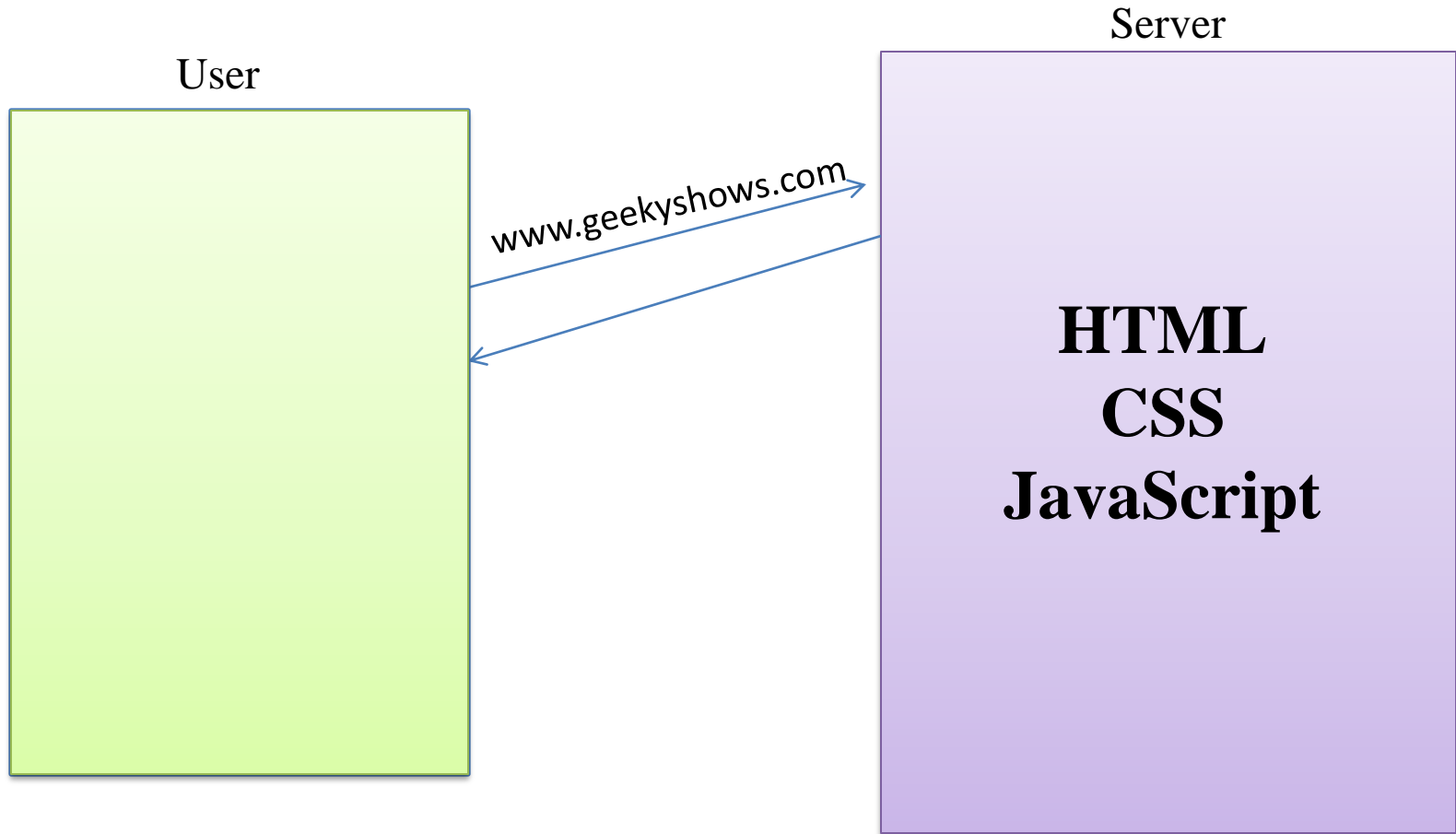JavaScript

Do You Know?

- HTML

- CSS

What is JavaScript ?

JavaScript is the programming language of HTML and the Web. It makes web page dynamic. It is an interpreted programming language with object-oriented capabilities.

JavaScript History

- 1995 by Brendan Eich (NetScape)

- Mocha

- LiveScript

- JavaScript

- ECMAScript

User

Server

www.geekyshows.com

**HTML
CSS
JavaScript**

Tools

- Notepad
- Notepad ++
- Any Text Editor

JavaScript and Java Same?

Advantage of JavaScript

- Client Side Execution

- Validation on Browser

- Easy Language

Disadvantage of JavaScript

- Less Secure
- No Hardware Access
- JavaScript Enable Browsers

Way of adding JavaScript

- Inline
  - Inside head Tag
  - Inside body Tag

- External file
  - Inside head Tag
  - Inside body Tag

<script>

</script>

<u>Inline</u>

- **Inside head Tag**

```
<html>
        <head><title>Hello JS</title>
                <script type="text/javascript">
                        document.write("Hello Geekyshows");
                </script>
        </head>
        <body>
            <h1>I am Heading</h1>
            <p>I am first Paragraph.</p>
        </body>
</html>
```

Inline

- **Inside body Tag**

<html>

                                               <head><title>Hello JS</title></head>

                                               <body>

                                                        <h1>I am Heading</h1>

                                                        <p>I am first Paragraph.</p>

                                                        <script type="text/javascript">

                                                            document.write("Hello Geekyshows");

                                                        </script>

                                               </body>

</html>

External

- **Inside head Tag**

<html>

    <head><title>Hello JS</title>

        <script type="text/javascript">
        </script>

    </head>
    <body>

        <h1>I am Heading</h1>

        <p>I am first Paragraph.</p>

    </body>

</html>

document.write("Hello Geekyshows");

- Save with .js extension
        Ex: - geek.js
- Now link this file to HTML

External

- **Inside body Tag**

```
<html>
        <head><title>Hello JS</title></head>
        <body>
            <h1>I am Heading</h1>
            <p>I am first Paragraph.</p>
            <script type="text/javascript" src="geek.js" type="text/javascript">
            </script>document.write("Hello Geekyshows");
            </script>
        </body>
</html>
```

document.write("Hello Geekyshows");

- Save with .js extension
    Ex: - geek.js
- Now link this file to HTML

```
<script type="text/javascript">          <script src = "geek.js" type="text/javascript">
    document.write("Hello World");
</script>                                 </script>
```

- <script> - Opening Script Tag.
- src – It's an attribute of script tag. It defines source/location of script file.
- geek.js – This is our script file. Where geek is file name and .js is the extension of javascript file.
- type – It's an attribute of script tag which tells the browser it is a javascript. This is optional now a days.
- text/javascript – Its type of document
- document.write("Hello World"); - This is a function to display data.
- </script> - Closing Script tag

```
<html>
        <head><title>Hello JS</title></head>
        <body>
            <h1>I am Heading</h1>
             <p>I am first Paragraph.</p>

                <script src = "geek1.js" type="text/javascript"></script>
                <script src = "geek2.js" type="text/javascript"></script>
                <script src = "geek3.js" type="text/javascript"></script>

        </body>
</html>
```

```html
<html>
        <head><title>Hello JS</title></head>
        <body>
            <h1>I am Heading</h1>
             <p>I am first Paragraph.</p>


            <script type="text/javascript">
            <script src = "geek1.js" type="text/javascript">
                document.write("Hello world");
                document.write("Hello world");
            </script>
            </script>

            <script src = "geek1.js" type="text/javascript"> </script>

        </body>
</html>
```

Display

- document.write( )
- window.alert( )
- console.log( )
- innerHTML

document.write( )

This function is used to write arbitrary HTML and content into page. If we use this function after an HTML document is fully loaded, will delete all existing HTML. It is used only for testing purpose.

Ex: - document.write("Hello World");

    document.write(variable);

    document.write(4+2);

    document.write("Hello World.
");

    document.write("Hello World.
" + variable + "<br>");

window.alert( )

This function is used to display data in alert dialog box. alert really should be used only when you truly want to stop everything and let the user know something.

Ex: - window.alert("Hello World");

      window.alert(variable);

      window.alert(4+2);

      window.alert("Hello World" + variable);

console.log( )

This function is used to display data in console. This is used for debugging purpose. We can identify our code's error.

Ex: - console.log("Hello World");

console.log(variable);

console.log(4+2);

console.log("Hello World" + variable);

Identifier

An identifier is a name having a few letters, numbers and special characters _ (underscore). It is used to identify a variable, function, symbolic constant and so on.

Ex : -

      X2
      PI
      Sigma
      matadd

Variables

A variable is an identifier or a name which is used to refer a value. A variable is written with a combination of letters, numbers and special characters _ (underscore) and $ (dollar) with the first letter being an alphabet.
Ex: c, fact, b33, total_amount etc.

<u>Rules</u>

- Variable can contain combination of letters, digits, underscores (_), and dollar signs ($).

- Must begin with a letter A-Z or a-z or underscore or dollar sign

- A variable name cannot start with a number

- Must not contain any space characters

- JavaScript is case-sensitive

- Can't use reserved keywords
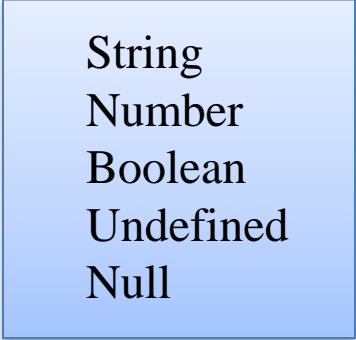
Keywords or Reserved Words

var	delete	for	let	break
super	void	case	do	static
function	new	switch	while	interface
catch	else	if	package	finally
this	with	class	enum	default
implements	private	throw	yield	typeof
const	export	import	protected	return
true	continue	extends	in	instanceof
public	try	debugger	false	

In JavaScript we do not need to specify type of the variable because it is dynamically used by JavaScript engine.

We can use **var** data type. It can hold any type of data like String, Number, Boolean, etc.

Primitive Data Type

String
Number
Boolean
Undefined
Null

Non-Primitive Data type

Object
Array
RegExp

Declaring Variable

- Variable can contain combination of letters, digits, underscores (_), and dollar signs ($).

- Must begin with a letter A-Z or a-z or underscore or dollar sign

- A variable name cannot start with a number

- Must not contain any space characters

- JavaScript is case-sensitive

- Can't use reserved keywords

name
Cab125
New_delhi
Rup$

58show
Steel City
var

Declaring Variable

var roll;

var name;

var price;

These all are undefined

A variable declared without a value will have the value undefined.

Initializing Variable

var roll;
roll = 101;

var roll = 101;

roll = 101;

var name;
name = "geeky shows";

var name = "geeky shows";

name = "geeky shows";

var price;
price = 125.36;

var price = 125.36;

price = 125.36;

- Strings are written inside double or single quotes.
- Numbers are written without quotes.
- If you put a number in quotes, it will be treated as a text string.

Initializing Variable

var ans = true;

var result = false;

Initializing Variable

var x = 10, y = 20, c = 30;

var fname = "Geeky", lname = "Shows";

var name = "Geeky Shows", roll = 101;

var name = "Geeky Shows",

    roll = 101,

    address = "Steel City";

Re-Declaring Variable

If you re-declare a JavaScript variable, it will not lose its value.

var name = "Geeky Shows";

var name;

document.write(name);

Geeky Shows

- The statements are executed, one by one, in the same order as they are written.
- JavaScript programs (and JavaScript statements) are often called JavaScript code.
- Semicolons separate JavaScript statements.
- Ending statements with semicolon is not required, but highly recommended.
- JavaScript ignores multiple spaces.
- Use line Break (Enter Key).

Comments

- Single Line Comment

- Multi Line Comment

Single Line Comment

Single line comments start with **//**.

Text between **//** and the end of the line will be ignored by JavaScript.

// you can assign any type of value

var imvalue = 101;

var imvalue = 101;   // assign any type of value

Multi Line Comment

Multi-line comments start with /* and end with */.

Any text between /* and */ will be ignored by JavaScript.

Ex: -

/* Comment Here */

Adding // in front of a code line changes the code lines from an executable line to a comment.

```
var imvalue = 101;
```

```
// var imvalue = 101;
```

JavaScript Operators

- Arithmetic Operators

- Comparison (Relational) Operators

- Logical Operators

- Bitwise Operators

- Assignment Operators

Arithmetic Operators

Operators	Meaning	Example	Result
+	Addition	4+2	6
-	Subtraction	4-2	2
*	Multiplication	4*2	8
/	Division	4/2	2
%	Modulus operator to get remainder in integer division	5%2	1
+ +	Increment	A = 10; A+ +	11
− −	Decrement	A = 10; A− −	9

Relational Operators

Operators	Meaning	Example	Result
<	Less than	5<2	False
>	Greater than	5>2	True
<=	Less than or equal to	5<=2	False
>=	Greater than or equal to	5>=2	True
==	Equal to	5==2	False
! =	Not equal to	5! =2	True
===	Equal value and same type	5 === 5	True
		5 === "5"	False
! ==	Not Equal value or Not same type	5 ! == 5	False
		5 ! == "5"	True

Logical Operators

Operator	Meaning	Example	Result
&&	Logical and	(5<2)&&(5>3)	False
||	Logical or	(5<2)||(5>3)	True
!	Logical not	!(5<2)	True

& &

Operand 1	Operand 2	Result
True	True	True
True	False	False
False	True	False
False	False	False

||

Operand 1	Operand 2	Result
True	True	True
True	False	True
False	True	True
False	False	False

!

