

COURSE CODE : SIT1302

COURSE NAME: INTERNET PROGRAMMING

CHAPTER NAME : JAVA SCRIPT

Introduction to JavaScript, Advantages, Data Types – Variables – Operators - Control Statements – Functions - Objects – Array – Strings – Math – Boolean – Global - Date, and Number - Windows and Frames - Forms and Validation.

Introduction to JavaScript

JavaScript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.

Advantages

- **Less server interaction** – You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
- **Immediate feedback to the visitors** – They don't have to wait for a page reload to see if they have forgotten to enter something.
- **Increased interactivity** – You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
- **Richer interfaces** – You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

Data Types

JavaScript allows you to work with three **primitive data types**

- **Numbers**, eg. 123, 120.50 etc.
- **Strings** of text e.g. "This text string" etc.
- **Boolean** e.g. true or false.

Trivial data types

null and **undefined**, each of which defines only a single value.

composite data type

object

Variables

Variables are declared with the **var** keyword as follows. JavaScript is **untyped** language. This means that a JavaScript variable can hold a value of any data type.

```
<script type="text/javascript">
  <!--
    var money;
    var name;
  //-->
</script>
```

You can also declare multiple variables with the same **var** keyword as follows –

```
<script type="text/javascript">
  <!--
    var money, name;
  //-->
</script>
```

Storing a value in a variable is called **variable initialization**.

```
<script type="text/javascript">
  <!--
    var name = "Ali";
    var money;
    money = 2000.50;
  //-->
</script>
```

JavaScript Variable Scope

The scope of a variable is the region of your program in which it is defined. JavaScript variables have only two scopes.

- **Global Variables** – A global variable has global scope which means it can be defined anywhere in your JavaScript code.

- **Local Variables** – A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

Within the body of a function, a local variable takes precedence over a global variable with the same name.

```
<html>
<body onload = checkscope();>
  <script type = "text/javascript">
    <!--
      var myVar = "global"; // Declare a global variable
      function checkscope( ) {
        var myVar = "local"; // Declare a local variable
        document.write(myVar);
      }
    //-->
  </script>
</body>
</html>
```

This produces the following result –

local

Operators

What is an operator?

Let us take a simple expression **4 + 5 is equal to 9**. Here 4 and 5 are called **operands** and '+' is called the **operator**. JavaScript supports the following types of operators.

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

Arithmetic Operators

+ , - , / , * , % , ++ , --

The following code shows how to use arithmetic operators in JavaScript.

```
<html>
<body>

<script type="text/javascript">
  <!--
    var a = 33;
    var b = 10;
    var c = "Test";
    var linebreak = "<br />";

    document.write("a + b = ");
    result = a + b;
    document.write(result);
    document.write(linebreak);

    document.write("a - b = ");
    result = a - b;
    document.write(result);
    document.write(linebreak);

    document.write("a / b = ");
    result = a / b;
    document.write(result);
    document.write(linebreak);

    document.write("a % b = ");
    result = a % b;
    document.write(result);
    document.write(linebreak);

    document.write("a + b + c = ");
    result = a + b + c;
    document.write(result);
    document.write(linebreak);

    a = ++a;
    document.write("++a = ");
    result = ++a;
    document.write(result);
    document.write(linebreak);

    b = --b;
```

```
document.write("--b = ");
result = --b;
document.write(result);
document.write(linebreak);
//-->
</script>
```

Set the variables to different values and then try...

```
</body>
</html>
```

Output

```
a + b = 43
a - b = 23
a / b = 3.3
a % b = 3
a + b + c = 43Test
++a = 35
--b = 8
Set the variables to different values and then try...
```

Comparison Operators

==, !=, >, >=, <, <=

The following code shows how to use comparison operators in JavaScript.

```
<html>
<body>

<script type="text/javascript">
  <!--
    var a = 10;
    var b = 20;
    var linebreak = "<br />";

    document.write("(a == b) => ");
    result = (a == b);
    document.write(result);
    document.write(linebreak);

    document.write("(a < b) => ");
    result = (a < b);
    document.write(result);
    document.write(linebreak);
```

```
document.write("(a > b) => ");
result = (a > b);
document.write(result);
document.write(linebreak);

document.write("(a != b) => ");
result = (a != b);
document.write(result);
document.write(linebreak);

document.write("(a >= b) => ");
result = (a >= b);
document.write(result);
document.write(linebreak);

document.write("(a <= b) => ");
result = (a <= b);
document.write(result);
document.write(linebreak);
//-->
</script>
```

Set the variables to different values and different operators and then try...

```
</body>
</html>
```

Output

```
(a == b) => false
(a < b) => true
(a > b) => false
(a != b) => true
(a >= b) => false
a <= b => true
```

Set the variables to different values and different operators and then try...

