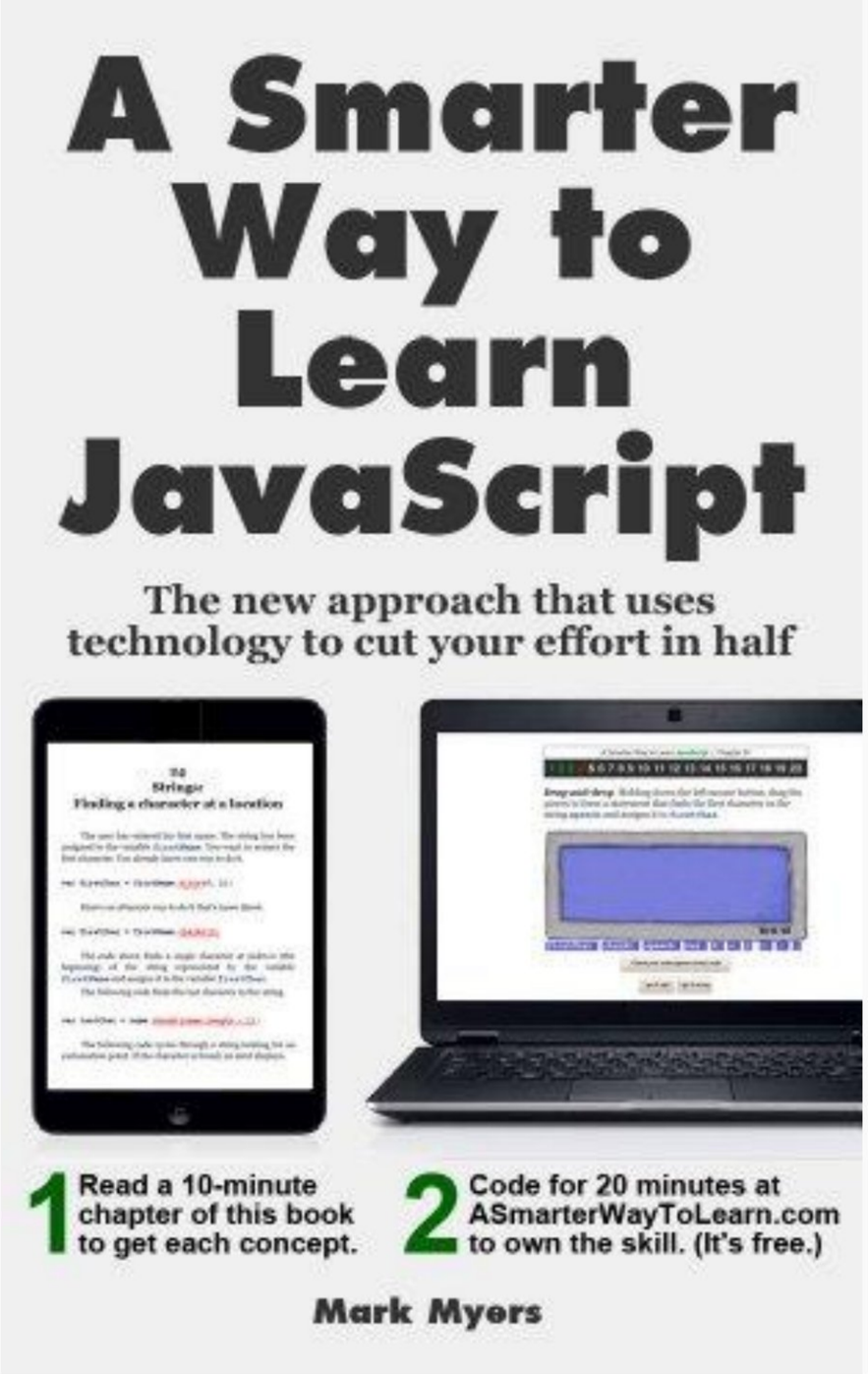
Book we will follow



Scanned with CamScanner

Online Exercises

Below link provide you online exercises

http://asmarterwaytolearn.com/js/index-of-exercises.html

Scanned with CamScanner

Installation

1. To work with JavaScript you don’t need any software
2. You just need to have Browser installed on your machine and you can use any text editor to write code.
3. To manage projects and organize files we use IDE
4. We will use Microsoft’s Visual Studio Code
5. To run JavaScript on server side we will use Node.js

Scanned with CamScanner

Installation

1. Download and install Visual Studio code
    a. https://code.visualstudio.com

2. Download install Node.js
    a. https://nodejs.org/en/

3. Use default setting while installing both tools

Scanned with CamScanner

Demo
How to use Visual Studio Code

Scanned with CamScanner

Initial Code and Setup

1. Open Visual Studio code and create two files
    a. index.html
    b. Index.js

Scanned with CamScanner

index.html file

```
<html>
    <head>
        <script src="./index.js"></script>
    </head>
    <body>
        Hello World
    </body>
</html>
```

Scanned with CamScanner

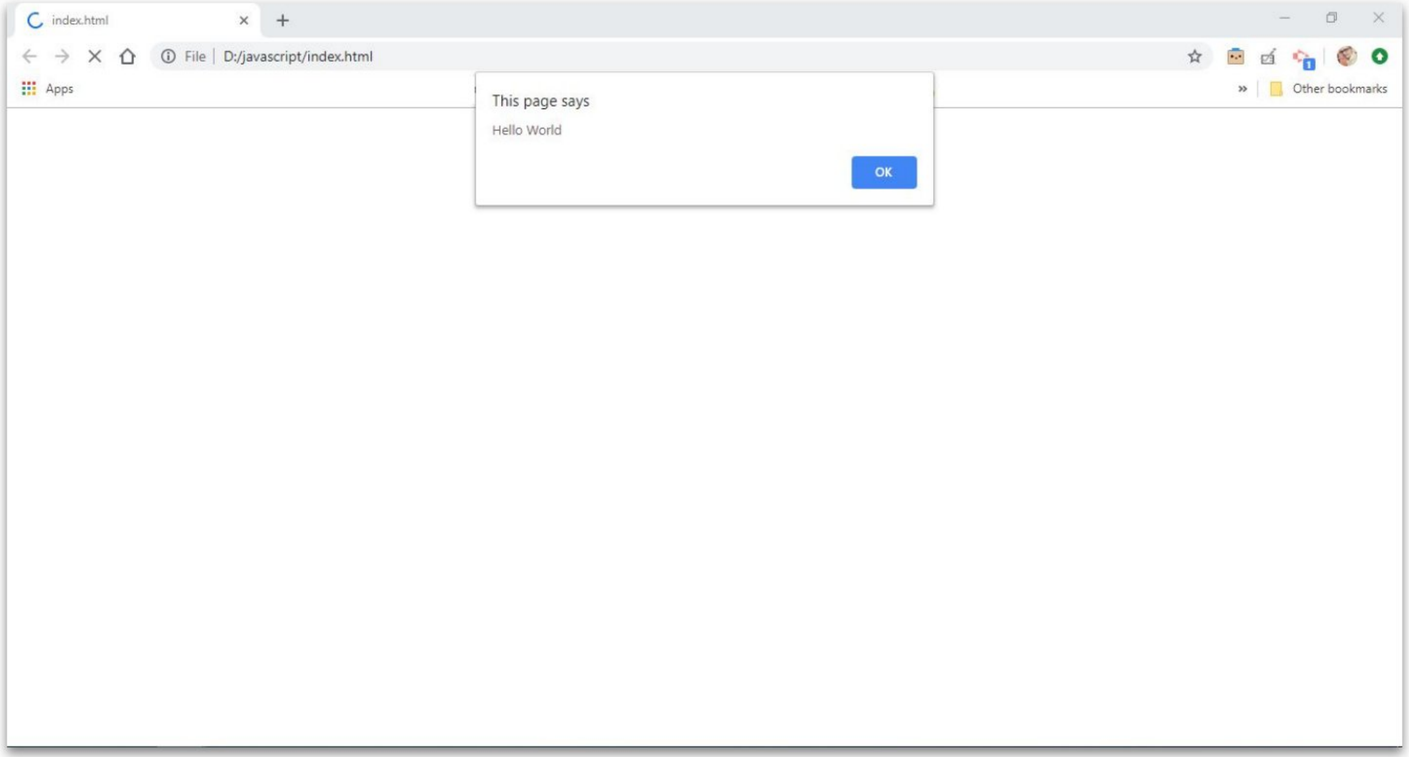
index.js file

```
alert("Hello World");
```