Operand	Result
False	True
True	False

Bitwise Operators

Operator	Meaning
<<	Shifts the bits to left
>>	Shifts the bits to right
~	Bitwise inversion (one's complement)
&	Bitwise logical AND
|	Bitwise logical OR
^	Bitwise exclusive or

Bitwise logical AND &

Operand 1	Operand 2	Result (operand1&operand2)
True	True	True
True	False	False
False	True	False
False	False	False

Operand 1	Operand 2	Result (operand1&operand2)
1	1	1
1	0	0
0	1	0
0	0	0

Bitwise logical OR |

Operand 1	Operand 2	Result (operand1|operand2)
True 1	True 1	True 1
True 1	False 0	True 1
False 0	True 1	True 1
False 0	False 0	False 0

Bitwise XOR ^

Operand 1	Operand 2	Result (operand1^operand2)
True 1	True 1	False 0
True 1	False 0	True 1
False 0	True 1	True 1
False 0	False 0	False 0

Bitwise NOT ~

Operand	Result
True 1	False 0
False 0	True 1

Assignment Operators

Operator	Example	Equivalent Expression
=	$m\ =\ 10$	$m\ =\ 10$
+=	$m\ +=\ 10$	$m\ =\ m+10$
$-=$	$m\ -=\ 10$	$m\ =\ m-10$
$*=$	$m\ *=\ 10$	$m\ =\ m*10$
$/=$	$m\ /=$	$m\ =\ m/10$
$\%=$	$m\ \%=\ 10$	$m\ =\ m\%10$
$<<=$	$a<<=b$	$a\ =\ a<<b$
$>>=$	$a>>=b$	$a\ =\ a\ >>\ b$
$>>>=$	$a\ >>>=\ b$	$a\ =\ a\ >>>\ b$
$\&=$	$a\ \&=\ b$	$a\ =\ a\ \&\ b$
$\wedge=$	$a\ \wedge=\ b$	$a\ =\ a\ \wedge\ b$
$\|=$	$a\ \|=\ b$	$a\ =\ a\ \|\ b$

Getting input from user

prompt( ) – The browser provides a built-in function which can be used to get input from the user, named prompt. The prompt( ) method displays a dialog box that prompts the visitor for input.

Once the prompt function obtains input from the user, it returns that input.

Syntax: - prompt(text, defaultText)

Ex:- prompt("Enter Your Name: ", "name");

prompt("Enter Your Roll No. : ");

Good Approach

- Inline

- External

geek.js

```
document.write("Hello");
document.write("Hello");
document.write("Hello");
document.write("Hello");
document.write("Hello");
document.write("Hello");
```

```html
<html>
    <head><title>Geeky Shows</title>
        <style>
            p{color: red;}
            h1{color: #F0E68C;}
        </style>
        <script>
            document.write("Hello");
            document.write("Hello");
            document.write("Hello");
            document.write("Hello");
            document.write("Hello");
            document.write("Hello");
        </script>
    </head>
    <body>
        <h1>I am Heading</h1>
        <p>I am paragraph</p>

    </body>
</html>
```

```html
<html>
    <head><title>Geeky Shows</title>
        <style>
            p{color: red;}
            h1{color: #F0E68C;}
        </style>
        <script src = "geek.js>
        </script>
    </head>
    <body>
        <h1>I am Heading</h1>
        <p>I am paragraph</p>

    </body>
</html>
```

Good Approach

- Inside head Tag

- Inside body Tag

```
<html>
    <head><title>Geeky Shows</title>
        <style>
            p{color: red;}
            h1{color: #F0E68C;}
        </style>
        <script>
        document.write("Hello");
        document.write("Hello");
        document.write("Hello");
        document.write("Hello");
        document.write("Hello");
        document.write("Hello");
        </script>
    </head>
    <body>
        <h1>I am Heading</h1>
        <p>I am paragraph</p>

        </body>
</html>
```

```
<html>
    <head><title>Geeky Shows</title>
        <style>
            p{color: red;}
            h1{color: #F0E68C;}
        </style>
    </head>

    <body>

        <h1>I am Heading</h1>
        <p>I am paragraph</p>
        <script>
        document.write("Hello");
        document.write("Hello");
        document.write("Hello");
        document.write("Hello");
        document.write("Hello");
        document.write("Hello");
        </script>
</body>
</html>
```

If Statement

It is used to execute an instruction or block of instructions only if a condition is fulfilled.

Syntax: -

```
if (condition)
    {
            block of statements;
    }
```

If statement with logical operator

```
if ( (condition1) && (condition2) )
        {
          block of statements;
        }
```

If else Statement

if… else statement is used when a different sequence of instructions is to be executed depending on the logical value(True/False) of the condition evaluated.

Syntax: -

```
if(condition)
   {
       Statement_1_Block;
   }
else
   {
       Statement_2_Block;
   }
Statement_3;
```

Else If Statement

To show a multi-way decision based on several conditions, we use else if statement.
Syntax: -

```
If(condition_1)
  {
   statements_1_Block;
  }
else if(condition_2)
  {
    statement_2_Blocks;
  }
else if(condition_n)
  {
   Statements_n_Block;
  }
else
   statements_x;
```

Switch Statement

All the cases will be evaluated if you don't use break statement.

Check several possible constant values for an expression.

Syntax: -

```
switch(expression){
        case expression 1:
                block of statements 1;
                break;
        case expression2;:
                block of statements 2;
                break;
        .
        .
        default:
                default block of instructions;
        }
```

```
switch(expression){
        case expression 1:
                block of statements 1;
                break;
        case expression 2:
        case expression 3:
                block of statements 2;
                break;

        default:
                default block of instructions;
        }
```

For Loop

The for loop is frequently used, usually where the loop will be traversed a fixed number of times.

Syntax:

```
for (initialization;   test condition;    increment or decrement)
{
        block of statements;
}
```

```
for (initialization; test condition;   increment or decrement)
  {
    block of statements;
  }
```

```
for(i = 0;  i < 5;   i++)
 {
    document.write(i);
 }
```

```
var i = 0;
for( ;  i < 5;   i++)
  {
      document.write(i);
  }
```

```
var i = 0;
for(; ; i++)
{
    if(i==3)
      {
          break;
      }
    document.write(i);
}
```

```
var i = 0;
for( ;   i<5 ;  )
  {
     i++;
     document.write(i);
  }
```

Nested For Loop

For loop inside for loop.

Syntax:

```
        for (initialization;    test condition;     increment or decrement)
        {
                block of statements;
                for (initialization;    test condition;     increment or decrement)
                {
                        block of statements;
                }
        }
```

```
for(i = 0;   i < 3;    i++)
  {
      document.write(i);
      for(j = 0;   j < 5;    j++)
      {
        document.write(j);
      }
  }
```

While loop

The while loop keeps repeating an action until an associated condition returns false.

Syntax:

```
while (test condition)
   {
       body of the loop;
       increment/decrement ;
   }
```

```
var i = 0;
while (i < 5)
  {
      document.write(i);
      i++ ;
  }
```

```
var i = 0;
while (true)
  {
      if (i == 3)
             break;
      document.write(i);
       i++ ;
  }
```

Nested While loop

While Loop inside While Loop

```
var i = 0;
while (i < 3)
  {
    document.write(i);
    i++ ;
    var j = 0;
    while (j < 5)
    {
        document.write(j);
        j++ ;
    }
  }
```

Do While Loop

The do while loop is similar to while loop, but the condition is checked after the loop body is executed. This ensure that the loop body is run at least once.

Syntax :

```
do
  {
     statements;
  } while(test condition);
```

```
var i = 0;
do
 {
     document.write(i);
     i++ ;
 } while (i < 5);
```

```
var i = 0;
do
  {
      if (i == 3)
          break;
      document.write(i);
      i++ ;
  } while (true);
```

Nested Do While Loop

Do while inside Do while

```
var i = 0;
do
 {
     document.write(i);
     i++ ;
    var j = 0;
    do
     {
        document.write(j);
        j++ ;
    } while (j < 5);
 } while (i < 3);
```

Break and Continue

Break Statement – This statement is used to "jumps out" of a loop.

The break statement breaks the loop and continues executing the code after the loop (if any).

Continue Statement – This statement is used to "jumps over" one iteration in the loop.

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

Escape Sequences

Character Constant	Meaning
\b	Backspace
\f	Form feed
\n	Move to new line
\r	Carriage return (Enter)
\t	Horizontal Tab
\v	Vertical Tab
\\	Print back slash
\?	Print question mark
\'	Print single quote
\"	Print double quote
\0	Null character

Function

Function are subprograms which are used to compute a value or perform a task.

Type of Function

- Library or Built-in functions
  Ex: - valueOf( ) , write( ), alert( ) etc

- User-defined functions

Creating and Calling a Function

Creating a Function

Syntax: -

function function_name( )

{

    Block of statement;

}

Ex: -

function display( )

{

    document.write("Geekyshows");

}
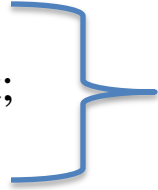
Body of Function

Calling a Function

Syntax: -

    function_name( );

Ex: -

    display( );

Rules

- Function name only starts with a letter, an underscore (_).

- Function name cannot start with a number.

- Do not use reserved keywords. e.g. else, if etc.

- Function names are case-sensitive in JavaScript.

How Function Call Works

The code inside a function isn't executed until that function is called.

```
document.write ("First Line <br />");
display( );
document.write("GeekyShows");
function display( )
 {
    document.write("Welcome to GeekyShows <br />");
 }
 document.write("Last Line <br />");
```

Function with Parameters

Syntax: -

function function_name (parameter1, parameter2, ….)

  {

     Block of statement;

  }

Ex: -

function display(name)

  {

    document.write(name);

  }

- JavaScript function definitions do not specify data types for parameters.
- JavaScript functions do not perform type checking on the passed arguments.
- JavaScript functions do not check the number of arguments received.

Call Function with Parameter

Syntax: -

function function_name (para1, para2, ….)
{
    Block of statement;
}

Syntax:-

function_name(argument1, argument2);

function display(name)
{
    document.write(name);
}

Ex: -
display("Geekyshows");

Function Argument Missing

If a function is called with missing arguments, the missing values are set to undefined.

```
function add (a, b, c)
  {
        document.write("A: " + a + "B: " + b + "C: "+ c);
  }

  add (10, 20);
```

<u>Arguments Object</u>

The arguments object contains an array of the arguments used when the function was called.

This object contains an entry for each argument passed to the function, the first entry's index starting at 0. The arguments object is not an Array. It is similar to an Array, but does not have any Array properties except length.

```
function add(num1, num2) {
  // arguments[0] = 10
  // arguments[1] = 20
}

add(10, 20);
```

Many Function Arguments

If a function is called with too many arguments, these arguments can be reached using the **arguments** object which is a built-in.

```
function add (a, b)
  {
        document.write("A: " + a + "B: " + b);
        document.write("C: " + arguments[2] + "D: " + arguments[3]);

  }

 add (10, 20, 30, 40);
```

Default Parameter

Syntax: -

function function_name (para1, para2, para3="value")

  {

    Block of statement;

  }

Syntax: -

function function_name (para1, para2="value", para3)   // problem undefined

  {

    Block of statement;

  }

Default Parameter

Syntax: -

function function_name (para1, para2="value", para3)   // problem undefined

  {

    Block of statement;

  }

Syntax: -

function function_name (para1, para2="value1", para3="value2")

  {

    Block of statement;

  }

Default Parameter

Ex: -

```
function add (a, b, c=70)
 {
    document.write("A= " + a + "<br>");
    document.write("B= " + b + "<br>");
    document.write("C= " + c + "<br>");
 }

add(10, 20);                    // 10 20 70
add(10, 20, 30);                // 10 20 30
add(10);                        // 10 undefined 70
```

Default Parameter

JavaScript also allows the use of arrays and null as default values.

Ex: -

function add (a, b, c=null)   // null is case sensitive

```
 {
    document.write("A= " + a + "<br>");
    document.write("B= " + b + "<br>");
    document.write("C= " + c + "<br>");
}
add(10, 20);                     // 10 20 null
add(10, 20, 30);                 // 10 20 30
add(10);                         // 10 undefined null
```

```
function add(a=[101])
{
  document.write("A= " + a[0] + "<br>");
}
add([10]);          // 10
add();              // 101
```

Rest Parameters

The rest parameter allows to represent an indefinite number of arguments as an array.

Syntax: -

function function_name (**...**args)

  {

    Block of statement;

  }

Syntax: -

function function_name (a, **...**args)

  {

    Block of statement;

  }

The rest operator must be the last parameter to a function.

Rest Vs Arguments

There are three main differences between rest parameters and the arguments object:-

- Rest parameters are only the ones that haven't been given a separate name, while the arguments object contains all arguments passed to the function.

- The arguments object is not a real array, while Rest Parameters are Array instances, meaning methods like sort, map, forEach or pop can be applied on it directly.

- The arguments object has additional functionality specific to itself (like the callee property).

Return Statement

A return statement may be return Any type data, including arrays and objects.

Syntax : -

        return (variable or expression);

Ex: -

        return (3);

        return (a + b);

        return (a);

Return Statement

Syntax: -

function function_name(para1, para2, …..)

{

    Block of statement;

    return (expression);

}

Ex: -

function add(a, b) {

    return (a + b);   // return a + b ;

}

Variable Scope

JavaScript has two scopes: -

- Global

- Local

Global Scope

A variable that is declared outside a function definition is a global variable, and its value is accessible and modifiable throughout your program.

In a web browser, global variables are deleted when you close the browser window (or tab), but remain available to new pages loaded into the same window.

```
var a = 10;          // Global variable
function add(b) {
    return (a + b); // a is global variable
}
document.write(add(20));
```

Local Scope

A variable that is declared inside a function definition is local. It is created and destroyed every time the function is executed, and it cannot be accessed by any code outside the function. Inside a function, if a variable has not been declared with var it is created as a global variable.

```
function add(b) {
    var a = 10;     // Local Variable
    return (a + b);
}
document.write(add(20));
```

```
function add(b) {
    a = 10;              // Global Variable
    return (a + b);
}
document.write(add(20));
```

Local Scope

If there is function inside a function the inner function can access outer function's variables but outer function can not access inner function's variables.