Logical Operators

&& (logical AND)

|| (logical OR)

! (logical NOT)

Try the following code to learn how to implement Logical Operators in JavaScript.

```
<html>
<body>

<script type="text/javascript">
  <!--
    var a = true;
    var b = false;
    var linebreak = "<br />";

    document.write("(a && b) => ");
    result = (a && b);
    document.write(result);
    document.write(linebreak);

    document.write("(a || b) => ");
    result = (a || b);
    document.write(result);
    document.write(linebreak);

    document.write("! (a && b) => ");
    result = !(a && b);
    document.write(result);
    document.write(linebreak);
  //-->
</script>
```

```
<p>Set the variables to different values and different operators and then try...</p>
</body>
</html>
```

Output

```
(a && b) => false
(a || b) => true
!(a && b) => true
Set the variables to different values and different operators and then try...
```

Bitwise Operators

& (Bitwise AND)

| (Bitwise OR)

^ (Bitwise XOR)

~ (Bitwise Not)

<< (Left Shift)

>> (Right Shift)

Try the following code to implement Bitwise operator in JavaScript.

```
<html>
<body>

<script type="text/javascript">
  <!--
    var a = 2; // Bit presentation 10
    var b = 3; // Bit presentation 11
    var linebreak = "<br />";

    document.write("(a & b) => ");
    result = (a & b);
    document.write(result);
    document.write(linebreak);

    document.write("(a | b) => ");
    result = (a | b);
    document.write(result);
    document.write(linebreak);

    document.write("(a ^ b) => ");
    result = (a ^ b);
    document.write(result);
    document.write(linebreak);

    document.write("(~b) => ");
    result = (~b);
    document.write(result);
    document.write(linebreak);

    document.write("(a << b) => ");
    result = (a << b);
    document.write(result);
    document.write(linebreak);

    document.write("(a >> b) => ");
    result = (a >> b);
    document.write(result);
    document.write(linebreak);
  //-->
</script>

<p>Set the variables to different values and different operators and then try...</p>
```



```
</body>
</html>
```

Output

```
(a & b) => 2
(a | b) => 3
(a ^ b) => 1
(~b) => -4
(a << b) => 16
(a >> b) => 0
Set the variables to different values and different operators and then try...
```

Assignment Operators

JavaScript supports the following assignment operators –

`+=, -=, *=, /=, %=`

Try the following code to implement assignment operator in JavaScript.

```
<html>
<body>

<script type="text/javascript">
  <!--
    var a = 33;
    var b = 10;
    var linebreak = "<br />";

    document.write("Value of a => (a = b) => ");
    result = (a = b);
    document.write(result);
    document.write(linebreak);

    document.write("Value of a => (a += b) => ");
    result = (a += b);
    document.write(result);
    document.write(linebreak);

    document.write("Value of a => (a -= b) => ");
    result = (a -= b);
    document.write(result);
    document.write(linebreak);
```

```

document.write("Value of a => (a *= b) => ");
result = (a *= b);
document.write(result);
document.write(linebreak);

document.write("Value of a => (a /= b) => ");
result = (a /= b);
document.write(result);
document.write(linebreak);

document.write("Value of a => (a %= b) => ");
result = (a %= b);
document.write(result);
document.write(linebreak);
//-->
</script>

<p>Set the variables to different values and different operators and then try...</p>
</body>
</html>

```

Output

```

Value of a => (a = b) => 10
Value of a => (a += b) => 20
Value of a => (a -= b) => 10
Value of a => (a *= b) => 100
Value of a => (a /= b) => 10
Value of a => (a %= b) => 0
Set the variables to different values and different operators and then try...

```

Miscellaneous Operator

There are two miscellaneous operators in JavaScript: the **conditional operator** (? :) and the **typeof operator**.

Conditional Operator (? :)

The conditional operator first evaluates an expression for a true or false value and then executes one of the two given statements depending upon the result of the evaluation.

Sr.No	Operator and Description
1	? : (Conditional) If Condition is true? Then value X : Otherwise value Y

Example

Try the following code to understand how the Conditional Operator works in JavaScript.

```
<html>
<body>

<script type="text/javascript">
  <!--
    var a = 10;
    var b = 20;
    var linebreak = "<br />";

    document.write ("((a > b) ? 100 : 200) => ");
    result = (a > b) ? 100 : 200;
    document.write(result);
    document.write(linebreak);

    document.write ("((a < b) ? 100 : 200) => ");
    result = (a < b) ? 100 : 200;
    document.write(result);
    document.write(linebreak);
  //-->
</script>

<p>Set the variables to different values and different operators and then try...</p>
</body>
</html>
```

Output

```
((a > b) ? 100 : 200) => 200
((a < b) ? 100 : 200) => 100
Set the variables to different values and different operators and then try...
```

typeof Operator

The **typeof** operator is a unary operator that is placed before its single operand, which can be of any type. Its value is a string indicating the data type of the operand.

The *typeof* operator evaluates to "number", "string", or "boolean" if its operand is a number, string, or boolean value and returns true or false based on the evaluation.

Here is a list of the return values for the **typeof** Operator.