Scanned with CamScanner

Initial Code and Setup

1. Open file in browser and you will see alert in browser window



Scanned with CamScanner

Demo
Create First JavaScript Example

Scanned with CamScanner

Demo
How to run JavaScript in Node.js

Scanned with CamScanner

Alerts

1. An alert is a box that pops up to give the user a message.
2. Here's code for an alert that displays the message "Thanks for your input!"

```
alert("Hello World");
```

3. alert is a keyword
4. The quoted text "Thanks for your input!" is called a text string or simply a string.

Scanned with CamScanner

Alerts

1. ‘window.alert’ and ‘alert’ are the same
2. window is object in browser which contains many other objects and properties, and alert is one of them.
3. alert is a function that take any input and display it in popup to user
4. Alerts are not available when you are working with server side JavaScript in Node.js

Scanned with CamScanner

Console.log

1. console.log is function that write message on console/terminal
2. Objective of console.log is to create logs for debugging
3. Instead of displaying text to user it shows output in browser’s developer tool
4. Also when working with server side javascript, we can use console.log for logging and output will be in terminal

Scanned with CamScanner

Console.log

```
console.log("Hello World");
console.log(2+8);
```

Scanned with CamScanner

document.write

1. For testing purpose you can use document.write to display message or text in browser window

```
document.write("Hello Word");
document.write(2+8);
```

This will be displayed in browser window

Scanned with CamScanner

Variables

Scanned with CamScanner

Variables

1. Variables are containers for storing data values.
2. Variables are used to store information to be referenced and manipulated in a computer program
3. Variable is a term that refers to a particular value
4. A variable changes over time or based on user input.
5. A variable is assigned a value in one place and then used repetitively.

Scanned with CamScanner

Variables

Keyword to create variable

Value

```
var nationality = "Pakistani";
```

Variable name/identifier

Scanned with CamScanner

Variables

```
var nationality = "Pakistani";

var age = 25;

var isFeePaid = true;

var weight = 60.55;
```

Scanned with CamScanner

Declaration and Initialization

1. You can declare and initialize in single line or you can do that in two line

```
var age = 25;  OR

var age;    -- Declaration
age = 25;   -- Initialization
```

Only declaration will leave variable *undefined*

Scanned with CamScanner

Data Types

Scanned with CamScanner

Data Types

1. Variables contains data
2. Each data has type, that’s why we call it data type
3. Variables can hold numbers like 100 and text values like "John Doe"
4. In programming, text values are called text strings.

Scanned with CamScanner

Data Types

1. Six types are considered to be primitives.
    a. Number -- integers, floats etc
    b. String -- an array of characters
    c. Boolean -- true or false
    d. Null -- No Value
    e. Undefined -- a declared variable but hasn’t been given a value
    f. Symbol -- a unique value that's not equal to any other value

Scanned with CamScanner

Data Types

1. Complex Types
    a. Objects
    b. Functions

Function is callable object that executes a block of code, it lies under the object type

Scanned with CamScanner

Variable for String

1. To store text in variable we use string

```
var name = "Mark";
name = "John";
```

Scanned with CamScanner

String - Single quotes and double quotes

1. Text strings are always enclosed in quotes
2. You can use single or double quotes

```
var name = "Mark";
var nationality = 'US';
var message = "What is your father's number";
```

Scanned with CamScanner

Variable for Numbers

1. To store whole numbers or decimal values we use number data type

```
var age = 25;
var weight = 150.5;
var newWeight = weight + 20;
var originalNumber = 12 + 5;
```

Scanned with CamScanner

Variable for Boolean

1. To store true/false boolean data type

```
var isFeePaid = true;

var examPassed = false;
```

Scanned with CamScanner

Undefined

1. The undefined data type can only have one value-the special value *undefined*
2. If a variable has been declared, but has not been assigned a value, has the value undefined

```
var name;
var age;
```

Scanned with CamScanner

Undefined

1. You can be empty a variable by setting the value to undefined. The type will also be undefined

```
var name = "Mark";
name = undefined;
```

2. Now this variable contains no value, and it will cause error if we try to apply some operation on it

Scanned with CamScanner

Null

1. This is another special data type that can have only one value-the null value.
2. A null value means that there is no value.
3. It is not equivalent to an empty string ("") or 0, it is simply nothing

```
var name = null;
var nationality = "Mark";
nationality = null;
```

Scanned with CamScanner

Difference between null and undefined

1. undefined means a variable has been declared but has not yet been assigned a value.
2. null is an assignment value. It can be assigned to a variable as a representation of no value
3. undefined and null are two distinct types: undefined is a type itself (undefined) while null is an object.
4. Unassigned variables are initialized by JavaScript with a default value of undefined. JavaScript never sets a value to null. That must be done programmatically.

Scanned with CamScanner

JavaScript Data Types are Dynamic

1. JavaScript has dynamic types. This means that the same variable can be used to hold different data types
2. You can assign any type of value in variable any time and data type of variable will be changed

```
var name = "Mark";  // It's String
name = 25;  // Now changed to Number
name = true;  // Now changed to Boolean
```

Scanned with CamScanner

typeof Operator

1. You can use the JavaScript typeof operator to find the type of a JavaScript variable.
2. The typeof operator returns the type of a variable or an expression

```
typeof "Hello"        // Returns "string"
var a = 45;
typeof a;             // Returns "number"
typeof true;          // Returns "boolean"
```

Scanned with CamScanner

Statement and Expression

Scanned with CamScanner

Statements

1. A computer program is a list of "instructions" to be "executed" by a computer.
2. In a programming language, these programming instructions are called statements.
3. A JavaScript program is a list of programming statements.
4. A statement can set a variable equal to a value.
5. A statement can also be a function call, i.e. document.write().

Scanned with CamScanner

Statements

1. Statements define what the script will do and how it will be done.
2. Most JavaScript programs contain many JavaScript statements.
3. The statements are executed, one by one, in the same order as they are written.

Scanned with CamScanner

Statements

1. Each line below is a statement

```
var a = 4;          // Statement 1
var b = 2;          // Statement 2
var c = 0;          // Statement 3
c = a + b;          // Statement 4
alert(a);           // Statement 4
console.log(c);     // Statement 4
```

Scanned with CamScanner

End of Statement with semicolon ;

1. To end statement add semicolon at the end of each executable statement
2. Semicolons separate JavaScript statements.

```
var a = 4;
```

3. When separated by semicolons, multiple statements on one line are allowed

```
i = 3; j = 5; k = i + j;
alert(i);console.log(k);
```

Scanned with CamScanner

End of Statement without semicolon ;

1. Typically we end statements with semicolon but JavaScript semicolon is optional, but is highly recommended to end with semicolon
2. The end of a statement is most often marked by pressing enter and starting a new line.

