foods);//
"Pizza","Burger","Snacks","Sandwich","Fries"
var arr = foods.slice(2); // Just start index
console.log(foods);// output same as above
console.log(arr); //"Snacks", "Sandwich","Fries"
```

Other Array functions

1. filter()
2. find()
3. indexOf()
4. lastIndexOf()
5. map()
6. reverse()
7. sort()

And others

String

String

1. JavaScript strings are used for storing and manipulating text.
2. String is zero or more characters written inside quotes

```
var a = "Hello World";
```

3. String is zero or more characters written inside quotes

```
var name1 = "John";   // Double quotes
var name2 = 'Mark';   // Single quotes
```

String Length Property

1. String has built-in length property which return length of character in string

```
var a = "Hello World";
alert(a.length);                    // 11
```

Escape Characters

```
var a = "Hello\'World"; // Single quote
var b = "Hello\"World"; // Double quote
var c = "Hello\\World"; // Backslash
var d = "Hello\nWorld"; // New Line
var e = "Hello\tWorld"; // Horizontal Tab
```

String functions

String functions

During development you will often require to manipulate string and for that string have many functions to do the job

1. toLowerCase()
2. toUpperCase()
3. slice()
4. indexOf()
5. lastIndexOf()
6. charAt()
7. replace()
8. split()

toLowerCase() function

1. toLowerCase function convert string in lowercase letters
2. It is useful when comparing user input

```
var food = "sANdWiCh";

var updatedFood = food.toLowerCase();
console.log(food);                    // sANDWiCh

console.log(updatedFood);             // sandwich
```

toUpperCase() function

1. toUpperCase function convert string in uppercase letters
2. It is useful when comparing user input

```
var food = "sANdWiCh";

var updatedFood = food.toUpperCase();
console.log(food);                    // sANDWiCh

console.log(updatedFood);             // SANDWICH
```

slice() function

1. slice() extracts a part of a string and returns the extracted part in a new string.
2. The method takes 2 parameters: the start position, and the end position (end not included).
3. String character count starts from 0 same as Arrays
4. First character at zero, second at 1

```
var a = "Hello World";
//H=0,e=1,l=2,l=3,o=4,space=5,W=6...
```

slice() function

1. This example extract portion from 6th index to 8th index

```
var a = "Hello World";
// 6 to (9-1)
var b = a.slice(6,9);     //returns Wor
```

slice() function

1. If we omit second parameter it will return rest of the string

```
var a = "Hello World";
// 6 to end
var b = a.slice(6);     //returns World
```

slice() function

1. If a parameter is negative, the position is counted from the end of the string.

```
var a = "Hello World";
//d=-1,l=-2,r=-3,o=-4,W=-5,space=-6,o=-7...
// -5 to -3
var b = a.slice(-5,-2);  //returns Wor
```

indexOf() function

1. indexOf() returns the index of the first occurrence of a specified text in a string
2. If not found, -1 is returned.

```
var a = "To be or not to be";
var b = a.indexOf("be"); //returns 3
```

indexOf() function

1. Second optional argument in indexOf() specify position to to begin search in string. If not provided its default to 0

```
var a = "To be or not to be";
var b = a.indexOf("be",  10);      //returns 16
```

lastIndexOf() function

1. lastIndexOf() returns the index of the last occurrence of a specified text in a string.
2. It searches backwards, from the end to the beginning
3. If not found, -1 is returned

```
var a = "To be or not to be";
var b = a.lastIndexOf("be");          //returns 16
```

lastIndexOf() function

1. Second optional argument in lastIndexOf() specify position to to begin search in string. If not provided its default to last index

```
var a = "To be or not to be";
var b = a.lastIndexOf("be",10);  //returns 3
```

2. It will start looking for text ‘be’ from index 10 to backwards

charAt() function

1. slice() function extract the portion of string provided the starting and ending positions
2. charAt() function takes single index input and return character at that index
3. Returns empty string if index does not exists on negative index provided

```
var a = "To be or not to be";
var b = a.charAt(7);   //returns r
```

replace() function

1. The replace() function replaces a specified value with another value in a string
2. The replace() function does not change the string it is called on. It returns a new string.