Type	String Returned by typeof
Number	"number"
String	"string"
Boolean	"boolean"
Object	"object"
Function	"function"
Undefined	"undefined"
Null	"object"

Example

The following code shows how to implement **typeof** operator.

```
<html>
<body>

  <script type="text/javascript">
    <!--
```

```

var a = 10;
var b = "String";
var linebreak = "<br />";

result = (typeof b == "string" ? "B is String" : "B is Numeric");
document.write("Result => ");
document.write(result);
document.write(linebreak);

result = (typeof a == "string" ? "A is String" : "A is Numeric");
document.write("Result => ");
document.write(result);
document.write(linebreak);
//-->
</script>

<p>Set the variables to different values and different operators and then try...</p>
</body>
</html>

```

Output

```

Result => B is String
Result => A is Numeric
Set the variables to different values and different operators and then try...

```

Control Statements

It used to control the flow of execution within a program .Two types

1. Conditional

Decision Making Statement

1. if statement
2. if .. else statement
3. if – else – if statement
4. switch

Loop Control Statement

- 1.while
2. do – while
3. for
4. for – in

2. Unconditional

1. break
2. continue
3. goto

if statement

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

Syntax

The syntax for a basic if statement is as follows –

```
if (expression){  
    Statement(s) to be executed if expression is true  
}
```

Here a JavaScript expression is evaluated. If the resulting value is true, the given statement(s) are executed. If the expression is false, then no statement would be not executed. Most of the times, you will use comparison operators while making decisions.

Example

Try the following example to understand how the **if** statement works.

```
<html>  
<body>  
  
  <script type="text/javascript">  
    <!--  
      var age = 20;  
  
      if( age > 18 ){  
        document.write("<b>Qualifies for driving</b>");  
      }  
    //-->  
  </script>  
  
  <p>Set the variable to different value and then try...</p>  
</body>  
</html>
```

Output

```
Qualifies for driving  
Set the variable to different value and then try...
```

if...else statement:

The **'if...else'** statement is the next form of control statement that allows JavaScript to execute statements in a more controlled way.

Syntax

```
if (expression){  
    Statement(s) to be executed if expression is true  
}  
else{  
    Statement(s) to be executed if expression is false  
}
```

Here JavaScript expression is evaluated. If the resulting value is true, the given statement(s) in the 'if' block, are executed. If the expression is false, then the given statement(s) in the else block are executed.

Example

Try the following code to learn how to implement an if-else statement in JavaScript.

```
<html>  
<body>  
  
  <script type="text/javascript">  
    <!--  
      var age = 15;  
  
      if( age > 18 ){  
        document.write("<b>Qualifies for driving</b>");  
      }  
  
      else{  
        document.write("<b>Does not qualify for driving</b>");  
      }  
    //-->  
  </script>  
  
  <p>Set the variable to different value and then try...</p>  
</body>
```

```
</html>
```

Output

Does not qualify for driving
Set the variable to different value and then try...

if...else if... statement

The **if...else if...** statement is an advanced form of **if...else** that allows JavaScript to make a correct decision out of several conditions.

Syntax

The syntax of an if-else-if statement is as follows –

```
if (expression 1){  
    Statement(s) to be executed if expression 1 is true  
}  
  
else if (expression 2){  
    Statement(s) to be executed if expression 2 is true  
}  
  
else if (expression 3){  
    Statement(s) to be executed if expression 3 is true  
}  
  
else{  
    Statement(s) to be executed if no expression is true  
}
```

There is nothing special about this code. It is just a series of **if** statements, where each **if** is a part of the **else** clause of the previous statement. Statement(s) are executed based on the true condition, if none of the conditions is true, then the **else** block is executed.

Example

Try the following code to learn how to implement an if-else-if statement in JavaScript.

```
<html>  
<body>  
  
    <script type="text/javascript">  
        <!--  
            var book = "maths";
```



```

if( book == "history" ){
    document.write("<b>History Book</b>");
}

else if( book == "maths" ){
    document.write("<b>Maths Book</b>");
}

else if( book == "economics" ){
    document.write("<b>Economics Book</b>");
}

else{
    document.write("<b>Unknown Book</b>");
}
//-->
</script>

<p>Set the variable to different value and then try...</p>
</body>
<html>

```

Output

Maths Book

Set the variable to different value and then try...

Switch Statement

The objective of a **switch** statement is to give an expression to evaluate and several different statements to execute based on the value of the expression. The interpreter checks each **case** against the value of the expression until a match is found. If nothing matches, a **default** condition will be used.

```

switch (expression)
{
    case condition 1: statement(s)
    break;

    case condition 2: statement(s)
    break;
    ...

    case condition n: statement(s)
    break;

```

```
    default: statement(s)
}
```

The **break** statements indicate the end of a particular case. If they were omitted, the interpreter would continue executing each statement in each of the following cases.

We will explain **break** statement in *Loop Control* chapter.

Example

Try the following example to implement switch-case statement.

```
<html>
<body>

<script type="text/javascript">
  <!--
    var grade='A';
    document.write("Entering switch block<br />");
    switch (grade)
    {
      case 'A': document.write("Good job<br />");
        break;

      case 'B': document.write("Pretty good<br />");
        break;

      case 'C': document.write("Passed<br />");
        break;

      case 'D': document.write("Not so good<br />");
        break;

      case 'F': document.write("Failed<br />");
        break;

      default: document.write("Unknown grade<br />")
    }
    document.write("Exiting switch block");
  //-->
</script>

<p>Set the variable to different value and then try...</p>
</body>
</html>
```

Output

```
Entering switch block
```

Good job
Exiting switch block
Set the variable to different value and then try...