Scanned with CamScanner

End of Statement without semicolon ;

```
var a = 5                // New line will end statement
a * 4
alert(a)
console.log(a)


var a = 5a * 4              // Error, Will Not work
alert(a)console.log(a)      // Error, Will not work
```

Scanned with CamScanner

Expressions

1. An expression is a combination of values, variables, function calls and operators, which computes to a value.
2. The computation is called an evaluation.

```
a + b;                          // expression
4 / 2;                          // expression
var a = 5;
a * 4;                          // expression
"John" + " " + "Doe";           // expression
```

Scanned with CamScanner

Comments

Scanned with CamScanner

Comments

1. Comments are for the human, not the machine.
2. They help you and others understand your code when it comes time to revise, they make code more readable.
3. You can also use commenting to comment out portions of your code for testing and debugging.
4. They will prevent execution of code
5. There are two ways mark text as a comment, single line comments and multi-line comments

Scanned with CamScanner

Single line comments

1. Single line comments start with //.
2. Any text between // and the end of the line will be ignored (will not be executed).

```
// Declare and initialize variable
var a = 6; //This is comment


//This below code will not execute
//var b = 8;
```

Scanned with CamScanner

Multi-line comments

1. Multi-line comments start with /* and end with */.
2. Any text between /* and */ will be ignored.

```
/*
This code declared and initialize
variable a and show on screen
*/
var a = 6;
alert(a);
```

Scanned with CamScanner

Variable Names

Scanned with CamScanner

Variable Names Legal and Illegal

1. A variable name can't contain any spaces
2. A variable name can contain only letters, numbers, dollar signs, and underscores.
3. The first character must be a letter, or an underscore (_), or a dollar sign ($).
4. Subsequent characters may be letters, digits, underscores, or dollar signs.
5. Numbers are not allowed as the first character of variable.

Scanned with CamScanner

Variable Names Legal and Illegal

1. Legal names:

```
var hello = 56;
var _xyz = 44;
var $work = 90;
var user2 = 56;
var i_info = 99;
var my$work = 77;
```

Scanned with CamScanner

Variable Names Legal and Illegal

1. Illegal names:

```
var 2user = 12;   // Can't start with number
var my user = 23; // Can't contains space
var hello#world = 34;
var my-info = 44;
var my?info = 45;
var my*info = 45;
```

Scanned with CamScanner

Reserved Keywords

1. Reserved Keywords cannot be used as variable name
2. Here are few reserved keywords

abstract	arguments	await*	boolean
break	byte	case	catch
char	class*	const	continue
debugger	default	delete	do
double	else	enum*	eval
export*	extends*	false	final
finally	float	for	function
goto	if	implements	import*

Scanned with CamScanner

abstract	arguments	await*	boolean
break	byte	case	catch
char	class*	const	continue
debugger	default	delete	do
double	else	enum*	eval
export*	extends*	false	final
finally	float	for	function
goto	if	implements	import*
in	instanceof	int	interface
let*	long	native	new
null	package	private	protected
public	return	short	static
super*	switch	synchronized	this
throw	throws	transient	true
try	typeof	var	void
volatile	while	with	yield

Scanned with CamScanner

Case Sensitive

1. Variable names are case sensitive.
2. So **`rose`** and **`Rose`** are two different variables

```
var rose = "Hello";
var Rose = "Hello";
alert(rose);
alert(Rose);
alert(ROSE); // Error
```

Scanned with CamScanner

Camel Case

1. If there's more than one word in the variable name, then it is recommended to use camel case
2. A camelCase name begins in lower case. If there's more than one word in the name, each subsequent word gets an initial cap, creating a hump.

Scanned with CamScanner

Camel Case

```
var userResponse
var userResponseTime
var userResponseTimeLimit
var response
```

This Style of naming a variable is called camel case

Scanned with CamScanner

Operators

Scanned with CamScanner

Arithmetic Operators

```
var a = 5;
var b = 3;
var c = a + b;        // Addition, result 8
var d = a - b;        // Subtraction, result 2
var e = a * b;        // Multiplication, result 15
var f = a / b;        // Division, result 1.66
var g = a % b;        // Modulus, result 2
var h = a ** b;       // Exponentiation, result 125
```

Scanned with CamScanner

Assignment Operators

1. Assignment operator assign value to variables
2. When you need to apply arithmetic operation and assign value to same variable then you can also use them

```
var a = 5; // equals = is assignment operator
a = a + 2; // Assign 7 in variable a
OR
var a = 5;
a+=2;      // Assign 7 in variable a
```

Scanned with CamScanner

Assignment Operators

Example	Same as
`a += 5;`	`a = a + 5;`
`a -= 5;`	`a = a - 5;`
`a *= 5;`	`a = a * 5;`
`a /= 5;`	`a = a / 5;`
`a %= 5;`	`a = a % 5;`
`a **= 5;`	`a = a ** 5;`

Scanned with CamScanner

Eliminating ambiguity -- BODMAS

1. Complex arithmetic expressions can pose a problem when there are multiple operators in single expression

```
var a = 5 + 2 * 3 - 2 / 2;  // result 10
```

2. The evaluation of above expression is depends on BODMAS rule
3.

Scanned with CamScanner

Eliminating ambiguity -- BODMAS

B	Brackets first
O	Orders (i.e. Powers and Square Roots, etc.)
DM	Division and Multiplication (left-to-right)
AS	Addition and Subtraction (left-to-right)

Scanned with CamScanner

Eliminating ambiguity -- BODMAS

```
var a = 5 + 2 * (3 - 2) / 2; // result 6
```

1. 3 - 2 with brackets will be evaluated first, result 1
2. 2 * result of (3 - 2) so 2 * 1, result 2
3. Result of 2 * (3 -2) divide by 2 so 2 / 2, result 1
4. 5 + result of 2 * (3 - 2) / 2, so 5 + 1, result 6

Scanned with CamScanner

Eliminating ambiguity -- BODMAS

```
var a = 3 + 5 * 2;                    // result 13
var b = 8 / 2 - 1;                    // result 3
var c = 3 % 2 + 4 - 1;                // result 4
var d  = a + 5 * c - b / (3 + b);// result 32.5
```

Scanned with CamScanner

Operator Precedence -- Few of them

Precedence	Operator	Associativity
1	`. []`	left-to-right
	`new`	right-to-left
2	`()`	left-to-right
3	`++ --`	not applicable
4	`! ~ -` (negation) `+` (unary plus) `typeof void delete`	right-to-left
5	`* / %`	left-to-right
6	`+ -`	left-to-right
7	`== != === !==`	left-to-right
8	`&&`	left-to-right
9	`||`	left-to-right
10	`?:`	right-to-left
11	`= += -= *= /= %=&= |= ^=`	right-to-left
12	`,`	left-to-right

Scanned with CamScanner

Increment and Decrement Operator

1. While working on application you will frequently required to increase variable by 1 or decrease variable by 1
2. For this situation you can use increment and decrement operator
3. ++ increment operator
4. -- decrement operator
5. These operators can be used as prefix and postfix

Scanned with CamScanner

Increment and Decrement Operator

1. We can increase or decrease value using existing addition and subtraction operators

```
var a = 12;
a = a + 1;          // 13
a = a - 1;          // 12
var b = 12;
b += 1;             // 13
b -= 1;             // 13
```

Scanned with CamScanner

Increment and Decrement Operator

Prefix Increment and Decrement

```
var age = 12;
++age;
alert(age); //Result 13
--age;
alert(age); //Result 12
```

Scanned with CamScanner

Increment and Decrement Operator

Postfix Increment and Decrement

```
var age = 12;
age++;
alert(age); //Result 13, same as prefix
age--;
alert(age); //Result 12, same as postfix
```