Function arguments (parameters) work as local variables inside functions.

Block Scope

==Variables declared with *var* do not have block scope.==

```
If(true){
  var i = 10;   // accessible from outside block
  document.write(i);
}
document.write(i);  // possible to access i from outside block
```

Block Scope

Identifiers declared with *let* and *const* do have block scope.

If(true){

  let i = 10;   // not accessible from outside block

  document.write(i);

}

document.write(i); // not possible to access i from outside block

Variable Hoisting

Hoisting is JavaScript's default behavior of moving declaration to the top of the function, if defined in a function, or the top of the global context, if outside a function.

```
var a;
```
Variable declaration

```
a = 10;
```
Variable Initialization

```
var a = 10;
```

```
var a;
a = 10;
```

We Write like this

```
var a = 10;
document.write(a);
var b = 20;
```

Compile Phase

```
var a;
var b;
a = 10;
document.write(a);
b = 20;
```

Variable Hoisting

A variable can be used before it has been declared.

Only variable declarations are hoisted to the top, not variable initialization.

We Write like this

```
a = 10;
document.write(a);
var a;
```

Compile Phase

```
var a;
a = 10;
document.write(a);
```

We Write like this

```
var a = 10;
document.write(a + " " + b);
var b = 20;
```

Compile Phase

```
var a = 10;
var b;
document.write(a + " " + b);
b = 20;
```

Closure

A closure is a function having access to the parent scope. It preserve the data from outside.

A closure is an inner function that has access to the outer (enclosing) function's variables.

For every closure we have three scopes:-

- Local Scope (Own scope)

- Outer Functions Scope

- Global Scope

Function Expression

When we create a function and assign it to a variable, known as function expression.

Ex: -

var myfun = function show( ){
    document.write("GeekyShows");
};

myfun();

Note: -

- You can't call function expression before function definition.
- Function expressions in JavaScript are not hoisted, unlike function declarations.

myfun();

var myfun = function show( ){
    document.write("GeekyShows");
};

Anonymous Functions

Anonymous functions allow the creation of functions which have no specified name.

- Can be stored in a Variable
- Can be Returned in a Function
- Can be pass in a Function

Syntax: -

```
function ( ) {
    body of function;
};
```

Semicolon

Store Anonymous Function in Variable

```javascript
var a = function ( ) {
        document.write("Geekyshows");
};
a();


var a = function (x, y) {
        document.write(x + " " + y);
};
a(10, 20);
```

Passing Anonymous Function as Argument

```javascript
function disp(myfun){
    return myfun();
}


document.write(disp(function(){
    return "GeekyShows";
}));
```

Returning Anonymous Function

```
function disp(a){
        return function(b){
                return a+b;
            };
}
var af = (disp(10));
document.write(af(20));
```

<u>Arrow Function</u>

An arrow function expression (previously, and now incorrectly known as fat arrow function) has a shorter syntax compared to function expressions. Arrow functions are always anonymous.

Syntax: -

    ( ) => { statements};

var myfun = function show( ) {
    document.write("GeekyShows");
};

var myfun = ( ) =>{document.write("GeekyShows");};

Arrow Function

- Do not call before definition

    myfun();

    var myfun = ( ) => { document.write("Geekyshows");};

- An arrow function cannot contain a line break between its parameters and its arrow.

    var myfun = ( )

     => { document.write("Geekyshows");};

    myfun();

Arrow Function

- With one Parameter
  - Syntax: (para) => {statement};
  - Syntax: para => {statement};

- More than one Parameter
  - Syntax: (para1, para2, paraN) => {statement};

- No Parameter
  - Syntax: ( ) => {statement};

Arrow Function

- Default Parameter
  - Syntax: (para1, para2 = value) => {statement};

- Rest Parameter
  - Syntax: (para1, …args) => {statement};

Arrow Function

Syntax: (para1, para2) => expression;

Ex: (a, b) => a+b;

Above code is equivalent to:

function add(a, b) {

         return a + b;

}

Syntax: (para1, para2) => {expression};   // it won't work

Ex: (a, b) => {a+b};

Syntax: (para1, para2) => {*return* expression};

Ex: (a, b) => {return a+b};

Immediately Invoked Function Expression (IIFE)

IIFE (Immediately Invoked Function Expression) is a JavaScript function that runs as soon as it is defined.

It is a design pattern which is also known as Self-Executing Anonymous Function and contains two major parts. The first is the anonymous function with lexical scope enclosed within the Grouping Operator (). This prevents accessing variables within the IIFE idiom as well as polluting the global scope.

The second part is creating the immediately executing function expression (), through which the JavaScript engine will directly interpret the function.

Ex: -

(function( ) { document.write("Geekyshows");})( );

(function(a, b) { document.write(a + " " + b);})(10, 20);

Immediately Invoked Function Expression (IIFE)

- Avoid Creating Global variable and Functions

- As it doesn't define variable and function globally so there will be no name conflicts

- Scope is limited to that particular function

Pass by Value

- JavaScript arguments are passed by value: The function only gets to know the values, not the argument's locations.

- If a function changes an argument's value, it does not change the parameter's original value.

- Changes to arguments are not visible (reflected) outside the function.

Pass by reference

- In JavaScript, object references are values.
- Because of this, objects will behave like they are passed by reference:
- If a function changes an object property, it changes the original value.
- Changes to object properties are visible (reflected) outside the function.

Typeof operator

The typeof operator is used to get the data type (returns a string) of its operand. The operand can be either a literal or a data structure such as a variable, a function, or an object.

Syntax: -

typeof operand

typeof(operand)

Ex: -

typeof "a";

Undefined

The undefined type is used for variable or object properties that either do not exist or have not been assigned a value. The only value an undefined type can have is *undefined*.

var a;

document.write(a);   // Not assigned a value - Undefined

document.write(b);   // Not exist – Undefined Error

<u>Null</u>

The null value indicates an empty value; it is essentially a placeholder that represents "nothing". The null value is defined as an empty object so using typeof operator on a variable holding null shows its type to be object.

var a = null;

document.write(a + "<br>");

document.write(typeof(a) + "<br>");

Undefined Vs Null

Undefined means the value hasn't been set, whereas null means the value has been set to be empty.

var, let and const

var - The scope of a variable declared with var is its current execution context, which is either the enclosing function or, for variables declared outside any function, global.

let - let allows you to declare variables that are limited in scope to the block, statement, or expression on which it is used.

const - This declaration creates a constant whose scope can be either global or local to the block in which it is declared. Global constants do not become properties of the window object, unlike var variables. An initializer for a constant is required; that is, you must specify its value in the same statement in which it's declared which can't be changed later.

Object Oriented Programming

Object-oriented programming (OOP) is a **programming language model** organized around objects rather than "actions" and data rather than logic.

Concepts of OOP

- Encapsulation

- Abstraction

- Inheritance

- Polymorphism

Objects

An object is a collection of properties, and a property is an association between a name (or key) and a value. A property's value can be a function, in which case the property is known as a method. In addition to objects that are predefined in the browser, you can define your own objects.

var fees ={

       Rahul: 100,

       Sumit: 200,   Properties

       Rohan: 300

       total : function ( ) { return(100+200+300); }   Methods

    };

Type of Objects

- User-defined Objects – These are custom objects created by the programmer to bring structure and consistency to a particular programming task.

- Native Objects – These are provided by the JavaScript language itself like String, Number, Boolean, Function, Date, Object, Array, Math, RegExp, Error as well as object that allow creation of user-defined objects and composite types.

- Host Objects – These objects are not specified as part of the JavaScript language but that are supported by most host environments, typically browsers like window, navigator.

- Document Objects – These are part of the Document Object Model (DOM), as defined by the W3C. These objects presents present the programmer with a structured interface to HTML and XML documents. Access to the document objects is provided by the browser via the document property of the window object (window.document).

Declaration and initialization of Object

- **Using Object Literal**

Syntax: - var object_name = { };

Ex: -

var fees = {};

fees['Rahul'] = 100; ⟶ fees.Rahul = 100;

fees['Sumit'] = 200; ⟶ fees.Sumit = 200;

fees['Rohan'] = 300; ⟶ fees.Rohan = 300;

fees["Super Man"] = 400; ⟶ fees.Super Man = 400;

Multiword key required quotation

Declaration and initialization of Object

- **Using Object Literal**

Syntax: - var object_name = { };

Ex: -

var fees = { };

fees['Rahul'] = 100; ⟶ fees.Rahul = 100;

fees['Sumit'] = 200; ⟶ fees.Sumit = 200;

fees['Rohan'] = 300; ⟶ fees.Rohan = 300;

fees['total'] = function ( ) { return(100+200+300); };

fees['total'] = sum;

fees.total = function ( ) { return(100+200+300); };

fees.total = sum;

function sum ( )
 {
    return(100+200+300);
 }

Declaration and initialization of Object

- **Using Object Literal**

Syntax: - var object_name = {key1:value1, key2:value2, key_n:value_n};

Ex: - var fees ={Rahul: 100, Sumit: 200, Rohan: 300};

var fees ={

        Rahul: 100,

        Sumit: 200,

        Rohan: 300

    };

var fees ={

        Rahul: 100,

        Sumit: 200,

        Rohan: 300

        "Super Man": 400

    };

var fees ={Rahul: 100, Sumit: 200, Rohan:300, "Super Man": 400};

Declaration and initialization of Object

- **Using Object Literal**

Syntax: - var object_name = {key1:value1, key2:value2, key_n:value_n, key: function};

Ex: - var fees ={ Rahul: 100, Sumit: 200, Rohan: 300, total: function ( ) {
return(100+200+300); }   };

var fees ={

        Rahul: 100,

        Sumit: 200,

        Rohan: 300,

        total : function ( ) { return(100+200+300); }

    };

Declaration and initialization of Object

- **Using Object Constructor**

Syntax: - var object_name = new Object( );

Ex: - var fees = new Object ( );   var fees = { };

fees['Rahul'] = 100; → fees.Rahul = 100;

fees['Sumit'] = 200; → fees.Sumit = 200;

fees['Rohan'] = 300; → fees.Rohan = 300;

Declaration and initialization of Object

- **Using Object Constructor**

Syntax: - var object_name = new Object ( );

Ex: -

var fees = new Object ( );

fees['Rahul'] = 100; ⟶ fees.Rahul = 100;

fees['Sumit'] = 200; ⟶ fees.Sumit = 200;

fees['Rohan'] = 300; ⟶ fees.Rohan = 300;

fees['total'] = function ( ) { return(100+200+300); };

fees['total'] = sum;

fees.total = function ( ) { return(100+200+300); };

fees.total = sum;

function sum ( )
{
   return(100+200+300);
}

Accessing Properties

A property of an object is some piece of named data it contains. These are accessed with dot operator applied to an object alternative to the dot operator is the array [ ] operator.

Syntax: - object_name.property_name

Ex: -

var fees ={Rahul: 100, Sumit: 200, Rohan: 300};

var fees = { };

fees['Rahul'] = 100; or fees.Rahul = 100;

document.write(**fees['Rahul']**);
document.write(**fees["Rahul"]**);
document.write(**fees.Rahul**);

<u>Accessing Properties</u>

var fees ={Rahul: 100, Sumit: 200, Rohan: 300, "Super Man": 400};

var fees = { };

fees['Super Man'] = 100;
fees.Super Man = 100;

document.write(**fees['Super Man']**);
document.write(**fees["Super Man"]**);
document.write(fees.Super Man);

Accessing Methods

Object members that are functions are called methods. <mark>These are accessed with dot operator applied to an object alternative to the dot operator is the array [ ] operator.</mark>

Syntax: - object_name.Method_name( );

Ex: -

var fees ={Rahul: 100, Sumit: 200, Rohan: 300, total: function ( ) { return(100+200+300); } };

var fees = { };

fees['total'] = function ( ) { return(100+200+300); };

fees.total = function ( ) { return(100+200+300); };

document.write(**fees.total( )**);
document.write(**fees.["total"]( )**);

Adding Properties/Methods

Syntax:-

Object_name.Property_name = value;

Object_name['Property_name'] = value;

Ex: -

fees.Sonam = 600;

fees['Sonam'] = 600;

Deleting Properties

Delete operator is used to delete instance properties.

Syntax:- delete object_name.property_name

Ex: - delete fees.Rahul;

After removal with delete operator, the property has the undefined value.

<u>Factory Function</u>

When a function returns an object, we call it a factory function. It can produce object instance without *new* keyword or *classes*.

Ex:-

```
function mobile( ) {
        return {
                model: 'Galaxy',
                price: function(){ return ("Price: Rs. 3000");}
               };
}
var samsung = mobile( );
document.write(samsung.model + " " + samsung.price( ));
```

<u>Factory Function with Parameter</u>

```
function mobile(model_no) {
        return {
                model: model_no,
                price: function(){
                        return ("Price is Rs. 3000");
                }
        };
}
var samsung = mobile('galaxy');
var nokia = mobile('3310');
document.write(samsung.model + " " + samsung.price( ));
document.write(nokia.model + " " + nokia.price( ));
```

Constructor

Object instance are created with constructor, which are basically special function that prepare new instance of an object for use.

```
function Mobile(){
        this.model = '3310';
        this.price = function(){
                document.write(this.model + " Price Rs. 3000");
                }
}
        var samsung = new Mobile();
        samsung.price();
```

Constructor with Parameter

```
function Mobile(model_no){
        this.model = model_no;
        this.price = function(){
                document.write(this.model + " Price Rs.3000 <br>");
        }
}
        var samsung = new Mobile('Galaxy');
        var nokia = new Mobile('3310');
        samsung.price();
        nokia.price();
```

Check Properties exist

- **Typeof operator**

Syntax:- if (typeof object_name.key !== 'undefined')

Ex:- if(typeof nokia.memory !== 'undefined') {

       document.write("Available");

} else {

       document.write("Doesn't exist");

}

Check Properties exist

- in operator

Syntax: - if ('key' in object_name)

Ex:- if('memory' in nokia) {

       document.write("Available");

} else {

       document.write("Doesn't exist");

}

Check Properties exist

- hasOwnProperty ()

Syntax: - if (object_name.hasOwnProperty("key"))

Ex:- if(nokia.hasOwnProperty('color')) {

    document.write("Available");

} else {

    document.write("Doesn't exist");

}

For in loop

The for...in loop is used to loop through an object's properties.