Looping Statements

While writing a program, you may encounter a situation where you need to perform an action over and over again. In such situations, you would need to write loop statements to reduce the number of lines.

The while Loop

The most basic loop in JavaScript is the **while** loop which would be discussed in this chapter. The purpose of a **while** loop is to execute a statement or code block repeatedly as long as an **expression** is true. Once the expression becomes **false**, the loop terminates.

Syntax

The syntax of **while loop** in JavaScript is as follows –

```
while (expression){  
    Statement(s) to be executed if expression is true  
}
```

Example

Try the following example to implement while loop.

```
<html>  
<body>  
  
<script type="text/javascript">  
    <!--  
        var count = 0;  
        document.write("Starting Loop ");  
  
        while (count < 10){  
            document.write("Current Count : " + count + "<br />");  
            count++;  
        }  
  
        document.write("Loop stopped!");  
    //-->  
</script>
```

```
<p>Set the variable to different value and then try...</p>
</body>
</html>
```

Output

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!
Set the variable to different value and then try...
```

The do...while Loop

The **do...while** loop is similar to the **while** loop except that the condition check happens at the end of the loop. This means that the loop will always be executed at least once, even if the condition is **false**.

Syntax

The syntax for **do-while** loop in JavaScript is as follows –

```
do{
    Statement(s) to be executed;
} while (expression);
```

Example

Try the following example to learn how to implement a **do-while** loop in JavaScript.

```
<html>
<body>
```

```
<script type="text/javascript">
  <!--
    var count = 0;

    document.write("Starting Loop" + "<br />");
    do{
      document.write("Current Count : " + count + "<br />");
      count++;
    }

    while (count < 5);
    document.write ("Loop stopped!");
  //-->
</script>

<p>Set the variable to different value and then try...</p>
</body>
</html>
```

Output

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Loop Stopped!
Set the variable to different value and then try...
```

For Loop

The **'for'** loop is the most compact form of looping. It includes the following three important parts –

- The **loop initialization** where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.
- The **test statement** which will test if a given condition is true or not. If the condition is true, then the code given inside the loop will be executed, otherwise the control will come out of the loop.
- The **iteration statement** where you can increase or decrease your counter.

You can put all the three parts in a single line separated by semicolons.

Syntax

The syntax of for loop in JavaScript is as follows –

```
for (initialization; test condition; iteration statement){  
    Statement(s) to be executed if test condition is true  
}
```

Example

Try the following example to learn how a for loop works in JavaScript.

```
<html>  
<body>  
  
  <script type="text/javascript">  
    <!--  
      var count;  
      document.write("Starting Loop" + "<br />");  
  
      for(count = 0; count < 10; count++){  
        document.write("Current Count : " + count );  
        document.write("<br />");  
      }  
  
      document.write("Loop stopped!");  
    //-->  
  </script>  
  
  <p>Set the variable to different value and then try...</p>  
</body>  
</html>
```

Output

```
Starting Loop  
Current Count : 0  
Current Count : 1  
Current Count : 2  
Current Count : 3  
Current Count : 4  
Current Count : 5  
Current Count : 6  
Current Count : 7  
Current Count : 8  
Current Count : 9  
Loop stopped!
```

Set the variable to different value and then try...

For – in

The **for...in** loop is used to loop through an object's properties. As we have not discussed Objects yet, you may not feel comfortable with this loop. But once you understand how objects behave in JavaScript, you will find this loop very useful.

Syntax

```
for (variablename in object){  
    statement or block to execute  
}
```

In each iteration, one property from **object** is assigned to **variablename** and this loop continues till all the properties of the object are exhausted.

Example

Try the following example to implement 'for-in' loop. It prints the web browser's **Navigator** object.

```
<html>  
<body>  
  
  <script type="text/javascript">  
    <!--  
      var aProperty;  
      document.write("Navigator Object Properties<br /> ");  
  
      for (aProperty in navigator) {  
        document.write(aProperty);  
        document.write("<br />");  
      }  
      document.write ("Exiting from the loop!");  
    //-->  
  </script>  
  
  <p>Set the variable to different object and then try...</p>  
</body>  
</html>
```

Output

```
Navigator Object Properties
serviceWorker
webkitPersistentStorage
webkitTemporaryStorage
geolocation
doNotTrack
onLine
languages
language
userAgent
product
platform
appVersion
appName
appCodeName
Exiting from the loop!
Set the variable to different object and then try...
```

Functions

A function is a group of reusable code which can be called anywhere in your program. This eliminates the need of writing the same code again and again. It helps programmers in writing modular codes. Functions allow a programmer to divide a big program into a number of small and manageable functions.

Function Definition

Before we use a function, we need to define it. The most common way to define a function in JavaScript is by using the **function** keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces.