Scanned with CamScanner

Increment and Decrement Operator

1. What is the difference between Prefix and Postfix
2. You will NOT find any difference if you will not assign result of prefix and postfix to any other variable
3. Prefix operator first increase/decrease the value in variable and then assign result to other variable
4. Postfix operator first assign the value in other value then increase/decrease value in actual variable

Scanned with CamScanner

Increment and Decrement Operator

Prefix Increment

```
var age = 12;
var newAge = ++age;
alert(age);        //Result 13
alert(newAge);     //Result 13
```

Scanned with CamScanner

Increment and Decrement Operator

Postfix Increment

```
var age = 12;
var newAge = age++;
alert(age);        //Result 13
alert(newAge);     //Result 12 , see the difference
```

Scanned with CamScanner

Increment and Decrement Operator

Prefix Decrement

```
var age = 12;
var newAge = --age;
alert(age);        //Result 11
alert(newAge);     //Result 11
```

Scanned with CamScanner

Increment and Decrement Operator

Postfix Decrement

```
var age = 12;
var newAge = age--;
alert(age);        //Result 11
alert(newAge);     //Result 12 , see the difference
```

Scanned with CamScanner

Increment and Decrement Operator

Example 1

```
var a = 4;
var b = 2;
var c = a++ - b; // first value of a placed here
which is 4 then increase 1 in a so will become 5
alert(a);          // 5
alert(b);          // 2
alert(c);          // 2
```

Scanned with CamScanner

Increment and Decrement Operator

Example 2

```
var a = 4;
var b = 2;
var c = ++a + b; // first value of a increased so
will become 5 then value of a will be placed here, 5
alert(a);        // 5
alert(b);        // 2
alert(c);        // 7
```

Scanned with CamScanner

Increment and Decrement Operator

Example 3

```
var a = 4;
var b = 3;
var c = a++ + --b - --a;
alert(a);          // 4
alert(b);          // 2
alert(c);          // 2
```

Scanned with CamScanner

Increment and Decrement Operator

Example 4

```
var a = 4;
var b = 3;
var c = ++a + b++ - a + --b;
alert(a);          // 5
alert(b);          // 3
alert(c);          // 6
```

Scanned with CamScanner

String Concatenation

Scanned with CamScanner

Concatenating Text strings

1. The + operator can also be used for concatenating strings
2. E.g:

```
var firstName = "Abraham";
var lastName = "aLincoln";
//concatenate firstName, space character and lastName
var fullName = firstName + " " + lastName;
alert(fullName);
```

Scanned with CamScanner

Concatenating strings and numbers

1. Adding two numbers, will return the sum, but adding a number and a string will return a string:

```
var a = "6" + 2;                // "62"
var b = 3 + "6";                // "36"
var c = "Hello " + 2;           // "Hello 2"
var d = 2 + "Hello ";           // "2 Hello"
var e = "Hello "+ 3 + 4;        // "Hello 34"
var f = 3 + 4 + "Hello ";       // "7Hello"
var g = "Hello "+ (3 + 4);      // "Hello 7"
```

Scanned with CamScanner

Prompt and Parsing String

Scanned with CamScanner

prompt or window.prompt()

1. The prompt()/window.prompt() method displays a dialog box that prompts the visitor for input.
2. In a prompt, you can specify a second string. This is the default response that appears in the field when the prompt displays.
3. The prompt() method returns the input value if the user clicks "OK". If the user clicks "cancel" the method returns null.

Scanned with CamScanner

prompt or window.prompt()

```
var question = "What is your name?";
var defaultAnswer = "John";
var name = prompt(question, defaultAnswer);
console.log("Name = "+ name);
```

Scanned with CamScanner

prompt or window.prompt()

1. If you ask for number in prompt and try apply addition operator on it then it will concatenate the value because prompt returns string

```
var question = "What is your age?";
var age = prompt(question);// Assuming input 12
var newAge = age + 5;       // It will concatenate
console.log("New Age = "+ newAge); //result 125
```

Scanned with CamScanner

Convert string to integer

1. Addition operator in string concatenate values, even if value in string is number

```
var value1 = "3";
var value2 = "5";
var value3 = value1+value2;
console.log(value3);// result, 35
```

Scanned with CamScanner

Convert string to integer

1. prompt function also returns string even if you provide number in input box

```
//Assuming we will provide value 5 in input
var age = prompt("What is your age");
var num = 4;
var sum = age + num;
console.log(sum); // result, 54
```

Scanned with CamScanner

Convert string to integer

1. If you are sure that string has number, then you have to convert it into number to perform addition
2. We need to use **parseInt** function for conversion

```
var age = prompt("What is your age");//input 5
var num = 4;
var sum = parseInt(age) + num;
console.log(sum); // result, 9
```

Scanned with CamScanner

Convert string to integer

1. If value is not number then it will return NaN

```
var age = prompt("What is your age");//input abc
var num = 4;
var sum = parseInt(age) + num;
console.log(sum); // result, NaN
```

Scanned with CamScanner

Convert string to decimal

1. If your string is number with decimal places then you can use parseFloat for conversion

```
var age = prompt("What is your age");//input 5.5
var num = 4;
var sum = parseFloat(age) + num;
console.log(sum); // result, 9.5
```

Scanned with CamScanner

Convert string to Number

1. The Number() function converts the object argument to a number that represents the object's value.
2. If the value cannot be converted to a legal number, NaN is returned.

```
var age = prompt("What is your age");//input 5.5
var num = 4;
var sum = age + Number(num);
console.log(sum); // result, 9.5
```

Scanned with CamScanner

Convert string to Number

1. In case of Number() function you don't have to use separate parseInt or parseFloat function for conversion

```
var a = Number(true);          //returns 1
var b = Number(false);         //returns 0
var c = Number("999");         //returns 999
var d = Number("999 888");     //returns NAN
var e = Number("Hello");       //returns NAN
```

Scanned with CamScanner

Comparison Operators

Scanned with CamScanner

Comparison operators

1. Comparison operators are used in logical statements to determine equality or difference between variables or values. They will result in **true** or **false** only

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type

Operator	Description
>	greater than
<	less than
>=	greater than and equal to
<=	less than and equal to

Scanned with CamScanner

Comparison operators

```
var a = 3;
console.log(a == 6);          //return false
console.log(a === 6);         //return false
console.log(a != 6);          //return true
console.log(a !== 6);         //return true
console.log(a > 6);           //return false
console.log(a < 6);           //return true
console.log(a >= 6);          //return false
console.log(a <= 6);          //return true
```

Scanned with CamScanner

Comparison operators

1. == and === are used for equality check
2. == does not consider data type of values being compared and
3. == tries to convert one of the value and compare based on that
4. === also check datatype of the values and datatype of value on both side should be same otherwise it will return false

Scanned with CamScanner

Comparison operators

```
var a = 3;
console.log(a == 3);          //return true
console.log(a == "3");        //return true
console.log(a == 6);          //return false

console.log(a === 3);         //return true
console.log(a === "3");       //return false
console.log(a === 6);         //return false
```

Scanned with CamScanner

Comparison operators

1. Be careful JavaScript is dynamic language and many decision taken at start are still causing confusion
2. Comparing different type of values will result in answer which is not easily understandable
3. E.g: When comparing a string with a number, JavaScript will convert the string to a number while doing the comparison. An empty string converts to 0. A non-numeric string converts to NaN which is always false.

Scanned with CamScanner

Logical Operators

Scanned with CamScanner

Short-circuit Logical Operators

1. Logical operators are used to determine the logic between variables or values.
2. Logical operators required boolean operands on both side of operator
3. However, the && and || operators actually return the value of one of the specified operands, so if these operators are used with non-Boolean values, they will return a non-Boolean value.