```
var str = "To be or not to be";
var b = str.replace("be","hello");
// result "To hello or not to be"
```

replace() function

1. By default, the replace() function replaces only the first match
2. To replace all matches, use a regular expression with a /g flag (global match) and without quotes

```
var str = "To be or not to be";

var b = str.replace(/be/g,"hello");

// returns "To hello or not to hello"
```

replace() function

1. To replace case insensitive, use a regular expression with an /i flag (insensitive)

```
var str = "To be or not to be";
var b = str.replace(/to/i,"hello");

// returns "hello be or not to be"
```

replace() function

1. Combine both g and i flog to replace all matches and case insensitive

```
var str = "To be or not to be";
var b = str.replace(/To/gi,"hello");
// returns "hello be or not hello be"
```

split() function

1. The split() function is used to split a string into an array of substrings, and returns the new array.

```
var str = "To be or not to be";
var b = str.split(" ");// split with space
// returns array: ["To", "be", "or", "not",
"to", "be"]
```

split() function

1. Split can be done with commas, spaces or any character

```
var str = "To,be or|not to,be";
var a = str.split(",");// Split on commas
var b = str.split(" ");// Split on spaces
var c = str.split("|");// Split on pipe
```

Other String functions

There are few more string functions you can learn

1. charCodeAt()
2. concat()
3. endsWith()
4. includes()
5. match()
6. repeat()
7. replace()
8. search()
9. startsWith()
10. substr()
11. substring()
12. trim()

Math functions

Math

Math class provides many functions that allows you to perform mathematical tasks on numbers

Math.round() function

1. Math.round(x) returns the value of x rounded to its nearest integer
2. E.g calculating average score will result number with decimal places and you need to round them

```
var average = (15 + 23 + 39) /3 ; // 25.6666
var roundedAverage = Math.round(average); // 26
console.log(roundedAverage);
```

Math.round() function

```
var a = Math.round(4.7);     // 5
var b = Math.round(4.1);     // 4
var c = Math.round(4.5);     // 5
var d = Math.round(-4.1);    // -4
var e = Math.round(-4.7);    // -5
var f = Math.round(-4.5);    // -4
var g = Math.round(5);       // 5
```

Math.ceil() function

1. Math.ceil(x) returns the value of x rounded **up** to its nearest integer

```
var a = Math.ceil(4.7);     // 5
var b = Math.ceil(4.1);     // 5
var c = Math.ceil(-4.1);    // -4
var d = Math.ceil(-4.7);    // -4
```

Math.floor() function

1. Math.floor(x) returns the value of x rounded **down** to its nearest integer

```
var a = Math.floor(4.7);     // 4
var b = Math.floor(4.1);     // 4
var c = Math.floor(-4.1);    // -5
var d = Math.floor(-4.7);    // -5
```

Math.random() function

1. Suppose you want to simulate the throw of a die. In the simulation, you want it to randomly come up 1, 2, 3, 4, 5, or 6
2. If you want build a game that will allow user to guess a number
3. Math.random() returns a random number between 0 (inclusive), and 1 (exclusive)
4. Everytime you execute this function it will return random value

Math.random() function

```
var num = Math.random();
// result will be like 0.5251908043871081
```

If you want to generate random number between some range then you have to add some calculations like:

```
var num = Math.random();
var num2 = (num * 6) + 1;

var dice = Math.floor(num2); // 1 to 6
```

Other Math functions

There are few more string functions you can learn

1. Math.pow()
2. Math.sqrt()
3. Math.abs()
4. Math.sin()
5. Math.cos()
6. Math.min()
7. Math.max()
8. Math.exp()
9. Math.log()

Controlling the length of decimals

1. In arithmetic operation you may face numbers with many decimal place

```
var average = (15 + 23 + 39) /3 ; // 25.666666666
```

2. To limit decimal places to specified number you can call toFixed() function on number and round last digit

```
var avg = average.toFixed(3);// returns 25.667
```

Date Object

Date

1. In JavaScript, you might have to create a website with a calendar, a train schedule, or an interface to set up appointments.
2. These applications need to show relevant times based on the user's current timezone
3. You might need to use JavaScript to generate a report at a certain time every day

Date

1. The Date object is a built-in object in JavaScript that stores the date and time.
2. It provides a number of built-in methods for formatting and managing that data.
3. Date objects are created with new Date().

Date

```
var date = new Date();
console.log(date);
//Thu Nov 07 2019 11:44:50 GMT+0500 (Pakistan
Standard Time)
```

This will be created according to the current computer's system settings.
It will show complete date with current timezone, if you change the timezone of your computer it will show different date

Thu Nov 07 2019 11:44:50 GMT+0500 (Pakistan Standard Time)

Looking at the output, we have a date string containing the following:

Day of the Week	Month	Day	Year	Hour	Minute	Second	Timezone
Thu	Nov	07	2019	11	44	50	GMT+0500

Creating Date Objects

1. There are 4 ways to create a new date object

```
new Date()
new Date(year, month, day, hours, minutes,
seconds, milliseconds)
new Date(milliseconds)
new Date(date string)
```

Creating Date Objects

```
new Date()
new Date(2019, 7, 11, 10, 25, 40, 300);
new Date(1565501140300);
new Date("2019/9/8 10:15:15");
new Date("2019/9/8 10:15:15");
new Date("January 12 2019 10:15:15");
```

Unix time

1. JavaScript, understands the date based on a timestamp derived from Unix time, which is a value consisting of the number of milliseconds that have passed since midnight on January 1st, 1970. We can get the timestamp with the getTime() method.