Syntax

The basic syntax is shown here.

```
<script type="text/javascript">
  <!--
    function functionname(parameter-list)
    {
```



```
statements  
  
}  
  
//-->  
  
</script>
```

Example

Try the following example. It defines a function called sayHello that takes no parameters –

```
<script type="text/javascript">  
  
<!--  
  
function sayHello()  
  
{  
  
    alert("Hello there");  
  
}  
  
//-->  
  
</script>
```

Calling a Function

To invoke a function somewhere later in the script, you would simply need to write the name of that function as shown in the following code.

```
<html>  
<head>  
  
    <script type="text/javascript">  
        function sayHello()  
        {  
            document.write ("Hello there!");  
        }  
    </script>  
  
</head>  
<body>  
    <p>Click the following button to call the function</p>  
  
    <form>  
        <input type="button" onclick="sayHello()" value="Say Hello">
```

```
</form>

<p>Use different text in write method and then try...</p>
</body>
</html>
```

Output

Hello there!

Function Parameters

Till now, we have seen functions without parameters. But there is a facility to pass different parameters while calling a function. These passed parameters can be captured inside the function and any manipulation can be done over those parameters. A function can take multiple parameters separated by comma.

Example

Try the following example. We have modified our **sayHello** function here. Now it takes two parameters.

```
<html>
<head>

  <script type="text/javascript">
    function sayHello(name, age)
    {
      document.write (name + " is " + age + " years old.");
    }
  </script>

</head>
<body>
  <p>Click the following button to call the function</p>

  <form>
    <input type="button" onclick="sayHello('Zara', 7)" value="Say Hello">
  </form>

  <p>Use different parameters inside the function and then try...</p>
</body>
</html>
```

The return Statement

A JavaScript function can have an optional **return** statement. This is required if you want to return a value from a function. This statement should be the last statement in a function.

For example, you can pass two numbers in a function and then you can expect the function to return their multiplication in your calling program.

Example

Try the following example. It defines a function that takes two parameters and concatenates them before returning the resultant in the calling program.

```
<html>
<head>

<script type="text/javascript">
  function concatenate(first, last)
  {
    var full;
    full = first + last;
    return full;
  }

  function secondFunction()
  {
    var result;
    result = concatenate('Zara', 'Ali');
    document.write (result );
  }
</script>

</head>

<body>
<p>Click the following button to call the function</p>

<form>
  <input type="button" onclick="secondFunction()" value="Call Function">
</form>

<p>Use different parameters inside the function and then try...</p>

</body>
</html>
```

Objects

JavaScript is an Object Oriented Programming (OOP) language. A programming language can be called object-oriented if it provides four basic capabilities to developers –

- **Encapsulation** – the capability to store related information, whether data or methods, together in an object.
- **Aggregation** – the capability to store one object inside another object.
- **Inheritance** – the capability of a class to rely upon another class (or number of classes) for some of its properties and methods.
- **Polymorphism** – the capability to write one function or method that works in a variety of different ways.

Objects are composed of attributes. Attribute is considered to be property.

Object Properties

Object properties can be any of the three primitive data types, or any of the abstract data types, such as another object. Object properties are usually variables that are used internally in the object's methods, but can also be globally visible variables that are used throughout the page.

The syntax for adding a property to an object is –

```
objectName.objectProperty = propertyValue;
```

For example – The following code gets the document title using the **"title"** property of the **document** object.

```
var str = document.title;
```

User-Defined Objects

All user-defined objects and built-in objects are descendants of an object called **Object**.

The new Operator

The **new** operator is used to create an instance of an object. To create an object, the **new** operator is followed by the constructor method.

In the following example, the constructor methods are `Object()`, `Array()`, and `Date()`. These constructors are built-in JavaScript functions.

```
var employee = new Object();  
var books = new Array("C++", "Perl", "Java");
```

```
var day = new Date("August 15, 1947");
```

Array Object

The **Array** object lets you store multiple values in a single variable. It stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Syntax

Use the following syntax to create an **Array** object –

```
var fruits = new Array( "apple", "orange", "mango" );
```

The **Array** parameter is a list of strings or integers. When you specify a single numeric parameter with the Array constructor, you specify the initial length of the array. The maximum length allowed for an array is 4,294,967,295.

You can create array by simply assigning values as follows –

```
var fruits = [ "apple", "orange", "mango" ];
```

You will use ordinal numbers to access and to set values inside an array as follows.

fruits[0] is the first element

fruits[1] is the second element

fruits[2] is the third element

Array Methods

Here is a list of the methods of the Array object along with their description.

Sr.No	Method & Description
1	concat() Returns a new array comprised of this array joined with other array(s) and/or value(s).
2	forEach() Calls a function for each element in the array.
3	indexOf() Returns the first (least) index of an element within the array equal to the specified value, or -1 if none is found.
4	join() Joins all elements of an array into a string.
5	lastIndexOf() Returns the last (greatest) index of an element within the array equal to the specified value, or -1 if none is found.
6	pop() Removes the last element from an array and returns that element.
7	push() Adds one or more elements to the end of an array and returns the new length of the array.
8	reverse() Reverses the order of the elements of an array -- the first becomes the last, and the last becomes the first.
9	shift()

	Removes the first element from an array and returns that element.
10	slice() Extracts a section of an array and returns a new array.
11	sort() Sorts the elements of an array
12	splice() Adds and/or removes elements from an array.
13	toString() Returns a string representing the array and its elements.