Scanned with CamScanner

Logical Operators

1. There are three logical operators in JavaScript:
    a. || (OR)
    b. && (AND)
    c. ! (NOT)

Scanned with CamScanner

&& Logical Operator

1. This logical operator is used with two or more values (operands), and only evaluates to true if all the operands are true. It will return the false value if at least one value is false.

```
alert( true && true );    // true
alert( false && true );   // false
alert( true && false );   // false
alert( false && false ); // false
```

Scanned with CamScanner

&& Logical Operator

```
var a = 60;
var b = a > 50 && a < 70;
alert(b);        // return true


var c = 80;
var d = c > 50 && c < 70;
alert(d);        // return false
```

Scanned with CamScanner

|| Logical Operator

1. The logical operator || (OR) also is used with two or more values, but it evaluates to true if any of the operands (values) are true, so only evaluates to false if both operands are false.

```
alert( true || true );    // true
alert( false || true );   // true
alert( true || false );   // true
alert( false || false ); // false
```

Scanned with CamScanner

|| Logical Operator

```
var a = 60;
var b = a < 50 || a > 70;
alert(b);        // return false


var c = 80;
var d = c < 50 || c > 70;
alert(d);        // return true
```

Scanned with CamScanner

! Logical NOT

1. Using the ! operator in front of a boolean will convert it to opposite value. It means that a true value will return false, and a false will return true. This method is known as negation:

```
alert( !true ); // false
alert( !false ); // true
```

Scanned with CamScanner

! Logical NOT

```
var a = 60;
var b = !(a < 50);
alert(b);        // return true


var c = 80;
var d = !(c > 50);
alert(d);        // return false
```

Scanned with CamScanner

! Logical NOT

1. Using the ! operator in front of a other value will convert it to a Boolean and return an opposite value.

```
alert( !1 ); // false
alert( !0); // true
```

2. Here 1 will be converted to boolean first and it will be true and because of ! operator opposite will return
3. Same for 0, it will be converted to false and ! will negate it to true

Scanned with CamScanner

!! Double NOT

1. Using the ! operator twice in front of a value will convert it to a Boolean and negate it twice, useful when you want to convert value to boolean

```
alert( !!1 ); // true
alert( !!0); // true
```

2. Here 1 will be converted to boolean first and it will be true and because of first ! operator it will convert to false and then because of second ! operator it will be converted to true

Scanned with CamScanner

Why they are called short-circuit

1. && and || operator stops evaluation of expression once they find desired value
2. && stops evaluation as soon as it finds false and returns false
3. && returns true if all values are true
4. || stops evaluation as soon as it finds true and returns true
5. || returns false if all values are false

Scanned with CamScanner

Why they are called short-circuit

```
//returns false, evaluation stops at first value
var a = false && true && false;

//returns false, evaluation stops at last value
var b = true && true && false;

//returns false, evaluation stops at second value
var c = true && false && true;
```

Scanned with CamScanner

Why they are called short-circuit

```
//returns true, evaluation stops at second value
var d = false || true || false;

//returns true, evaluation stops at first value
var e = true || true || false;

//returns true, evaluation stops at last value
var f = false || false || true
```

Scanned with CamScanner

Conditions

Scanned with CamScanner

Conditions

1. Up until now, all the code in our programs has been executed chronologically
2. Very often when you write code, you want to perform different actions for different decisions.
3. You can use conditional statements in your code to do this.

Scanned with CamScanner

Conditions

1. In JavaScript we have the following conditional statements:
    a. Use ***if*** to specify a block of code to be executed, if a specified condition is true
    b. Use ***else*** to specify a block of code to be executed, if the same condition is false
    c. Use ***else if*** to specify a new condition to test, if the first condition is false
    d. Use ***switch*** to specify many alternative blocks of code to be executed (Discussed later)

Scanned with CamScanner

Conditions: if

1. The ***if*** statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.

```
if (condition) {
   //  block of code to be executed if the condition
is true
}
```

Scanned with CamScanner

Conditions: if

```
var age = 12;


if( age > 9 ) {
    console.log("Age = "+age);
}
```

Scanned with CamScanner

Conditions: else

1. Use the ***else*** statement to specify a block of code to be executed if the condition is false.

```
if (condition) {
   //block of code to be executed if the condition is true
} else {
   //block of code to be executed if the condition is false
}
```

Scanned with CamScanner

Conditions: else

```
var age = 15;


if( age > 18 ) {
    console.log("Qualifies for driving");
} else {
    console.log("Does not qualify for driving");
}
```

Scanned with CamScanner

Conditions: else if

1. Use the ***else if*** statement to specify a new condition if the first condition is false.

Scanned with CamScanner

Conditions: else if

```
if (condition1) {
   //block of code to be executed if condition1 is true
} else if (condition2) {
   //block of code to be executed if the condition1 is false
and condition2 is true
} else {
   //block of code to be executed if the condition1 is false
and condition2 is false
}
```

Scanned with CamScanner

Conditions: else if

```
var score = 80;
if( score > 80 ) {
    console.log("Grade A");
} else if( score > 70 ) {
    console.log("Grade B");
} else if( score > 60 ) {
    console.log("Grade C");
} else {
    console.log("Failed");
}
```

Scanned with CamScanner

Conditions: nested if

1. ***nested if*** statements means an if statement inside an if statement.

```
if (condition1) {
    //Code to be executed if the condition1 is true
    if (condition1) {
        //Code to be executed if the condition2 is true
    }
}
```

Scanned with CamScanner

Conditions: nested if

```
var score = 80;
if( score > 80 ) {
    console.log("Grade A");
    if( score > 90){
        console.log("Reward $100")
    }
}
```

Scanned with CamScanner

Set of Conditions

```
var score = 70;
if(score > 80 && score <= 100){
    console.log("Grade A");
}
else if (score > 70 && score <= 80){
    console.log("Grade B");
}
else if (score > 60 && score <= 70){
    console.log("Grade C");
}
```

Scanned with CamScanner

Set of Conditions

```
var input = "M";
if(input == "A" || input == "E" || input == "I"
          || input == "O" || input == "U"){
   console.log("It's a Vowel");
}
else {
   console.log("It's a consonants");
}
```

Scanned with CamScanner

Value conversion to boolean

1. In JavaScript value or expression can be converted to boolean
2. If you apply boolean comparison on values then JavaScript convert that value into boolean and make comparison
3. You can use values in if/else conditions and they will result in true or false according to values

Scanned with CamScanner

Value conversion to boolean

1. Following values will be converted to false
    a. null
    b. NaN
    c. 0 // Zero
    d. “” or ‘’ or `` // Empty String single or double quotes
    e. undefined
2. All else will be converted to true

Scanned with CamScanner

Value conversion to boolean

1. Following values will be converted to true
    a. {} // Object
    b. “AnyText” // String with any text
    c. 1 // Any number other than zero

Scanned with CamScanner

Number to Boolean

```
var age = 45;
// 45 will be converted to true, if will be executed
if(age){
    console.log("In If Age = "+age);
}
else{
    console.log("In else Age = "+age);
}
```

Scanned with CamScanner

null to Boolean

```
var name = null;
//null will be converted to false, else will be executed
if(name){
  console.log("In If name = "+name);
}
else{
  console.log("In else name = "+name);
}
```

Scanned with CamScanner

String to Boolean

```
var name = "Hello";
//Hello will be converted to true, if will be executed
if(name){
    console.log("In If name = "+name);
}
else{
    console.log("In else name = "+name);
}
```

Scanned with CamScanner

String to Boolean

```
var name = "";
//it will be converted to false, else will be executed
if(name){
    console.log("In If name = "+name);
}
else{
    console.log("In else name = "+name);
}
```

Scanned with CamScanner

undefined to Boolean

```
var name;  // Default value is undefined
//it will be converted to false, else will be executed
if(name){
  console.log("In If name = "+name);
}
else{
  console.log("In else name = "+name);
}
```