Syntax: -

for (var variable_name in object_name){

  block of statement

}

Ex: -

for (var specs in samsung) {

      document.write(specs);

}

<u>Class</u>

A specific category can be defined as class.

Example:-
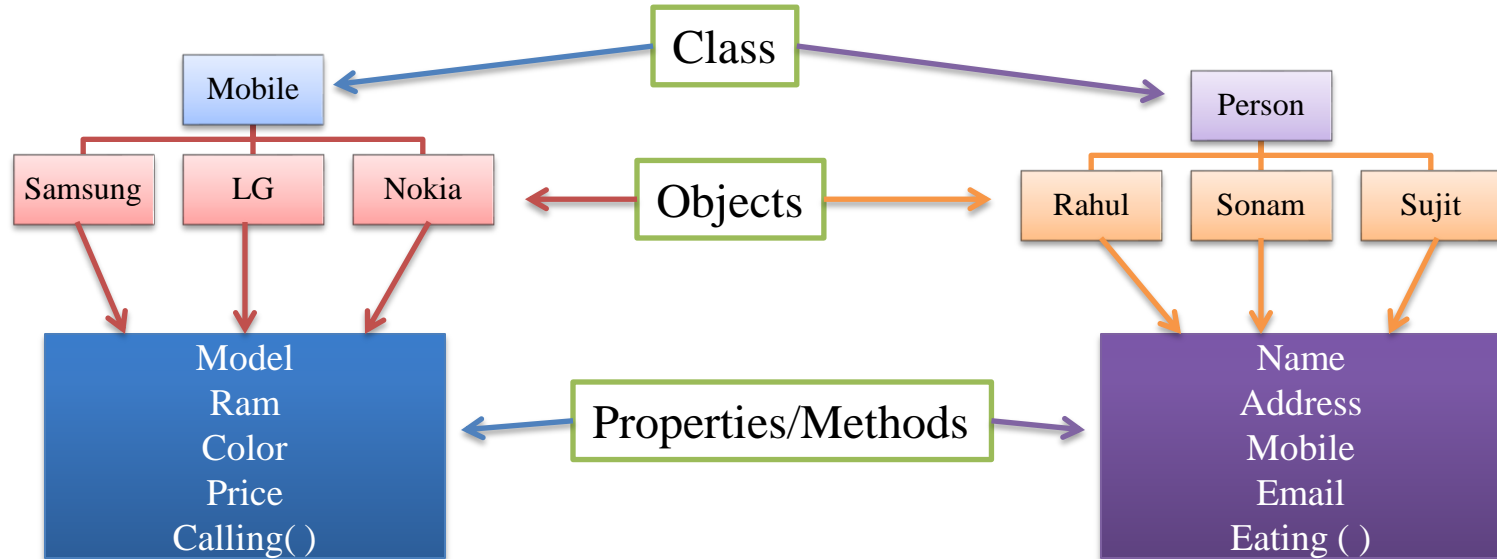
Defining a Class

We define class in JavaScripts using custom constructor.

```
var Mobile = function(model_no, sprice) {
        this.model = model_no;
        this.color = 'white';
        this.price = 3000;
        this.sp = sprice;
        this.sellingprice = function() {
                return (this.price + this.sp);
        };
};
var samsung = new Mobile('Galaxy', 2000);
var nokia = new Mobile('3310', 1000);
```

Private Properties and Methods

Using *var* or *let* or *const* you can create private properties and methods.

Ex: -

this.price

var price

let price

Prototype

Every object has an internal prototype that gives it its structure. This internal prototype is a reference to an object describing the code and data that all objects of that same type will have in common.

Prototype Object
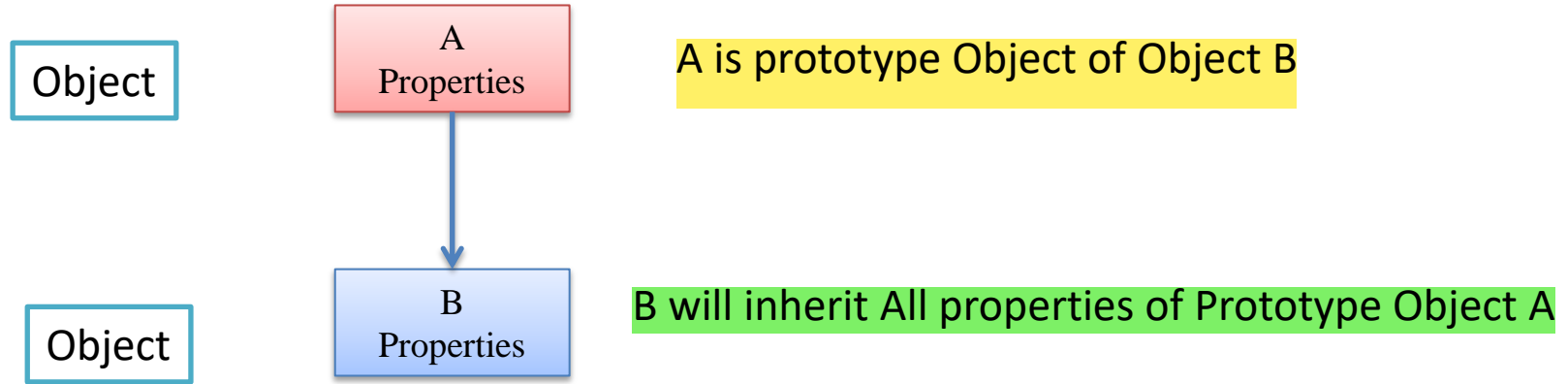
Every object is associated with another Object in JavaScript.

| Object |

A
Properties

A is prototype Object of Object B

| Object |

B
Properties

B will inherit All properties of Prototype Object A

Prototype Object

Every object is associated with another Object in JavaScript.
var b = { };

Object

Object.prototype
Properties

Object.prototype is prototype Object of Object B

Object

B
Properties

B will inherit All properties of Object.prototype
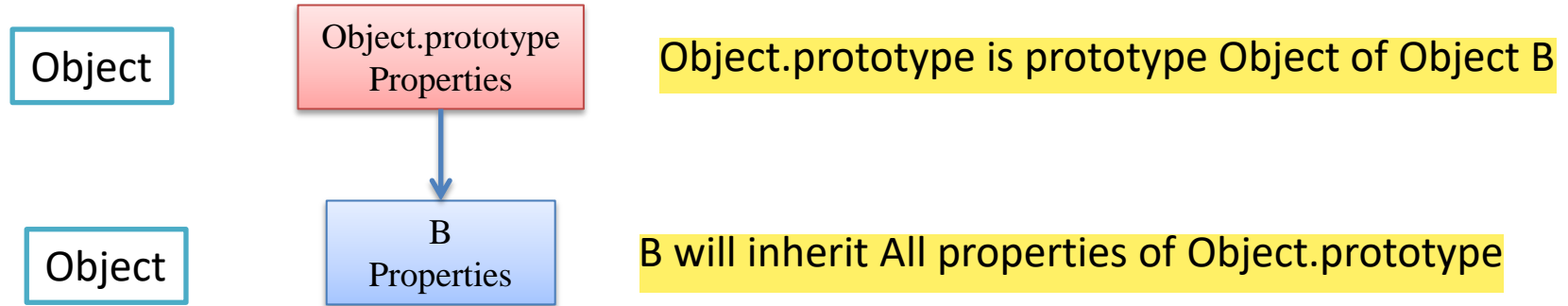
Note - Prototype Object of Object.prototype is null

Prototype Object

Every object is associated with another Object in JavaScript.

var b1 = new Object( );
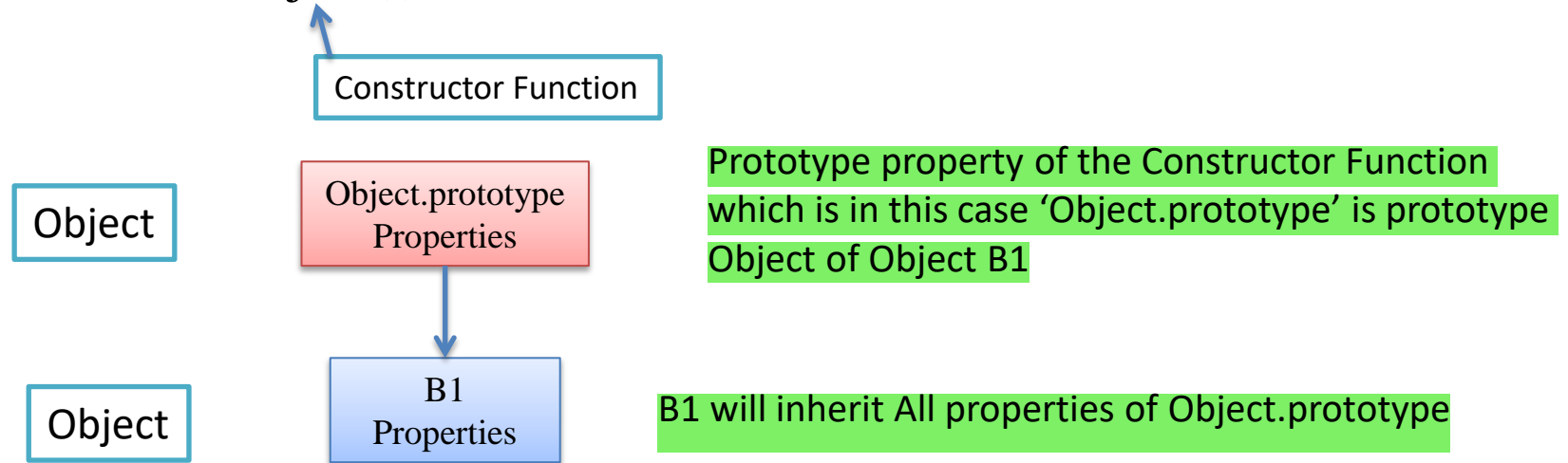
Constructor Function

Object

Object.prototype
Properties

Prototype property of the Constructor Function which is in this case 'Object.prototype' is prototype Object of Object B1

Object

B1
Properties

B1 will inherit All properties of Object.prototype

Note - Prototype Object of Object.prototype is null

Prototype Object

Every object is associated with another Object in JavaScript.

var b2 = new Array( );

Constructor

Object

Array.prototype
Properties
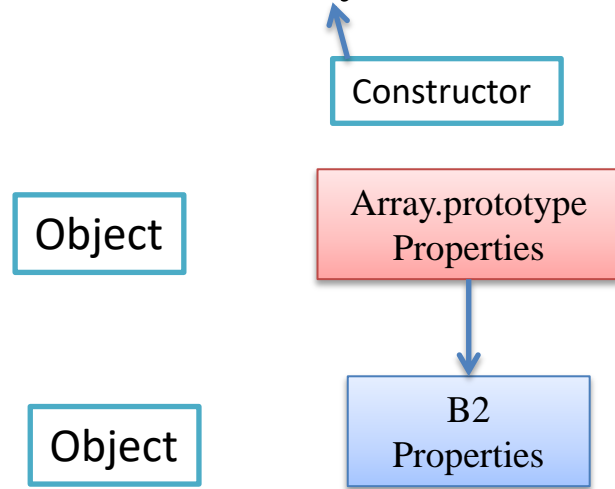
Prototype property of the Constructor Function which is in this case 'Array.prototype' is prototype Object of Object B2

B2
Properties

Object

B2 will inherit All properties of Array.prototype

Note - Prototype Object of Array.prototype is Object.prototype and
Prototype Object of Object.prototype is null
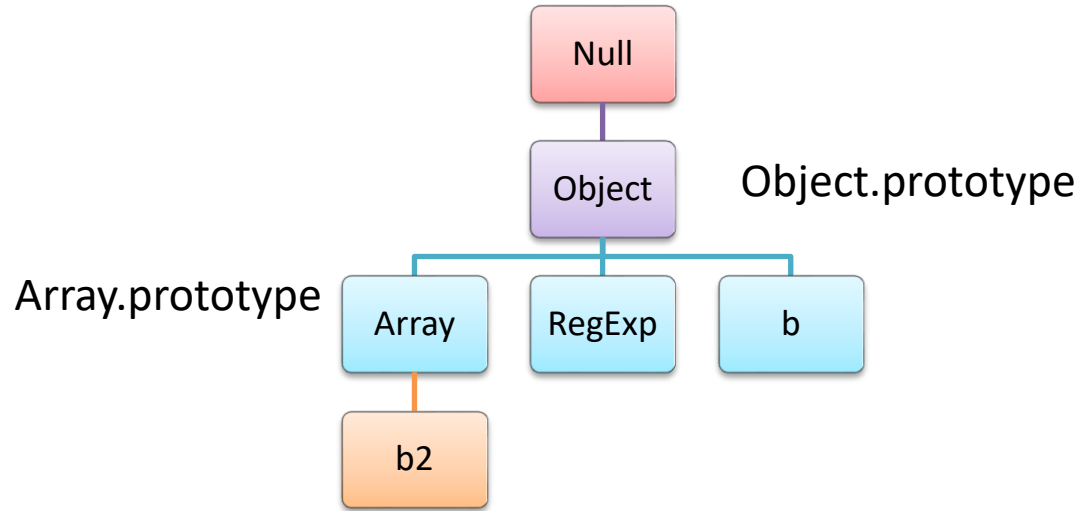
Prototype Object

prototype is the property of function which points to the prototype object. Prototype object can be accessed using *Function_Name.prototype*