String Object

The **String** object lets you work with a series of characters; As JavaScript automatically converts between string primitives and String objects .

Syntax

Use the following syntax to create a String object –

```
var val = new String(string);
```

The **String** parameter is a series of characters that has been properly encoded.

String Methods

Here is a list of the methods available in String object along with their description.

Sr.No	Method & Description
1	charAt() Returns the character at the specified index.

2	charCodeAt() Returns a number indicating the Unicode value of the character at the given index.
3	concat() Combines the text of two strings and returns a new string.
4	indexOf() Returns the index within the calling String object of the first occurrence of the specified value, or -1 if not found.
5	lastIndexOf() Returns the index within the calling String object of the last occurrence of the specified value, or -1 if not found.
6	localeCompare() Returns a number indicating whether a reference string comes before or after or is the same as the given string in sort order.
7	match() Used to match a regular expression against a string.
8	replace() Used to find a match between a regular expression and a string, and to replace the matched substring with a new substring.
9	search() Executes the search for a match between a regular expression and a specified string.
10	slice() Extracts a section of a string and returns a new string.
11	split() Splits a String object into an array of strings by separating the string into substrings.

12	substr() Returns the characters in a string beginning at the specified location through the specified number of characters.
13	substring() Returns the characters in a string between two indexes into the string.
14	toLocaleLowerCase() The characters within a string are converted to lower case while respecting the current locale.
15	toLocaleUpperCase() The characters within a string are converted to upper case while respecting the current locale.
16	toLowerCase() Returns the calling string value converted to lower case.
17	toString() Returns a string representing the specified object.
18	toUpperCase() Returns the calling string value converted to uppercase.
19	valueOf() Returns the primitive value of the specified object.

Math Object

The **math** object provides you properties and methods for mathematical constants and functions.

All the properties and methods of **Math** are static and can be called by using **Math** as an object without creating it.

Syntax

The syntax to call the properties and methods of Math are as follows

```
var pi_val = Math.PI;  
var sine_val = Math.sin(30);
```

Math Methods

Here is a list of the methods associated with Math object and their description

Sr.No	Method & Description
1	abs() Returns the absolute value of a number.
2	acos() Returns the arccosine (in radians) of a number.
3	asin() Returns the arcsine (in radians) of a number.
4	atan() Returns the arctangent (in radians) of a number.
5	atan2() Returns the arctangent of the quotient of its arguments.
6	ceil() Returns the smallest integer greater than or equal to a number.
7	cos() Returns the cosine of a number.
8	exp() Returns E^N , where N is the argument, and E is Euler's constant, the base of the natural logarithm.

9	floor() Returns the largest integer less than or equal to a number.
10	log() Returns the natural logarithm (base E) of a number.
11	max() Returns the largest of zero or more numbers.
12	min() Returns the smallest of zero or more numbers.
13	pow() Returns base to the exponent power, that is, base exponent.
14	random() Returns a pseudo-random number between 0 and 1.
15	round() Returns the value of a number rounded to the nearest integer.
16	sin() Returns the sine of a number.
17	sqrt() Returns the square root of a number.
18	tan() Returns the tangent of a number.
19	toSource() Returns the string "Math".

Boolean Object

The **Boolean** object represents two values, either "true" or "false". 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.

Syntax

Use the following syntax to create a **boolean** object.

```
var val = new Boolean(value);
```

Boolean Methods

Here is a list of the methods of Boolean object and their description.

Sr.No	Method & Description
1	toSource() Returns a string containing the source of the Boolean object; you can use this string to create an equivalent object.
2	toString() Returns a string of either "true" or "false" depending upon the value of the object.
3	valueOf() Returns the primitive value of the Boolean object.

Global Object

- It provides top-level properties and methods that are not part of any other object
- You cannot create an instance of the Global object
- These methods are called directly and are not prefixed with "global"
- Globally Available Methods
 1. escape()
 2. unescape()

3. eval()
4. isFinite()
5. isNaN()
6. parseFloat()
7. parseInt()

escape()

- Takes a string and returns a string where all non-alphanumeric characters such as spaces, tabs, and special characters have been replaced with their hexadecimal equivalents in the form %xx

Example:

```
var aString="O'Neill & Sons";  
// aString = "O'Neill & Sons"  
aString = escape(aString);  
// String="O%27Neill%20%26%20Sons"
```

unescape ()

- Takes a hexadecimal string value containing some characters of the form %xx and returns the ISO-Latin-1 ASCII equivalent of the passed values

Example:

```
var aString="O%27Neill%20%26%20Sons";  
aString = unescape(aString);  
// aString = "O'Neill & Sons"  
aString = unescape("%64%56%26%23");  
// aString = "dV&#"
```

eval()

- Takes a string and executes it as JavaScript code

Example:

```
var x;  
var aString = "5+9";
```

```
x = aString;
// x contains the string "5+9"
x = eval(aString);
// x will contain the number 14
```

isFinite()