Scanned with CamScanner

Value conversion to boolean

1. Value conversion also works with Logical operator && and ||
2. For &&
    a. If all values are converted to true, it will returns last value
    b. If any of the value converted to false it will that value
3. For ||
    a. If all values are converted to false, it will returns last value
    b. If any value converted to true it will return that value

Scanned with CamScanner

Value conversion to boolean &&

```
var a1 = 'Cat' && 'Dog';        // t && t returns "Dog"
var a2 = false && 'Cat';        // f && t returns false
var a3 = 'Cat' && false;        // t && f returns false
var a4 = ''    && false;        // f && f returns ""
var a5 = false && '';           // f && f returns false
var a6 = 0 && 1;                // f && t returns 0
```

Scanned with CamScanner

Value conversion to boolean &&

```
var a7 = 1 && false;          // t && f returns false
var a8 = true && {};          // t && t returns {}
var a9 = false && {};         // f && t returns false
var a10 = 'Cat' && (3==4);    // t && f returns false
var a11 = 'Cat' && 0;         // t && f returns 0
var a12 = undefined && 'Cat';// f && t returns undefined
```

Scanned with CamScanner

Value conversion to boolean ||

```
var a1 = 'Cat' || 'Dog';        // t || t returns "Cat"
var a2 = false || 'Cat';        // f || t returns "Cat"
var a3 = 'Cat' || false;        // t || f returns "Cat"
var a4 = ''    || false;        // f || f returns false
var a5 = false || '';           // f || f returns ""
var a6 = 0 || 1;                // f || t returns 1
```

Scanned with CamScanner

Value conversion to boolean ||

```
var a7 = 1 || false;              // t || f returns 1
var a8 = true || {};              // t || t returns true
var a9 = false || {};             // f || t returns {}
var a10 = 'Cat' || (3 == 4);      // t || f returns "Cat"
var a11 = 'Cat' || 0;             // t || f returns "Cat"
var a12 = undefined || 'Cat';     // f || t returns "Cat"
```

Scanned with CamScanner

For Loop

Scanned with CamScanner

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.

Scanned with CamScanner

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.

Scanned with CamScanner

For Loop

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

Output:

0

1

2

Scanned with CamScanner

For Loop

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

Output:

5

6

7

8

Scanned with CamScanner

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");
}
```

Scanned with CamScanner

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");
```

Scanned with CamScanner

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

Scanned with CamScanner

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

Scanned with CamScanner

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

Scanned with CamScanner

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

Scanned with CamScanner

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).

Scanned with CamScanner

Task

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

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

Scanned with CamScanner

Arrays

Scanned with CamScanner

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

Scanned with CamScanner

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;
```

Scanned with CamScanner

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

Scanned with CamScanner

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

Scanned with CamScanner

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];
```

Scanned with CamScanner

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

Scanned with CamScanner

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"}];
```

Scanned with CamScanner

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
```

Scanned with CamScanner

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
```

Scanned with CamScanner

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

Scanned with CamScanner

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
```

Scanned with CamScanner

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
```

Scanned with CamScanner

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
```

Scanned with CamScanner

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
```

Scanned with CamScanner

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
```

Scanned with CamScanner

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");
```

Scanned with CamScanner

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
```

Scanned with CamScanner

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
```

Scanned with CamScanner

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)

Scanned with CamScanner

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
```

Scanned with CamScanner

Stack

Last in First out

(LIFO)

Scanned with CamScanner

Stack (Last in First out)

1. Stack is a linear data structure represented by a real physical stack or pile where insertion and deletion of items takes place at one end called top of the stack.
2. The basic concept can be illustrated by thinking of your data set as a stack of plates or books where you can only take the top item off the stack in order to remove things from it
3. The basic implementation of a stack is also called a LIFO (Last In First Out)

Scanned with CamScanner

Stack (Last in First out)

1. To behave as stack you have to use push and pop function on array
2. **Push** function will add element at the **end** in array
3. **Pop** function will remove and return **last** elements from array

Scanned with CamScanner

Stack (Last in First out)

```
var foods = [];
foods.push("Pizza");
foods.push("Burger");
foods.push("Snacks");
console.log("Length "+foods.length); // Length 3
console.log(foods.pop()); // Remove Snacks from array
console.log(foods.pop()); // Remove Burger from array
console.log("Length "+foods.length); // Length 1
```

Scanned with CamScanner

Queue

First in First out

(FIFO)

Scanned with CamScanner

Queue (First in First out)

1. Queue is a linear data structure represented by a real physical queue.
2. You can think of it as a line in a grocery store or banks
3. A queue is open at both its ends.
4. The basic implementation of a stack is also called a FIFO (First-In-First-Out)

Scanned with CamScanner

Queue (First in First out)

1. To behave as queue you have to use push and shift function on array
2. **Push** function will add element at the **end** in array
3. **Shift** function will remove and return **first** elements from array

Scanned with CamScanner

Queue (First in First out)

```
var foods = [];
foods.push("Pizza");
foods.push("Burger");
foods.push("Snacks");
console.log("Length "+foods.length); // Length 3
console.log(foods.shift());// Remove Pizza from array
console.log(foods.shift());// Remove Burger from array
console.log("Length "+foods.length); // Length 1
```

Scanned with CamScanner

Unshift function

1. To add element in array we use push function or index
2. These will add element at end of array
3. If you want to add element at the start of array and move all element one index ahead then we can we use unshift function

Scanned with CamScanner

Unshift function

```
var foods = [];
foods.push("Pizza");
foods.push("Burger");
foods.push("Snacks");
console.log("Length "+foods.length); // Length 3
console.log(foods[0]); // "Pizza"
foods.unshift("Sandwich");
console.log(foods[0]); // "Sandwich"
console.log(foods[1]); // "Pizza"
```

Scanned with CamScanner

Iterating array with Loops

1. To access all element in array we can use loops to iterate over each array element

```
var foods = ["Pizza", "Burger", "Snacks"];
for(var i=0;i<foods.length;i++){
   alert(foods[i]);
}
```

2. ***i*** will work as index and this code will show alert 3 times each with “Pizza”, “Burger” and “Snacks” respectively

Scanned with CamScanner

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

Scanned with CamScanner

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

Scanned with CamScanner

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

Scanned with CamScanner

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

Scanned with CamScanner

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

Scanned with CamScanner

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

Scanned with CamScanner

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

Scanned with CamScanner

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

Scanned with CamScanner

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"
```

Scanned with CamScanner

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"
```

Scanned with CamScanner

Other Array functions

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

And others

Scanned with CamScanner

String

Scanned with CamScanner

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
```

Scanned with CamScanner

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
```

Scanned with CamScanner

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
```

Scanned with CamScanner

String functions

Scanned with CamScanner

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()

Scanned with CamScanner

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
```

Scanned with CamScanner

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
```

Scanned with CamScanner

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...
```

Scanned with CamScanner

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
```

Scanned with CamScanner

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
```

Scanned with CamScanner

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
```

Scanned with CamScanner

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
```

Scanned with CamScanner

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
```

Scanned with CamScanner

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
```

Scanned with CamScanner

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

Scanned with CamScanner

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
```

Scanned with CamScanner

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"
```

Scanned with CamScanner

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"
```

Scanned with CamScanner

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"
```

Scanned with CamScanner

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"
```

Scanned with CamScanner

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"]
```

Scanned with CamScanner

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
```

Scanned with CamScanner

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()

Scanned with CamScanner

Math functions

Scanned with CamScanner

Math

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

Scanned with CamScanner

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);
```

Scanned with CamScanner

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
```

Scanned with CamScanner

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
```