Prototype Object

function Mobile( ) {

}

Function

prototype
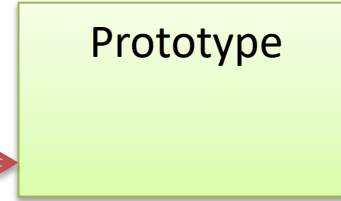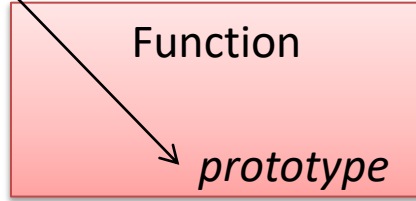
Prototype

var lg = new Mobile()

var g = new Mobile()

Object
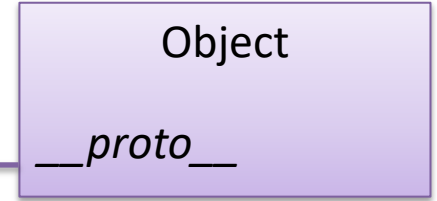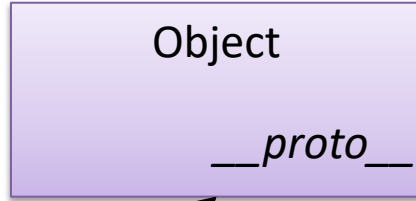
__*proto*__

Object

__*proto*__

When you create a new object of that function using *new* keyword JS Engine creates an object and sets a property named __*proto*__ which points to its function's prototype object

lg.__proto__ === Mobile.prototype

function Mobile( ) {
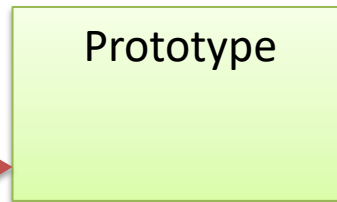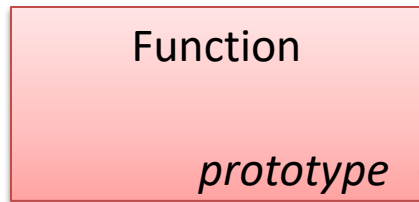
}
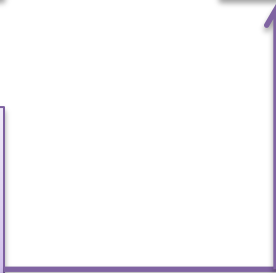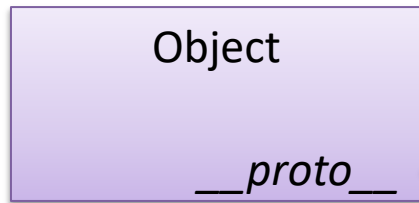
var lg = new Mobile()

lg.*a*   // undefined
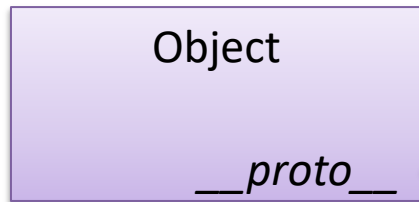
I am looking for *a*

Function

prototype

Prototype

Object

___proto___

```
function Mobile( ) {
   this.a = 10;
}
```

Mobile.prototype.z = 30

var m = new Mobile()

```
function Samsung( ) {
   Mobile.call(this);
   this.b = 20;
}
```

var s = new Samsung()

Mobile Function		Mobile.Prototype	
	prototype		*__proto__*

m Object	
	__proto__

Samsung Function		Samsung.Prototype	
	prototype		*__proto__*

s Object	
	__proto__

Object.prototype	
	__proto__

null
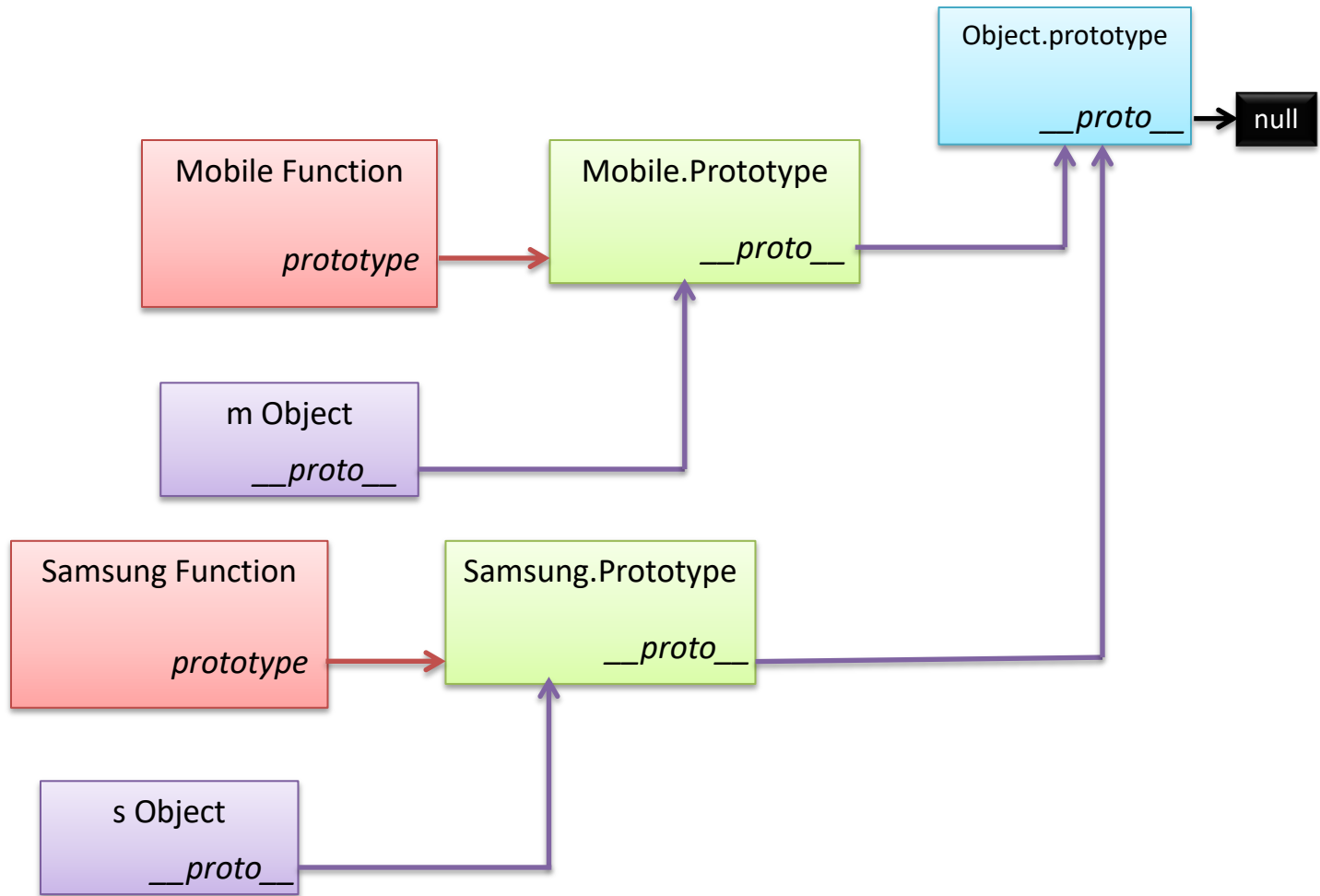
```
function Mobile( ) {
    this.a = 10;
}
```

Mobile.prototype.z = 30

var m = new Mobile()

```
function Samsung( ) {
    Mobile.call(this);
    this.b = 20;
}
```

var s = new Samsung()

Object.prototype

__proto__

null

Mobile Function

prototype

Mobile.Prototype

__proto__

m Object

__proto__

Samsung Function
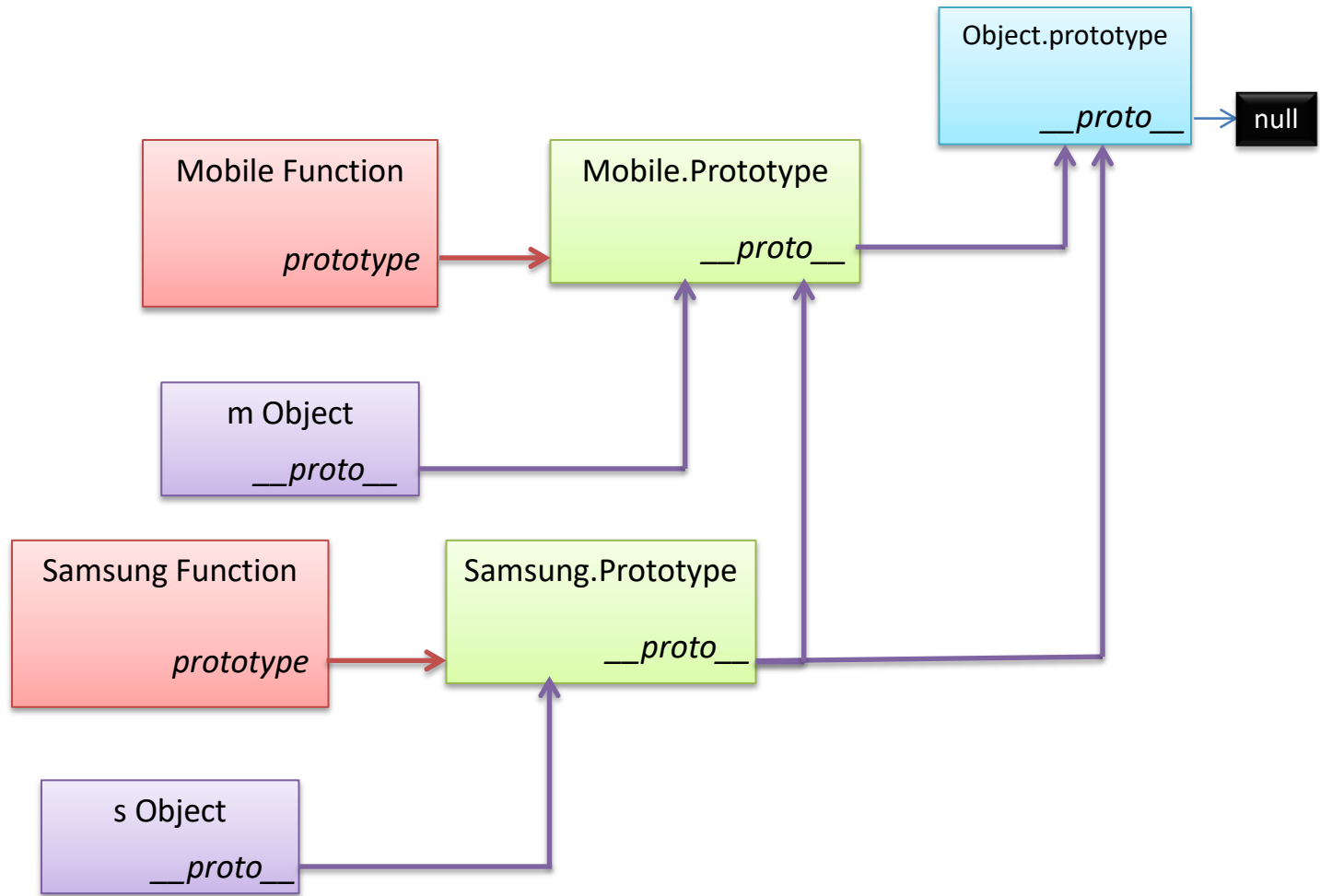
prototype
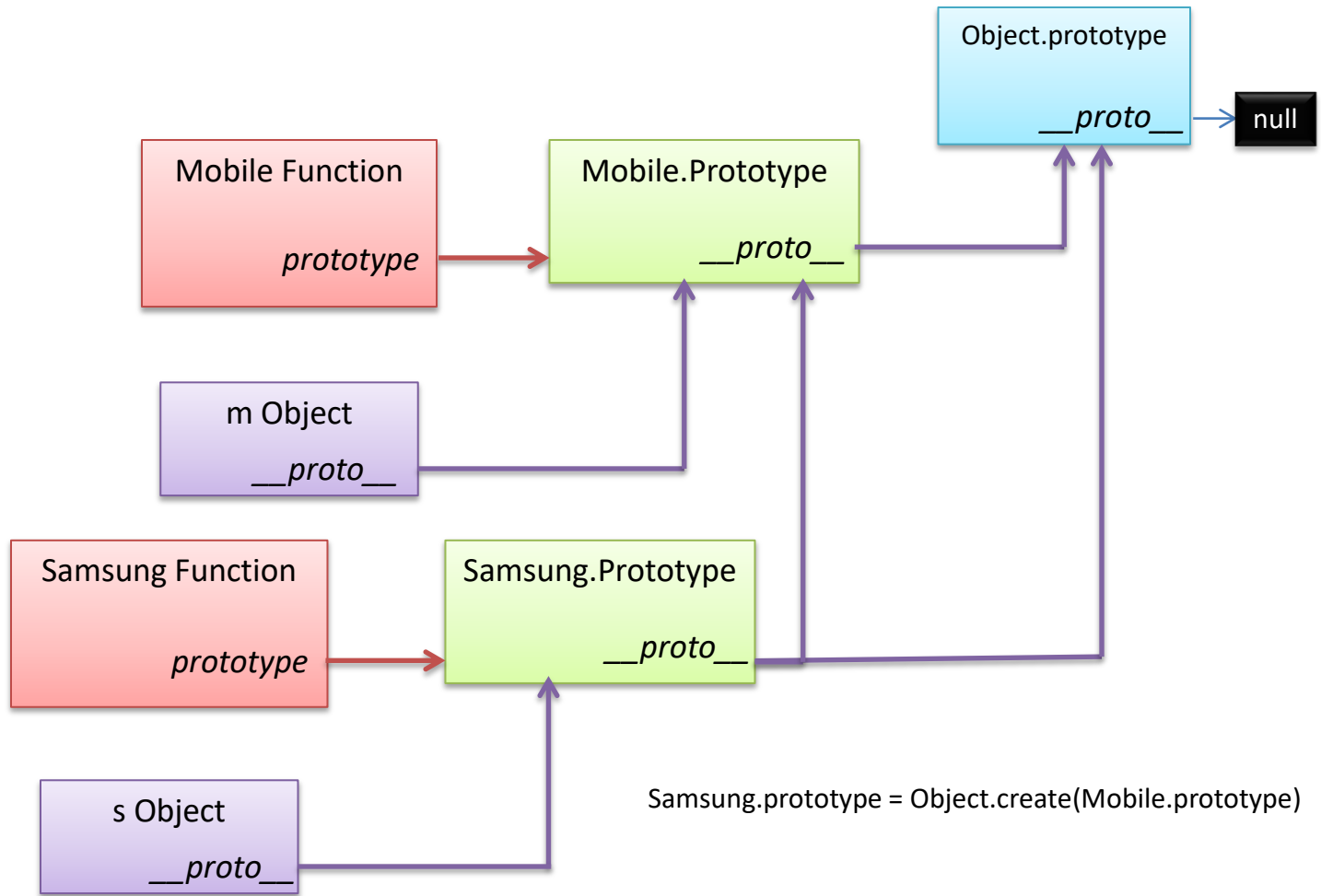
Samsung.Prototype