- Returns a Boolean indicating whether its number argument is finite or not

Example:

```
var x;
x = isFinite('56');
// x is true
x = isFinite(Infinity);
// x is false
```

isNaN()

- Returns a Boolean indicating whether its number argument is NaN

Example:

```
var x;
x = isNaN('56');
// x is False
x = isNaN(0/0);
// x is true
x = isNaN(NaN);
// x is true
```

parseInt()

- Converts the string argument to an integer and returns the value
- If the string cannot be converted, it returns NaN
- This method also should handle strings starting with numbers, but other mixed strings will not be converted

Example:

```
var x;  
x = parseInt("-53"); // x is -53  
x = parseInt("33.01568"); // x is 33  
x = parseInt("47.6k-red-dog"); // x is 47  
x = parseInt("a567.34"); // x is NaN  
x = parseInt("won't work"); // x is NaN
```

parseFloat()

- Converts the string argument to a floating-point number and returns the value
- If the string cannot be converted, it returns NaN
- The method should handle strings starting with numbers, but other mixed strings will not be converted

Example:

```
var x;  
x = parseFloat("33.01568"); // x is 33.01568  
x = parseFloat("47.6k-red-dog"); // x is 47.6  
x = parseFloat("a567.34"); // x is NaN  
x = parseFloat("won't work");// x is NaN
```

Date Object

- The Date object provides a sophisticated set of methods for manipulating dates and times
- To understand the relationship between Greenwich Mean Time (GMT), Coordinated Universal Time (UTC), and local time zones
- Creating Dates using Date() constructor

Syntax:

```
var myDate=new Date();
```

Manipulating Dates

- JavaScript provides a comprehensive set of **get** and **set** methods to read and write each field of a date
- The methods are
 - ✓ **getDate(), setDate(),**
 - ✓ **getMonth(), setMonth(),**
 - ✓ **getHours(), setHours(),**
 - ✓ **getMinutes(), setMinutes(),**
 - ✓ **getTime(), setTime,** and so on.
- In addition, UTC versions of all these methods are also included:
 - ✓ **getUTCMonth(), setUTCMonth()**
 - ✓ **getUTCHours(), setUTCHours(),** and so forth.
- One set of methods requires a special comment:
 - ✓ **getDay() and setDay().**
 - ✓ **These are used to manipulate the day of the week that is stored as an integer from 0 (Sunday) to 6 (Saturday).**

Example

```
<html>

<body>

<script type="text/javascript">

var today = new Date();

document.write("The current date : "+today+"<br />");

document.write("Date.getDate() : "+today.getDate()+"<br/>");

document.write("Date.getDay() : "+today.getDay()+"<br/>");

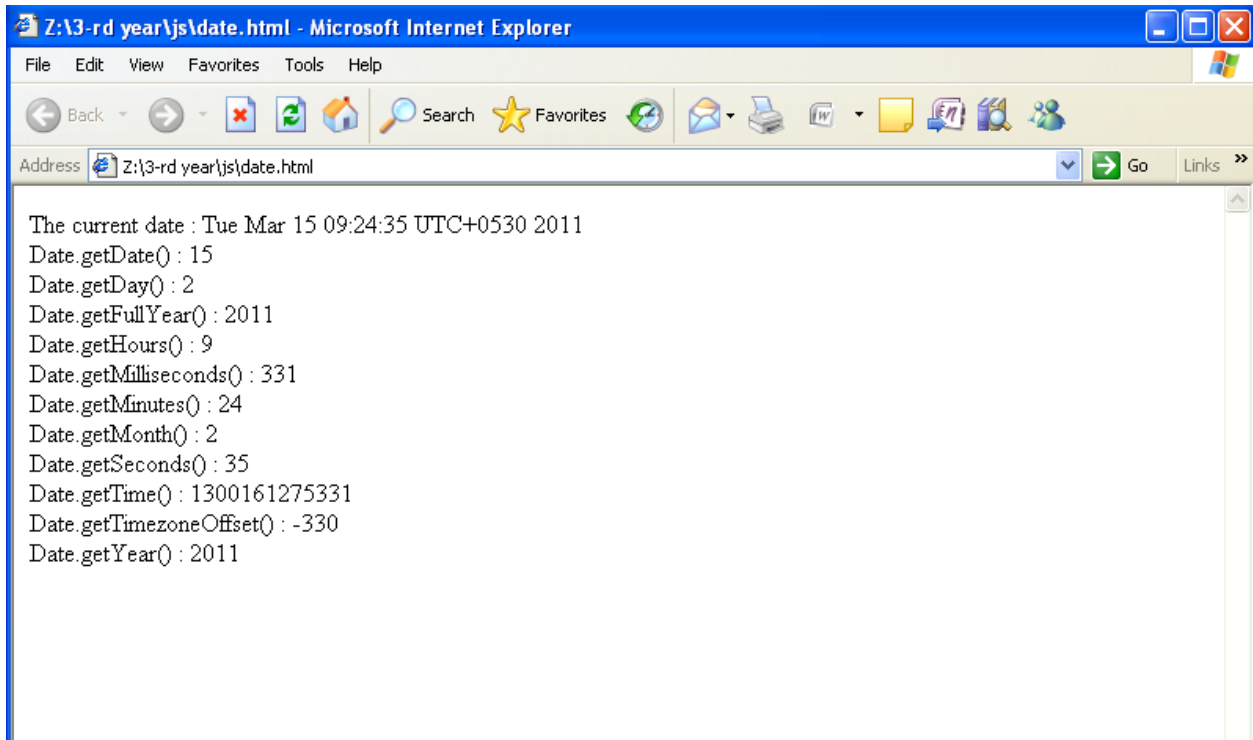
document.write("Date.getFullYear() : "+today.getFullYear()+"<br/>");

document.write("Date.getHours() : "+today.getHours()+"<br/>");
```



```
document.write("Date.getMilliseconds() : "+today.getMilliseconds()+"<br/>");  
  
document.write("Date.getMinutes() : "+today.getMinutes()+"<br/>");  
  
document.write("Date.getMonth() : "+today.getMonth()+"<br/>");  
  
document.write("Date.getSeconds() : "+today.getSeconds()+"<br/>");  
  
document.write("Date.getTime() : "+today.getTime()+"<br/>");  
  
document.write("Date.getTimezoneOffset() : "+today.getTimezoneOffset()+"<br/>");  
  
document.write("Date.getYear() : "+today.getYear()+"<br/>");  
  
</script>  
  
</body>  
  
</html>
```