Scanned with CamScanner

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
```

Scanned with CamScanner

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

Scanned with CamScanner

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
```

Scanned with CamScanner

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()

Scanned with CamScanner

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
```

Scanned with CamScanner

Date Object

Scanned with CamScanner

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

Scanned with CamScanner

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().

Scanned with CamScanner

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

Scanned with CamScanner

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

Scanned with CamScanner

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)
```

Scanned with CamScanner

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");
```

Scanned with CamScanner

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
```

Scanned with CamScanner

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)
```

Scanned with CamScanner

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.

Scanned with CamScanner

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	

Scanned with CamScanner

Retrieving the Date Components

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

`date.getFullYear();// 2019`	`date.getMinutes();  // 45`
`date.getMonth();  // 5`	`date.getSeconds();  // 25`
`date.getDate();  // 14`	`date.getMilliseconds();//0`
`date.getDay();  // 5`	`date.getTime();`
`date.getHours();  // 10`	`//1560491125000`

Scanned with CamScanner

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

Scanned with CamScanner

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	

Scanned with CamScanner

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)
```

Scanned with CamScanner

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

Scanned with CamScanner

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
```

Scanned with CamScanner

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

Scanned with CamScanner

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
```

Scanned with CamScanner

Functions

Scanned with CamScanner

Functions

1. A function is a block of JavaScript that does the same thing again and again.
2. A JavaScript function is executed when "something" invokes it (calls it).
3. It saves you repetitive coding and makes your code easier to understand.
4. You can reuse code: Define the code once, and use it many times

Scanned with CamScanner

Function Declarations

1. A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses ().
2. Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).
3. The parentheses may include parameter names separated by commas: (parameter1, parameter2, ...)
4. The code to be executed, by the function, is placed inside curly brackets: {}

Scanned with CamScanner

Function Declarations

```
function name(parameter1, parameter2) {
    // code to be executed
}
```

Scanned with CamScanner

Function Declarations

```
function sum(a, b){
    return a + b;
}
```

Declared functions are not executed immediately. They are "saved for later use", and will be executed later, when they are invoked (called upon).

Scanned with CamScanner

Invoking a Function

1. Functions execute when the function is called.
2. This process is known as invocation.
3. You can invoke a function by referencing the function name, followed by an open and closed parenthesis: ().

Scanned with CamScanner

Invoking a Function

1. Declarations

```
function showMessage(message){
    console.log(message);
}
```

2. Invoking

```
showMessage("Hello World");
```

Scanned with CamScanner

Parameters vs. Arguments

1. Parameters:
    a. Function parameters are listed inside the parentheses () in the function definition.
2. Arguments:
    a. Function arguments are the values received by the function when it is invoked.

Scanned with CamScanner

Parameters vs. Arguments

1. Declarations

Parameter

```
function showMessage(message){
    console.log(message);
}
```

2. Invoking

Argument

```
showMessage("Hello World");
```

Scanned with CamScanner

Passing Data to Function

1. In order for a function to become a programmable robot rather than a one-job robot, you have to set it up to receive the data you're passing.
2. You can pass any type of data to function depending on requirement

Scanned with CamScanner

Passing Data to Function

```
function multiply(num1, num2){
    var num3 = num1 * num2;
    console.log("Num3 ",num3);
}

multiply(3,6);
multiply(4,2);
```

Scanned with CamScanner

Passing Data to Function

```
function showMessage(name){
    console.log("Hello "+name);
}

showMessage("Mike");
showMessage("John");
```

Scanned with CamScanner

Parameter Rules

1. JavaScript function definitions do not specify data types for parameters.
2. JavaScript functions do not perform type checking on the passed arguments.
3. JavaScript functions do not check the number of arguments received.
4. If a function is called with missing arguments (less than declared), the missing values are set to: ***undefined***

Scanned with CamScanner

Parameter Rules

```
function showMessage(name){
    console.log("Hello "+name);
}
showMessage("John");     // Hello John
showMessage(45);         // Hello 45
showMessage(true);       // Hello true
showMessage();           // Hello undefined
showMessage("Mike", 12);// Hello Mike
```

Scanned with CamScanner

Function Return

1. Function can returns data back to caller
2. After executing logic in function if you want to return result to the caller of function then you use ***return*** keyword
3. When JavaScript reaches a ***return*** statement, the function will stop executing and return value is "returned" back to the "caller"
4. Every function in JavaScript returns ***undefined*** unless otherwise specified

Scanned with CamScanner

Function Return

```
function test(){


}


var a = test(); // return undefined
console.log(a); // undefined
```

Scanned with CamScanner

Function Return

In this example we explicitly tell the function to return 45

```
function test(){
    return 45;
}
var a = test(); // return 45
console.log(a); // 45
```

Scanned with CamScanner

Function Return

```
function multiply(num1, num2){
   return num1 * num2;
}
var a = multiply(3,6);  // returns 18
var b = multiply(4,2);  // returns 8
console.log(a);         // 18
console.log(b);         // 8
console.log(multiply(2,5)); // 10
```

Scanned with CamScanner

Function Return

```
function multiply(num1, num2){
    return num2;        // function execution ends here
    return num1 * num2;
}
var a = multiply(3,6);  // returns 6
console.log(a);         // 6
```

Scanned with CamScanner

Function in Expressions

1. JavaScript functions can be used in expressions
2. Just like we use variables in calculation we can use function and output function will be included in calculation

```
function multiply(num1, num2){
    return num1 * num2;
}
var a = multiply(3,4) + 5
```

Scanned with CamScanner

Function in Expressions

```
function multiply(num1, num2){
    return num1 * num2;
}
function sum(a, b){
    // Result of multiply sum with value of b
    return multiply(a,b) + b;   //16
}
var total = sum(3,4) + 6; // result 22
```

Scanned with CamScanner

Function in Expressions

```
function multiply(num1, num2){
    return num1 * num2;
}
function sum(a, b){
    return a + b;
}
// Call multiply first and result passed to sum
var total = sum(multiply(3,4), 2) + 6; // result 20
```

Scanned with CamScanner

Local vs Global Variables

1. Variables can have local or global scope
2. A global variable is one that's declared in the main body of your code, not inside a function.
3. A local variable is one that's declared inside a function.
4. A local variable can be either a parameter of the function, which is declared implicitly by being named as a parameter, or a variable declared explicitly in the function with the var keyword.

Scanned with CamScanner

Local vs Global Variables

5. Global variable is meaningful in every section of your code, whether that code is in the main body or in any of the functions.
6. Local variable is one that's meaningful only within the function that declares it.

Scanned with CamScanner

Local vs Global Variables

7. there are two differences between global and local variables—where they're declared, and where they're known and can be used.