__proto__

s Object

__proto__

```
function Mobile( ) {
   this.a = 10;
}
```

Mobile.prototype.z = 30

var m = new Mobile()

```
function Samsung( ) {
   Mobile.call(this);
   this.b = 20;
}
```

var s = new Samsung()

Object.prototype

Mobile Function

prototype

Mobile.Prototype

__proto__

null

m Object

__proto__

Samsung Function

prototype

Samsung.Prototype

__proto__

s Object

__proto__

Samsung.prototype = Object.create(Mobile.prototype)

Class

JavaScript classes, introduced in ECMAScript 2015 or ES 6, Classes are in fact "special functions".

There are two way to define class in JavaScript using class keyword:-

- Class Declaration
- Class Expression

Class Declaration

```
class class_name {
        constructor ( ) {
            Properties
        }
        Methods
}
```

```
class Mobile {
        constructor ( ) {
            this.model = 'Galaxy';
        }
        show() { return this.model +
                        "Price 3000";
        }
}
var nokia = new Mobile( );
```

Constructor

The constructor method is a special method for creating and initializing an object created within a class. <mark>There can be only one special method with the name "constructor" in a class.</mark>

class class_name {

    constructor ( ) {

        Properties

    }

}

class Mobile {

    constructor ( ) {

        this.model = 'Galaxy';

    }

    show() { return this.model +
                "Price Rs 3000";

    }

}

var nokia = new Mobile( );

Default Constructor

if you do not specify a constructor method a default constructor is used.

```
class Mobile {
        constructor ( ) {
            this.model = 'Galaxy';
        }
        show() { return this.model +
                "Price Rs 3000";
        }
}
var nokia = new Mobile( );
```

```
class Mobile {
        show() { return this.model +
                "Price Rs 3000";
        }
}
var nokia = new Mobile( );
```

Parameterized Constructor

```
class Mobile {
        constructor ( model_no ) {
            this.model = model_no;
        }
        show() { return this.model + "Price Rs 3000";
        }
}
var nokia = new Mobile('Galaxy');
```

Class Expression

```
var Mobile = class {
    constructor( ) {
        Properties
    }
};
```

```
var Mobile = class Mobile2 {
    constructor( ) {
        Properties
    }
};
```

Class Hoisting

Class Declarations and Class Expression are not hoisted. You first need to declare your class and then access it.

var nokia = new Mobile ( );

class Mobile {

}

class Mobile {
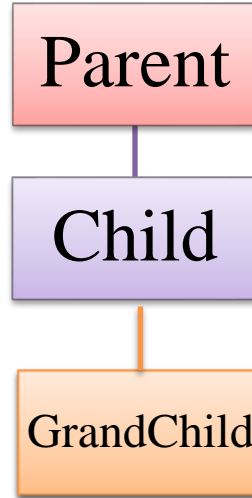
}

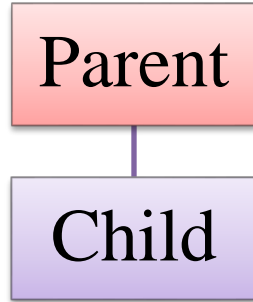var nokia = new Mobile( ) ;

Inheritance

Class Inheritance

class Father {

}

class Son extends Father {

}
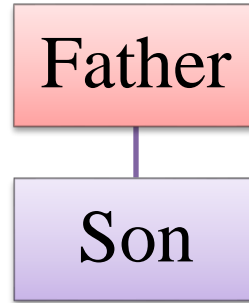
Class Inheritance

- **Inherit Built-in Object**
  - **Date**
  - **String**
  - **Array**

```
class myDate extends Date {


}
```

Super

Super ( ) is used to initialize parent class constructor. If there is a constructor present in subclass, it needs to first call super() before using "this". A constructor can use the super keyword to call the constructor of a parent class.

```
Class Father {
        constructor (money) {
            this.Fmoney = money;
        }
}
Class Son extends Father {
        constructor (money){
            super(money);
        }
}
var s = new Son(10000);
```

Method Overriding

Same function name with different implementation.

Parent → show ( ) {return "Super Class"; }

Child → show ( ) {return "Sub Class"; }
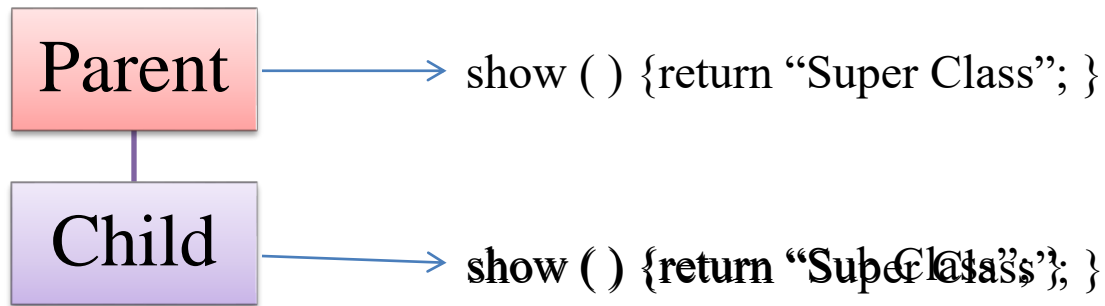
Static Method

The static keyword is used to define a static method for a class. Static methods are called without creating object and cannot be called through a class instance (object). Static methods are often used to create utility functions for an application.

Ex:-

```
class Mobile {
        constructor ( ) {      }
        static disp ( ) { return "Static Method"; }
}
Mobile.disp( ) ;
```

<u>Array</u>

Arrays are collection of data items stored under a single name. Array provide a mechanism for declaring and accessing several data items with only one identifier, thereby simplifying the task of data management.

We use array when we have to deal with multiple data items.

Arrays are a special type of objects. The typeof operator in JavaScript returns "object" for arrays.

Declaration and initialization of Array

- **Using Array Literal**

Syntax: - var array_name = [ ];

Ex: -

var geek = [ ];

geek[0] = "Rahul";

geek[1] = "Ram";

geek[2] = 56;

geek[42] = "Jay";

Declaration and initialization of Array

- **Using Array Literal**

Syntax: - var array_name = [value1, value2, value_n];

Ex: - var geek = ["Rahul", "Ram", 56, "Jay"];

var geek = [, , , ,]; ← All Values are undefined

var geek = [, , , 45, , , 78];

var a = 10, b = 20, c = 30;

var geek = [a, b, c];

var geek = ["Rahul", "Ram", 56, "Jay"];

Index	Value
geek[0]	Rahul
geek[1]	Ram
geek[2]	56
geek[3]	Jay

Note - By default, array starts with index 0.

Declaration and initialization of Array

- **Using Array Constructor**

Syntax: - var array_name = new Array( );

Ex: - var geek = new Array ( );   var geek = [ ];

geek[0] = "Rahul";

geek[1] = "Ram";

geek[2] = 56;

geek[42] = "Jay";

Declaration and initialization of Array

- **Using Array Constructor**

Syntax: - var array_name = new Array(value1, value2, value_n);

Ex: - var geek = new Array("Rahul", "Ram", 56, "Jay");

Ex: - var geek = new Array(10, 20, 30, 40, 50);

var geek = ["Rahul", "Ram", 56, "Jay"];

Syntax: - var array_name = new Array(single_numeric_value);

Ex: - var geek = new Array(5);

This will create an empty array with 5 length. So this is not good idea to use Array Constructor if you have only single numeric value.

var geek = new Array("Rahul", "Ram", 56, "Jay");

var geek = ["Rahul", "Ram", 56, "Jay"];

Index	Value
geek[0]	Rahul
geek[1]	Ram
geek[2]	56
geek[3]	Jay

Note - By default, array starts with index 0.

Important Points

- JavaScript arrays are zero-indexed: the first element of an array is at index 0.
- Using an invalid index number returns undefined.
- It's possible to quote the JavaScript array indexes as well (e.g., geek['2'] instead of geek[2]), although it's not necessary.
- Arrays cannot use strings as element indexes but must use integers.
- There is no associative array in JavaScript.

    geek["fees"] = 200;

- No advantage to use Array Constructor so better to use Array Literal for creating Arrays in JavaScript.

Accessing Array Elements

var geek = ["Rahul", "Ram", 56, "Jay"];

document.write(geek[0]);

document.write(geek[1]);

document.write(geek[2]);

document.write(geek[3]);

document.write(geek[23]);

Undefined

Index	Value
geek[0]	Rahul
geek[1]	Ram
geek[2]	56
geek[3]	Jay
geek[23]	

Accessing Array Elements

Access all at once

var geek = ["Rahul", "Ram", 56, "Jay"];

document.write (geek);

var geek = [ ];

geek[0] = "Rahul";

geek[1] = "Ram";

geek[2] = 56;

document.write (geek);

var geek = [ ];

geek[0] = "Rahul";

geek[1] = "Ram";

geek[2] = 56;

geek[20] = "Jay";

document.write(geek);

Modifying Array Elements

```
var geek = ["Rahul", "Ram", 56, "Jay"];
document.write(geek);
geek[0] = "Rohit";
document.write(geek);


var geek = ["Rahul", "Ram", 56, "Jay"];
var geekyshows = geek;
document.write(geekyshows);
document.write(geek);
geekyshows[0] = "Rohit";
document.write(geek);
```

Removing Array Elements

Array elements can be removed using delete operator. This operator sets the array element it is invoked on to undefined but does not change the array's length.

Syantx :- delete Array_name[index];

Ex:- delete geek[0];

Length Property

The length property retrieves the index of the next available position at the end of the array. The length property is automatically updated as new elements are added to the array. For this reason, length is commonly used to iterate through all elements of an array.

var geek = ["Rahul", "Ram", 56, "Jay"];

document.write(geek.length);

for loop with Array

```
var geek = ["Rahul", "Ram", 56, "Jay"];
for (var i = 0; i<= 3; i++){
        document.write(geek[i] + "<br>");
}


for (var i = 0; i< geek.length; i++){
        document.write(geek[i] + "<br>");
}
```

<u>forEach Loop</u>

<mark>The forEach calls a provided function once for each element in an array, in order.</mark>

<mark>Syntax: - array.forEach(function (value, index, arr) {</mark>

       });

Where,

   value – It is the current value of array index.

   index – Array's index number

   arr - The array object the current element belongs to

Ex:- geek.forEach(*function(name){*

       document.write(name);

   });

for of Loop

The for...of statement creates a loop iterating over iterable objects.

Syntax: -

for (var variable_name of array) {

}

Ex: -

for (var value of geek){

}

Input from User in Array

You can get input from user in an empty array :-

- var geek= [ ];

- var geek = new Array( );

- var geek = new Array(3); // 3 is length of array

Multidimensional Array

Multidimensional array is Arrays of Arrays.

Multidimensional array can be 2D, 3D, 4D etc.

Ex: -

2D - var name[[ ], [ ], [ ]]

Multidimensional Array

Rahul	Dell	10
Sonam	Hp	20
Sumit	Zed	30

[0][0] Rahul	[0][1] Dell	[0][2] 10
[1][0] Sonam	[1][1] Hp	[1][2] 20
[2[0] Sumit	[2][1] Zed	[2][2] 30

Concat ( ) Method

The concat() method is used to merge two or more arrays. This method does not change the existing arrays, but instead returns a new array.

Syntax:- new_array = old_array.concat(value1, value2, value_n);

Syntax:- new_array = old_array1.concat(old_array2, old_array_n);

Ex:-

var geek1 = ["Rahul", "Sonam", "Sumit"];

var new_geek = geek1.concat("Raj", "Rohit");

var geek1 = ["Rahul", "Sonam", "Sumit"];

var geek2 = ["Raj", "Rohit"];

var new_geek = geek1.concat(geek2);

Join ( ) Method

The join() method joins the elements of an array into a string, and returns the string. The elements will be separated by a specified separator. The default separator is comma (,).

Syntax: - array_name.join(separator);

Ex:-

var geek = ["Rahul", "Sonam", "Sumit"];

geek.join(" / ")   // Rahul / Sonam / Sumit

geek.join (" or ")   // Rahul or Sonam or Sumit

geek.join ("")   // RahulSonamSumit

Reverse ( ) Method

The reverse() method reverses the order of the elements in an array.