```
var date = new Date();

date.getTime();   // 1573110702109
```

Epoch time

1. Epoch time, also referred to as zero time, is represented by the date string 01 January, 1970 00:00:00 Universal Time (UTC), and by the 0 timestamp
2. Epoch time was chosen as a standard for computers to measure time by in earlier days of programming

```
var date = new Date(0);

console.log(date);

//Thu Jan 01 1970 00:00:00 GMT+0000 (UTC)
```

Retrieving the Date Components

1. Once we have a date, we can access all the components of the date with various built-in methods.
2. The methods will return each part of the date relative to the local timezone.
3. Each of these methods starts with get, and will return the relative number.

Date/Time	Method	Range	Example
Year	`getFullYear()`	YYYY	1970
Month	`getMonth()`	0-11	0 = January
Day (of the month)	`getDate()`	1-31	1 = 1st of the month
Day (of the week)	`getDay()`	0-6	0 = Sunday
Hour	`getHours()`	0-23	0 = midnight
Minute	`getMinutes()`	0-59	
Second	`getSeconds()`	0-59	
Millisecond	`getMilliseconds()`	0-999	
Timestamp	`getTime()`	Milliseconds since Epoch time	

Retrieving the Date Components

```
var date = new Date("June 14 2019 10:45:25");
```

```
date.getFullYear();// 2019
date.getMonth();   // 5
date.getDate();    // 14
date.getDay(); // 5
date.getHours();   // 10
```

```
date.getMinutes();   // 45
date.getSeconds();   // 25
date.getMilliseconds();//0
date.getTime();

//1560491125000
```

Modifying the Date

1. For all the **get** methods that we learned, there is a corresponding **set** method.
2. Where **get** is used to retrieve a specific component from a date, **set** is used to modify components of a date

Date/Time	Method	Range	Example
Year	`setFullYear()`	YYYY	1970
Month	`setMonth()`	0-11	0 = January
Day (of the month)	`setDate()`	1-31	1 = 1st of the month
Day (of the week)	`setDay()`	0-6	0 = Sunday
Hour	`setHours()`	0-23	0 = midnight
Minute	`setMinutes()`	0-59	
Second	`setSeconds()`	0-59	
Millisecond	`setMilliseconds()`	0-999	
Timestamp	`setTime()`	Milliseconds since Epoch time	

Modifying the Date

```
var date = new Date("June 14 2019 10:45:25");
console.log(date);
// Fri Jun 14 2019 10:45:25 GMT+0500 (Pakistan
Standard Time)
date.setFullYear(2017);
console.log(date);
// Wed Jun 14 2017 10:45:25 GMT+0500 (Pakistan
Standard Time)
```

Converting Day of Week to Text

1. getDay() function returns day of week, but it will give number representing day from 0 to 6
2. 0 represent Sunday, 1 represent Monday and so on
3. If you want on convert this value into text representation then you have to do it yourself
4. You create array represent day of week in text format and then map value from getDay() function to array

Converting Day of Week to Text

```
var daysList = ["Sun", "Mon", "Tue", "Wed",
"Thu", "Fri", "Sat"];
var date = new Date("June 14 2019 10:45:25");
var day = date.getDay(); // 5

var nameOfDay = daysList[day]; // Fri
```

Calculate Time difference

1. You can calculate time difference between two days using getTime and calculate difference in days
2. getTime returns time in milliseconds so you can find out time difference in milliseconds by subtracting dates
3. Then need find out of milliseconds in a day
   24 hours * 60 minutes * 60 seconds * 1000 milliseconds
4. And divide time difference in milliseconds with milliseconds of a day

Calculate Time difference

```
var d1 = new Date("June 14 2019 10:45:25");
var d2 = new Date("June 28 2019 10:45:25");
var timeDiff = d2.getTime() - d1.getTime();
var days = timeDiff / (1000 * 60 * 60 * 24);// 14
```