Global Variables	Local Variables
Declared in the main code	**Declared in a function**
Known everywhere, useable everywhere	**Known only inside the function, usable only inside the function**

Scanned with CamScanner

Local vs Global Variables

```
var a = 7;                  // Global Variable
function sum(){
    var b = 6;              // Local Variable
    var c = a + b;          // 13, Accessing global
    console.log("C "+c);
}
sum();
console.log("A = "+a);  // 7
```

Scanned with CamScanner

Local vs Global Variables

```
var a = 7;                    // Global Variable
function sum(){
    var b = 6;                // Local Variable
    a = b + 5;
    console.log("A "+a);// Accessing global variable
}
sum();
console.log("A = "+a);  // 11, value of a updated
```

Scanned with CamScanner

Local vs Global Variables

```
var a = 7;                    // Global Variable
function sum(){
   var b = 6;                // Local Variable
   a = b + 5;
}
sum();
console.log("B = "+b);
// error, b is not available outside sum function
```

Scanned with CamScanner

Local vs Global Variables

```
var a = 7;                    // Global Variable
function sum(){
  var a = 6;    // Local Variable a same name as global
  a = 3 + 2;    // Local a variable will be affected
  console.log("A "+a);//5, access local variable
}
sum();
console.log("A "+a);//7, access global variable
```

Scanned with CamScanner

Global Variables without var keyword

1. If you create variable without ***var*** keyword it will be global variable not matter where you have created it
2. Variable created without ***var*** inside function it be global variable
3. Variable created without ***var*** in main body/code will be global variable
4. It is recommended to use variable with ***var*** keyword to have defined context

Scanned with CamScanner

Global Variables without var keyword

```
a = 7;      // Without var still Global Variable
function sum(){
   var b = 6;            // Local Variable
   a = b + 5;
   console.log("A "+a);// Accessing global variable
}
sum();
console.log("A "+a);  // 11, value of a updated
```

Scanned with CamScanner

Global Variables without var keyword

```
a = 7;      // Without var still Global Variable
function sum(){
   b = 6;  // Global variable because its without var
   a = b + 5;
   console.log("A "+a);// Accessing global variable
}
sum();
console.log("B "+b);//b available outside of function
```

Scanned with CamScanner

Function Expressions

1. A JavaScript function can also be defined using an expression.
2. A function expression can be stored in a variable

```
var sum = function (a, b){  // function as expression
    return a + b;
};
var c = sum(4,5);
console.log(c);
```

Scanned with CamScanner

Function Expressions

1. After a function expression has been stored in a variable, the variable can be used as a function
2. The function in expression is actually an anonymous function (a function without a name).
3. Functions stored in variables do not need function names. They are always invoked (called) using the variable name.

Scanned with CamScanner

Function Expressions

```
var square = function(num) { return num * num; };
var b = square(4); // 16
```

Notice that function above ends with semicolon because it is part of executable statement

Scanned with CamScanner

Function Hoisting

1. Hoisting is JavaScript's default behavior of moving declarations to the top of the current scope.
2. Hoisting applies to variable declarations and to function declarations.
3. Because of this, JavaScript functions can be called before they are declared

Scanned with CamScanner

Function Hoisting

```
var total = sum(5,6); //Calling before declaration
console.log("Sum = "+total);

function sum(a, b){
    return a + b;
}
```

Scanned with CamScanner

Arguments Passed by Value

1. Primitive data is passed by value: The function only gets to know the values, not the argument's locations.
2. If a function changes an argument's value, it does not change the parameter's original value.
3. Changes to arguments are not visible (reflected) outside the function.

Scanned with CamScanner

Arguments Passed by Value

```
var num = 5;
function changeValue(a){
    a = 7;  // change to a will not affect num
}

changeValue(num);
console.log(num);//5, num will be be updated
```

Scanned with CamScanner

Arguments Passed by Reference

1. In JavaScript, object references are values.
2. Non-primitive value such as Array or a user-defined object are passed by reference
3. If function changes the object's properties, that change is visible outside the function

Scanned with CamScanner

Arguments Passed by Reference

```
var arr = [4, 6, 7, 9];
function updateArray(val){  // array received in val
    val[1] = 57;  // updating val will also update arr
}
console.log(arr[1]);     // 6 before calling function
updateArray(arr);
console.log(arr[1]);     // 57 after calling function
```

Scanned with CamScanner

Arguments Passed by Reference

```
var obj = { name: "John", age:56 };
function updateObject(val){ // object received in val
    val.age = 40; // updating val will also update arr
}
console.log(obj.age);  // 56 before calling function
updateObject(obj);
console.log(obj.age);  // 40 after calling function
```

Scanned with CamScanner

Recursive Function

1. A recursive function is a function that calls itself.
2. Recursion is a technique for iterating over an operation by having a function call itself repeatedly until it arrives at a result.
3. In some ways, recursion is analogous to a loop. Both execute the same code multiple times, and both require a condition (to avoid an infinite loop, or rather, infinite recursion in this case)

Scanned with CamScanner

Recursive Function

1. The classic example of a function where recursion can be applied is the factorial.
2. Factorial of a number n can be defined as product of all positive numbers less than or equal to n.
3. It is the multiplying sequence of numbers in a descending order till 1. It is defined by the symbol of exclamation (!).
4. E.g factorial of 6 is
    a. $6 \times 5 \times 4 \times 3 \times 2 \times 1 = 720$

Scanned with CamScanner

Recursive Function

```
function factorial(n) {
    if (n <= 1) {
    // Recursion will stop when this condition match
        return 1;
    } else {
        return n * factorial(n - 1); // calling itself
    }
}
```

Scanned with CamScanner

Switch Statement

1. The switch statement executes a block of code depending on different cases.
2. The switch statement is a part of JavaScript's "Conditional" Statements, which are used to perform different actions based on different conditions.
3. Switch statement works for equality checks only, you can not apply range, greater than or less then checks

Scanned with CamScanner

Switch Statement

4. This is how it works:
    a. The switch statement evaluates an expression.
    b. The value of the expression is then compared with the values of each case in the structure.
    c. If there is a match, the associated block of code is executed.
5. The switch has one or more case blocks and an optional default.

Scanned with CamScanner

Switch Syntax

```
switch(expression) {
    case 'value1':  // same as if (expression === 'value1')
        // code block
        break;
    case 'value2':
        // code block
        break;
    default:
        // code block
}
```

Scanned with CamScanner

Switch Statement

```
var day = 3;
switch (day) {
    case 6:
        console.log("Today is Saturday");
        break;
    case 0:
        console.log("Today is Sunday");
        break;
    default:
         console.log("Looking forward to the Weekend");
}
```

Scanned with CamScanner

Switch – Grouping of case

1. Sometimes you will want different cases to use the same code, or fall-through to a common default.
2. If you skip the break and expression match to the case where there is no break then it will also fall-through the next case

Scanned with CamScanner

Switch – Grouping of case

```
var day = 3;
switch (day) {
    case 6: // No break
        console.log("Today is Saturday");
    case 0: // No break in last case, both will execute
       console.log("Today is Sunday");
       break;
    default:
        console.log("Looking forward to the Weekend");
}
```

Scanned with CamScanner

Switch – Grouping of case

```
var day = 3;
switch (day) {
    case 6:
    case 0:
        console.log("Yaaaa! It's Weekend");
        break;
    default:
         console.log("Looking forward to the Weekend");
}
```

Scanned with CamScanner

Switch - Strict Comparison

1. Switch cases use strict comparison (===).
2. The values must be of the same type to match.
3. A strict comparison can only be true if the operands are of the same type.

Scanned with CamScanner

Switch - Strict Comparison

```
var x = "0";
switch (x) {
    case 0:
        console.log("Off");
        break;
    case 1:
        console.log("On");
        break;
    default:    // this will execute as value did not match
        console.log("No value found");
}
```

Scanned with CamScanner

While loop

1. The while loop loops through a block of code as long as a specified condition is true.

Syntax:

```
while (condition) {
    // code block to be executed
}
```

Scanned with CamScanner

While loop

1. In this example, the code in the loop will run, over and over again, as long as a variable (i) is less than 10:

```
while (i < 10) {
    console.log("I " + i);
    i++;
}
```

2. You will use while loop when execution is dependent on user input

Scanned with CamScanner

Do/While loop

1. The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.
2. You will use do/while loop when execution is dependent on user input but it needs to run code block at least once

Scanned with CamScanner

Do/While loop

Syntax:

```
do {
    // code block to be executed
}
while (condition);
```

Scanned with CamScanner

Do/While loop

```
do {
    console.log("I " + i);
    i++;
}
while (i < 10);
```

Scanned with CamScanner

Part 2 Ends

Scanned with CamScanner