Syntax :- array_name.reverse( );

Ex:- geek.reverse( );

Slice( ) Method

The slice( ) method returns a shallow copy of a portion of an array into a new array object selected from begin to end (end not included). The original array will not be modified.

Syntax: - array_name.slice(start, end)

Start

If begin is undefined, slice begins from index 0.

If begin is greater than the length of the sequence, an empty array is returned.

A negative index can be used, indicating an offset from the end of the sequence. slice(-2) extracts the last two elements in the sequence.

End

If end is omitted, slice extracts through the end of the sequence (arr.length).

If end is greater than the length of the sequence, slice extracts through to the end of the sequence (arr.length).

A negative index can be used, indicating an offset from the end of the sequence. slice(2,-1) extracts the third element through the second-to-last element in the sequence.

slice extracts up to but not including end.

Splice ( ) Method

The splice() method changes the contents of an array by removing existing elements and/or adding new elements. This method changes the original array.

Syntax:- array_name.splice (start, deletecount, replacevalues);

Start – The first argument start specifies at what position to add/remove items, use negative values to specify the position from the end of the array.

Deletecount – The second argument deletecount, is the number of elements to delete beginning with index start.

Replacevalues – replacevalues are inserted in place of the deleted elements. If more than one separate it by comma.

toString( ) Method

The toString ( ) Method returns a string containing the comma-separated values of the array. This method is invoked automatically when you print an array. It is equivalent to invoking join ( ) method without any arguments. The returned string will separate the elements in the array with commas.

Syntax: - array_name.toString();

Ex:-

var geek = ["Rahul", "Sonam", "Sumit", "Raj", "Rohan"];

geek.toString();

Array.isArray ( ) Method

The Array.isArray() method determines whether the passed value is an Array. This function returns true if the object is an array, and false if not.

Syntax:- Array.isArray(value);

Ex: -

var result = Array.isArray(["Rohan", "Raj"]);   // true

var result = Array.isArray("IAmString");   // false

IndexOf ( ) Method

This method allows to easily find the occurrence of an item in an array.

- If the item not found, it returns -1.
- The search will start at the specified position, or at the beginning if no start position is specified, and end the search at the end of the array.
- If the item is present more than once, the indexOf method returns the position of the first occurrence.

Syntax: - var position = array_name.indexOf(item, start);

Ex:- var position = geek.indexOf("Rohit", 2);

Fill ( ) Method

The fill() method fills all the elements in an array with a static value.

Syntax:- array_name.fill(value, start, end)

Ex:- geek.fill("Don")   // all elements fill with Don

geek.fill("Don", 1, 3)   // fill don starting with index 1 to 3

                                                            (3 not included)

unshift ( ) Method

The unshift() method adds one or more elements to the beginning of an array and returns the new length of the array.

This method changes the length of an array.

Syntax: -

Array_name.unshift(value1, value2, value_n);

Ex: - geek.unshift("Dell", "HP");

var geek_length = geek.unshift("Dell", "HP"); // it will return length of new array

Push ( ) Method

The push() method adds one or more elements to the end of an array and returns the new length of the array.

The new item will be added at the end of the array.

This method changes the length of the array.

Syntax: -

Array_name.push(value1, value2, value_n);

Ex: - geek.push("Dell", "HP");

var geek_length = geek.push("Dell", "HP"); // it will return length of new array

Shift( ) Method

The shift() method removes the first element from an array and returns that removed element. This method changes the length of the array.

Syntax: - array_name.shift( );

Ex: - geek.shift( );

pop( ) Method

The pop() method removes the last element from an array and returns that removed element. This method changes the length of the array.

Syntax: - array_name.pop( );

Ex: - geek.pop( );

Map ( ) Method

The map() method creates a new array with the results of calling a provided function on every element in the calling array.

The map() method calls the provided function once for each element in an array, in order.

map() does not execute the function for array elements without values.

map() does not change the original array.

Syntax:- array_name.map(function(currentValue, index, array)
```
        {


        }, thisValue)
```

Boolean

Boolean is the built-in object corresponding to the primitive Boolean data type. JavaScript boolean can have one of two values: true or false.

Primitive Values

var primitiveTrue = true;

var primitiveFalse = false;

Boolean Function

var functionTrue = Boolean(true);

var functionFalse = Boolean(flase);

Boolean Constructor

var constructorTrue = new Boolean(true);

var constructorFalse = new Boolean(false);

Boolean

If value parameter is omitted or is 0, -0, null, false, NaN, undefined, or the empty string (""), the object has an initial value of false.

Ex:-

var a = Boolean() // false

var a = Boolean(0) // false

var a = Boolean(-0) // false

var a = Boolean(NaN) // false

var a = Boolean(null) // false

String

String is the built in object corresponding to the primitive string data type.

String is group of characters.

Ex: -

"Welcome"	'Welcome'	`Welcome`
"Geeky Shows"	'Geeky Shows'	`Geeky Shows`
"12345"	'12345'	`12345`

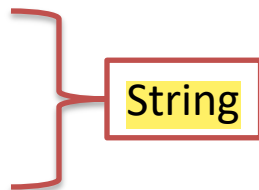# String

Primitive

var str = "Hello GeekyShows";

var str = 'Hello GeekyShows';

var str = `Hello GeekyShows`;

String

Constructor

var str = new String ("Hello GeekyShows");

var str = new String ('Hello GeekyShows');

var str = new String (`Hello GeekyShows`);

Object
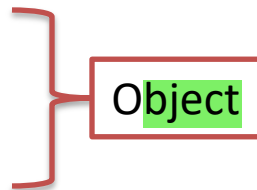
Accessing String

document.write("Hello GeekyShows");

document.write('Hello GeekyShows');

document.write(`Hello GeekyShows`);

var str = "Hello GeekyShows";

var str = 'Hello GeekyShows';

var str = `Hello GeekyShows`;

var str = new String ("Hello GeekyShows");

document.write(str);

Access String

document.write("Hello 'GeekyShows' World");

document.write('Hello "World" GeekyShows');

String Concatenation

var str1 = "Hello";

var str2 = 'GeekyShows';

document.write(str1 str2);

document.write (str1 "Geekyshows");

+ Concat Operator

var str1 = "Hello";

var str2 = 'GeekyShows';

document.write(str1 + str2);

document.write (str1 + "Geekyshows");

document.write (str1 + "Something" + str2);

Concat ( ) Method

The concat ( ) method accepts any number of arguments and returns the string obtained by concatenating the arguments to the string on which it was invoked.

Syntax: - string.concat(string1, string2, string_n);

Ex: - "Hello".concat("Something", "Geekyshows");

var str1 = "Hello";

var str2 = 'GeekyShows';

var new_str = str1.concat(str2);

var new_str = str1.concat("Something", str2);

Escape Notation

- **\0** the NULL character
- **\'** single quote
- **\"** double quote
- **** backslash
- **\n** new line
- **\r** carriage return
- **\v** vertical tab
- **\t** tab
- **\b** backspace
- **\f** form feed

Template Literal/ Template Strings

Template literals are string literals allowing embedded expressions. You can use **multi-line strings** and **string interpolation** features with them.

Template literals are enclosed by the back-tick (` `) character instead of double or single quotes.

var str1 = "Hello GeekyShows";

var str2 = 'Hello GeekyShows';

var str3 = `Hello GeekyShows`;

Multiple Line String

var str = "Hello

GeekyShows";

var str = "Hello \n GeekyShows";

var str = `Hello Line 1

Geekyshows Line 2`;

var str = `Hello Line 1

Something Line 2

Geekyshows Line 3`;

String Interpolation

Template literals can contain placeholders. These are indicated by the dollar sign and curly braces
(${expression})

var str1 = "Hello GeekyShows";

var str2 = 'Hello GeekyShows';

var str3 = `Hello GeekyShows`;

document.write(str1);

document.write(str1 + "World");

document.write(`${str1} World`);

document.write(`${5+4} World`);

document.write(`${a+b} World`);

document.write(`${functionCall} World`);

Tagged Template

Tagged Templates are advanced form of Template literal. Tags allow you to parse template literals with a function. The first argument of a tag function contains an array of string values. The remaining arguments are related to the expressions. In the end, your function can return your manipulated string.

String Methods

- charAt ( )
- charCodeAt( )
- toUpperCase( )
- toLowerCase ( )
- Trim( )
- Replace ( )
- Split ( )

- indexOf( )
- Search ( )
- Slice ( )
- Substring( )
- Substr ( )

Numbers

Number type in JavaScript includes both integer and floating point values. JavaScript numbers are always stored as double precision floating point numbers, following the international IEEE 754 standard.

JavaScript also provide an object representation of numbers.

Ex-

          12

          23.45

          5e3

Numbers

Primitive

```
var a = 10;          // Whole Number
var a = 10.45;       // Decimal Number
var a = 5e3;         // 5000 // 5 x 10^3   exponent
var a = 34e-5;       // 0.00034  exponent
```
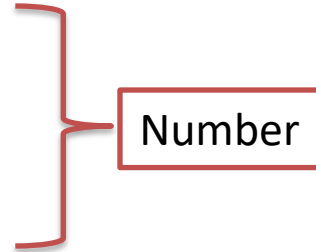
Number

Constructor

```
var a = new Number(10);
var a = new Number(10.45);
var a = new Number(5e3);
```

Object
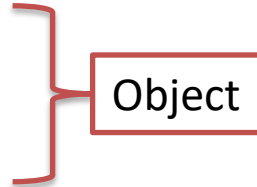
Accessing Number

```
var a = 10;
var a = 10.45;
var a = 5e3;
var a = 34e-5;
var a = new Number(10);
var a = new Number(10.45);
var a = new Number(5e3);

document.write(a);
```

Number with String

var a = "50";   // String
var b = 10;   // Number
var c = 20;   // Number
var d = "Hello";   // String
document.write(a + b);   // 5010
document.write(a - b);   // 40
document.write(b+c+a);   // 10+20+ "50" = 3050
document.write(a+b+c);   // "50" +10+ 20 = 501020
var x = "Result: " + b + c;   // Result: 1020

NaN

The NaN property represents "Not-a-Number" value. This property indicates that a value is not a legal number. NaN never compare equal to anything, even itself.

The NaN property is the same as the Number.Nan property.

Global isNaN ( ) Method

The isNaN() function is used to determines whether a value is an illegal number (Not-a-Number).

This function returns true if the value equates to NaN. Otherwise it returns false.

This function is different from the Number specific Number.isNaN() method.

The global isNaN() function, converts the tested value to a Number, then tests it.

Syntax: - isNaN(value)

Infinity and – Infinity

Infinity or -Infinity is the value JavaScript will return if a number is too large or too small. All Infinity values compare equal to each other.

Ex:-

document.write(5 / 0);   // infinity

document.write(- 5 / 0);   // - infinity

Number Methods

- toString ( )
- toExponential ( )
- toFixed ( )
- toPrecision ( )
- valueOf( )
- isFinite( )
- isInteger( )
- isNan( )
- isSafeInteger( )

toString( )

toString ( ) Method returns a number as a string in other words it converts number into string. We can use this method to output numbers as hexadecimal (16), octal(8), binary(2).

Syntax: -

Variable_name.toString( );

Ex: -

var a = 10;

document.write(a.toString());

document.write(a.toString(2));   // Binary of 10

toExponential ( )

The toExponential() method converts a number into an exponential notation.

Syntax:-

Variable_name.toExponential(y)

Where y is an integer between 0 and 20 representing the number of digits in the notation after the decimal point. If omitted, it is set to as many digits as necessary to represent the value.

Ex: -

```
var a = 58975.98745;
document.write(a.toExponential() + "<br>");
document.write(a.toExponential(2) + "<br>");
document.write(a.toExponential(4) + "<br>");
```

toFixed ( )

The toFixed() method converts a number into a string, keeping a specified number of decimals also it rounds the decimal . If the desired number of decimals are higher than the actual number, nulls are added to create the desired decimal length.

Syntax: -

a.toFixed(y)

Where y is the number of digits after the decimal point. Default is 0 (no digits after the decimal point)

Ex:-

var a = 19.65823;

document.write(a.toFixed() + "<br>");

document.write(a.toFixed(2) + "<br>");

document.write(a.toFixed(4) + "<br>");

document.write(a.toFixed(8) + "<br>");

toPrecision ( )

Syntax:-

Variable_name.toPrecision(y)

Where y is the number of digits. If omitted, it returns the entire number (without any formatting)

Ex:-

var a = 19.65823;

document.write(a.toPrecision() + "<br>");

document.write(a.toPrecision(2) + "<br>");

document.write(a.toPrecision(4) + "<br>");

document.write(a.toPrecision(9) + "<br>");

Number.isNaN()

The **Number.isNaN()** method determines whether a value is NaN (Not-A-Number).

This method returns true if the value is of the type Number, and equates to NaN. Otherwise it returns false.

Number.isNaN() is different from the global isNaN() function. The global isNaN() function converts the tested value to a Number, then tests it.

Number.isNaN() does not convert the values to a Number, and will not return true for any value that is not of the type Number.

Ex-

Number.isNaN(123) //false

Number.isNaN(-1.23) //false

Number.isNaN('123') //false

Number.isNaN('Hello') //false

Number.isInteger( )

The Number.isInteger() method determines whether a value an integer.

This method returns true if the value is of the type Number, and an integer, Otherwise it returns false.

Ex: -

```
document.write(Number.isInteger());            // false
document.write(Number.isInteger(100));         // true
document.write(Number.isInteger(-100));        // true
document.write(Number.isInteger(100.45) );     // false
document.write(Number.isInteger(200-100) );    // true
document.write(Number.isInteger(0.1) );        // false
document.write(Number.isInteger("100") );      // false
document.write(Number.isInteger("Hello") );    // false
```

Number.isSafeInteger()

The Number.isSafeInteger() method determines whether a value is a safe integer.

A safe integer is an integer that can be exactly all integers from (2^53 - 1) to -(2^53 - 1)

This method returns true if the value is of the type Number, and a safe integer. Otherwise it returns false.

Ex: -

Number.isSafeInteger(100)   //true

Number.isSafeInteger(-100)   //true

Number.isSafeInteger(0.1)   //false

Number.isSafeInteger(5645475675445636435436556655677567567556756) ;   // false

Global JS Methods

JavaScript global methods can be used on all JavaScript data types.

- Number ( )
- parseFloat ( )
- parseInt ( )

Number ( )

The Number() function converts the object argument to a number that represents the object's value.

If the value cannot be converted to a legal number, NaN is returned.

If the parameter is a Date object, the Number() function returns the number of milliseconds since midnight January 1, 1970 UTC.

Ex: -

Number(true)

Number("100")

Number(100/"Hello")

parseInt ( )

The parseInt() function parses a string and returns an integer.

Syntax:- parseInt(*string, radix*)

The radix parameter is used to specify which numeral system to be used, for example, a radix of 16 (hexadecimal) indicates that the number in the string should be parsed from a hexadecimal number to a decimal number.

If the radix parameter is omitted, JavaScript assumes the following:

•   If the string begins with "0x", the radix is 16 (hexadecimal)

•   If the string begins with any other value, the radix is 10 (decimal)

Only the first number in the string is returned.

Leading and trailing spaces are allowed.

If the first character cannot be converted to a number, parseInt() returns NaN.

parseInt ( )

```
document.write(parseInt("10")+"<br>");
document.write(parseInt("12.00")+"<br>");
document.write(parseInt("15.45")+"<br>");
document.write(parseInt("10 20 30")+"<br>");
document.write(parseInt("   90   ")+"<br>");
document.write(parseInt("10 years")+"<br>");
document.write(parseInt("years 10")+"<br>");
document.write(parseInt("020")+"<br>");
document.write(parseInt("12", 8)+"<br>");          // 12 octal = 10 decimal
document.write(parseInt("0x12")+"<br>");            // 12 hex = 18 decimal
document.write(parseInt("10", 16)+"<br>");          // 10 hex = 16 decimal
```

parseFloat ( )

The parseFloat() function parses a string and returns a floating point number. This function determines if the first character in the specified string is a number. If it is, it parses the string until it reaches the end of the number, and returns the number as a number, not as a string.

Syntax: - parseFloat(string)

- Only the first number in the string is returned!

- Leading and trailing spaces are allowed.

- If the first character cannot be converted to a number, parseFloat() returns NaN.

parseFloat ( )

document.write(parseFloat("10")+"<br>");

document.write(parseFloat("12.00")+"<br>");

document.write(parseFloat("15.45")+"<br>");

document.write(parseFloat("10 20 30")+"<br>");

document.write(parseFloat("   90   ")+"<br>");

document.write(parseFloat("10 years")+"<br>");

document.write(parseFloat("years 10")+"<br>");

document.write(parseFloat("020")+"<br>");

<u>Math</u>

The Math object holds a set of constants and methods that enable more complex mathematical operations than the basic arithmetic operators. We can not instantiate a Math Object. The Math object is static so it's properties and methods accessed directly.

Ex:-

Math.PI

Math.abs( )

Properties

- E   Returns Euler's number (approx. 2.718)
- LN2   Returns the natural logarithm of 2 (approx. 0.693)
- LN10   Returns the natural logarithm of 10 (approx. 2.302)
- LOG2E   Returns the base-2 logarithm of E (approx. 1.442)
- LOG10E   Returns the base-10 logarithm of E (approx. 0.434)
- PI   Returns PI (approx. 3.14)
- SQRT1_2   Returns the square root of 1/2 (approx. 0.707)
- SQRT2   Returns the square root of 2 (approx. 1.414)

Methods

- Math.abs(arg)   Returns the absolute value of arg
- Math.acos(arg)   Returns the arccosine of arg, in radians
- Math.acosh(arg)   Returns the hyperbolic arccosine of arg
- Math.asin(arg)   Returns the arcsine of arg, in radians
- Math.asinh(arg)   Returns the hyperbolic arcsine of arg
- Math.atan(arg)   Returns the arctangent of arg as a numeric value between -PI/2 and PI/2 radians
- Math.atan2(arg1, arg2)   Returns the arctangent of the quotient of its arguments
- Math.atanh(arg)   Returns the hyperbolic arctangent of arg

Methods

- Math.cbrt(arg)   Returns the cubic root of arg
- Math.ceil(arg)   Returns arg, rounded upwards to the nearest integer
- Math.cos(arg)   Returns the cosine of arg (arg is in radians)
- Math.cosh(arg)   Returns the hyperbolic cosine of arg
- Math.exp(arg)   Returns the value of Ex
- Math.floor(arg)   Returns arg, rounded downwards to the nearest integer
- Math.log(arg)   Returns the natural logarithm (base E) of arg
- Math.random()   Returns a random number between 0 and 1
- Math.round(arg)   Rounds arg to the nearest integer

Methods

- Math.max(arg1, arg2, ...,arg_n)   Returns the number with the highest value
- Math.min(arg1, arg2, ...,arg_n)   Returns the number with the lowest value
- Math.pow(arg1, arg2)   Returns the value of arg to the power of arg2
- Math.sin(arg)   Returns the sine of arg (arg is in radians)
- Math.sinh(arg)   Returns the hyperbolic sine of arg
- Math.sqrt(arg)   Returns the square root of arg
- Math.tan(arg)   Returns the tangent of an angle
- Math.tanh(arg)   Returns the hyperbolic tangent of a number
- Math.trunc(arg)   Returns the integer part of a number (arg)

<u>Date</u>

The Date object provides a sophisticated set of methods for manipulating dates and times.

- It reads client machine date and time so if the client's date or time is incorrect, your script will reflect this fact.
- Days of week and months of the year are enumerated beginning with zero.

       0 – Sunday, 1 – Monday and so on

       0 – January, 1 – February and so on

- Days of month begins with One.

Creating Date Object

Date objects are created with the new Date() constructor. Date Objects created by programmers are static. They do not contain a ticking clock.

Syntax:-

new Date( );

new Date(milliseconds);

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

new Date(dateString);

Creating Date Object

- new Date( ) - new Date() creates a new date object with the current date and time.

Ex: -

var tarikh = new Date( );

document.write(tarikh);

Creating Date Object

- new Date(milliseconds) – It creates a new date object as January 1, 1970, 00:00:00 Universal Time (UTC).

Ex: -

var tarikh = new Date(8640000);

document.write(tarikh);

Creating Date Object

- new Date(year, month, day, hours, minutes, seconds, milliseconds)

It creates object with the date specified by the integer values for the year, month, day, hours, minutes, second, milliseconds. You can omit some of the arguments.

Ex: -

var tarikh = new Date(2018, 4, 25, 9, 45, 35, 0);

var tarikh = new Date(2018, 4, 25, 9, 45, 35);

var tarikh = new Date(2018, 4, 25, 9, 45);

var tarikh = new Date(2018, 4, 25, 9);

var tarikh = new Date(2018, 4, 25);

var tarikh = new Date(2018, 4);

var tarikh = new Date(8640000);

Month and Week Day start with 0
0 – Sunday
0 – January
Month Day starts with 1
1 – 1

Creating Date Object

No. of arguments	Description (in order)
7	year, month, day, hour, minute, second, millisecond
6	year, month, day, hour, minute, second
5	year, month, day, hour, minute
4	year, month, day, hour
3	year, month, day
2	year and month
1	Millisecond

Creating Date Object

- new Date(dateString) - new Date(dateString) creates a new date object from a date string.

Ex: -

var tarikh = new Date("May 12, 2018 10:16:05");

Date Type	Format	Example
ISO Date	YYYY-MM-DD	"2018-06-21" (The International Standard)
Short Date	MM/DD/YYYY	"06/21/2018"
Long Date	MMM DD YYYY	"June 21 2018 or "21 June 2018"

ISO Dates

ISO 8601 is the international standard for the representation of dates and times.

Description	Format	Example
Year and Month	YYYY-MM	2018-06
Only Year	YYYY	2018
Date and Time	YYYY-MM-DD**T**HH:MM:SS**Z**	2018-06-21**T**12:00:00**Z**
Date and Time	YYYY-MM-DD**T**HH:MM:SS+HH:MM YYYY-MM-DD**T**HH:MM:SS-HH:MM	2018-06-21T12:00:00+06:30 2018-06-21T12:00:00-06:30

Date and Time is separated with a capital T.

UTC time is defined with a capital letter Z.

If you want to modify the time relative to UTC, remove the Z and add +HH:MM or -HH:MM instead.

Short Date

- Short dates are written with an "MM/DD/YYYY" format.
- In some browsers, months or days with no leading zeroes may produce an error.
- The behavior of "YYYY/MM/DD" is undefined. Some browsers will try to guess the format. Some will return NaN.
- The behavior of "DD-MM-YYYY" is also undefined. Some browsers will try to guess the format. Some will return NaN.

Long Date

- Long dates are most often written with a "MMM DD YYYY" format.

- Month and day can be in any order.

- Month can be written in full (January), or abbreviated (Jan).

- If you write "June, 21, 2018" Commas are ignored and Names are case insensitive.

Set Date Methods

- setDate()   Set the day as a number (1-31)
- setFullYear()   Set the year (optionally month and day)
- setHours()   Set the hour (0-23)
- setMilliseconds()   Set the milliseconds (0-999)
- setMinutes()   Set the minutes (0-59)
- setMonth()   Set the month (0-11)
- setSeconds()   Set the seconds (0-59)
- setTime()   Set the time (milliseconds since January 1, 1970)

Get Date Methods

- getFullYear()   Get the year as a four digit number (yyyy)
- getMonth()   Get the month as a number (0-11)
- getDate()   Get the day as a number (1-31)
- getHours()   Get the hour (0-23)
- getMinutes()   Get the minute (0-59)
- getSeconds()   Get the second (0-59)
- getMilliseconds()   Get the millisecond (0-999)
- getTime()   Get the time (milliseconds since January 1, 1970)
- getDay()   Get the weekday as a number (0-6)

Converting Dates to String

If you want to create a string in a standard format, Date provides three methods: -

- toString( )
- toUTCString( )
- toGMTString( )

toUTCString ( ) and toGMTString ( ) format the string according to Internet (GMT) standards, whereas toString ( ) creates the string according to Local Time.

Date Methods

- getDate()   Returns the day of the month (from 1-31)
- getDay()   Returns the day of the week (from 0-6)
- getFullYear()   Returns the year
- getHours()   Returns the hour (from 0-23)
- getMilliseconds()   Returns the milliseconds (from 0-999)
- getMinutes()   Returns the minutes (from 0-59)
- getMonth()   Returns the month (from 0-11)
- getSeconds()   Returns the seconds (from 0-59)
- getTime()   Returns the number of milliseconds since midnight Jan 1 1970, and a specified date
- getTimezoneOffset() Returns the time difference between UTC time and local time, in minutes

- getUTCDate() Returns the day of the month, according to universal time (from 1-31)
- getUTCDay() Returns the day of the week, according to universal time (from 0-6)
- getUTCFullYear() Returns the year, according to universal time
- getUTCHours() Returns the hour, according to universal time (from 0-23)
- getUTCMilliseconds() Returns the milliseconds, according to universal time (from 0-999)
- getUTCMinutes() Returns the minutes, according to universal time (from 0-59)
- getUTCMonth() Returns the month, according to universal time (from 0-11)
- getUTCSeconds() Returns the seconds, according to universal time (from 0-59)
- now() Returns the number of milliseconds since midnight Jan 1, 1970
- parse() Parses a date string and returns the number of milliseconds since January 1, 1970
- setDate() Sets the day of the month of a date object
- setFullYear() Sets the year of a date object
- setHours() Sets the hour of a date object
- setMilliseconds() Sets the milliseconds of a date object
- setMinutes() Set the minutes of a date object
- setMonth() Sets the month of a date object
- setSeconds() Sets the seconds of a date object

•	setTime()	Sets a date to a specified number of milliseconds after/before January 1, 1970
•	setUTCDate()	Sets the day of the month of a date object, according to universal time
•	setUTCFullYear()	Sets the year of a date object, according to universal time
•	setUTCHours()	Sets the hour of a date object, according to universal time
•	setUTCMilliseconds()	Sets the milliseconds of a date object, according to universal time
•	setUTCMinutes()	Set the minutes of a date object, according to universal time
•	setUTCMonth()	Sets the month of a date object, according to universal time
•	setUTCSeconds()	Set the seconds of a date object, according to universal time
•	toDateString()	Converts the date portion of a Date object into a readable string
•	toISOString()	Returns the date as a string, using the ISO standard
•	toJSON()	Returns the date as a string, formatted as a JSON date
•	toLocaleDateString()	Returns the date portion of a Date object as a string, using locale conventions
•	toLocaleTimeString()	Returns the time portion of a Date object as a string, using locale conventions
•	toLocaleString()	Converts a Date object to a string, using locale conventions
•	toString()	Converts a Date object to a string
•	toTimeString()	Converts the time portion of a Date object to a string
•	toUTCString()	Converts a Date object to a string, according to universal time
•	UTC()	Returns the number of milliseconds in a date since midnight of January 1, 1970, according to UTC time
•	valueOf()	Returns the primitive value of a Date object

What Next ?

- Learn Advance JavaScript Concepts visit www.geekyshows.com

- Subscribe our Youtube Channel for All Free videos based on this Notes :

    https://www.youtube.com/user/GeekyShow1

- Always update yourself via MDN Web Docs, Geekyshows, w3school and other web sources because Programming has No END