Output



Number Object

- **Number is the built-in object corresponding to the primitive number data type**

Syntax

The syntax for creating a **number** object is as follows –

```
var val = new Number(number);
```

In the place of number, if you provide any non-number argument, then the argument cannot be converted into a number, it returns **NaN** (Not-a-Number).

Example:

```
var x = new Number(); // returns x=0  
var y = new Number(17.5); // y=17.5
```

Properties of the Number Object

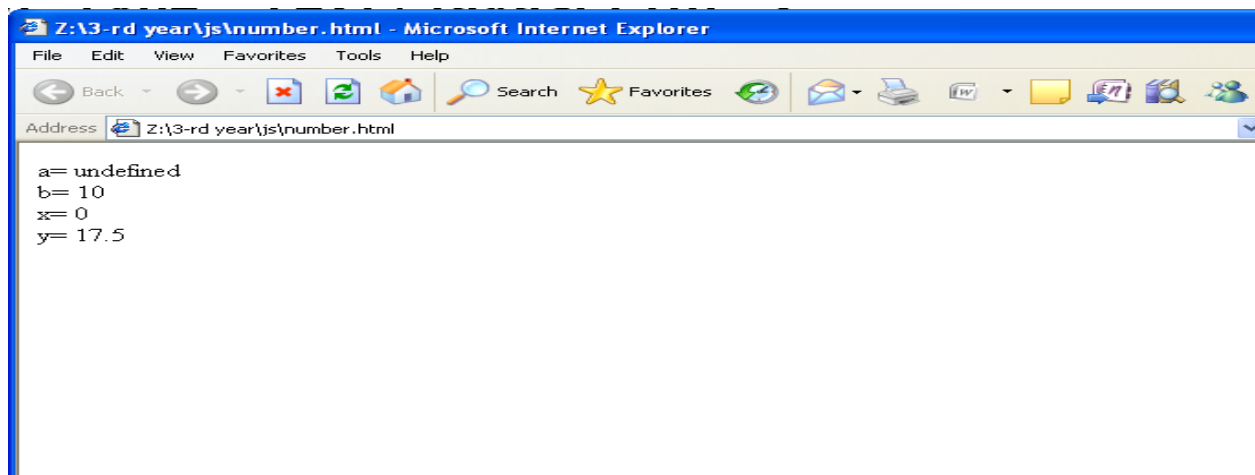
Property	Value
Number.MAX_VALUE	Largest magnitude representable
Number.MIN_VALUE	Smallest magnitude representable
Number.POSITIVE_INFINITY	The special value Infinity
Number.NEGATIVE_INFINITY	The special value -Infinity
Number.NaN	The special value NaN

Example

```
<html>  
<body>  
  <script type="text/javascript">  
    var a;  
    document.write("a= "+a+"<br>");  
    b=10;  
    document.write("b= "+b+"<br>");  
    var x = new Number();  
    document.write("x= "+x+"<br>");  
  </script>  
</body>  
</html>
```

```
var y = new Number(17.5);  
document.write("y= "+y);  
  
</script>  
</body>  
</html>
```

Output



Windows and Frames

- Window is an object that corresponds to the window that displays a Web page
- Such a window can be created dynamically

Properties

- ✓ These properties are assigned when creating the window.
- ✓ They value *yes* or *no* according to the state enabled or not.
- ✓ **scrollbars** -*yes* if scrollbars appear and *no* otherwise.
- ✓ **statusbar** -*yes* or *no*, depending on whether one shows the status bar or not.
- ✓ **toolbar** -*yes* or *no*, depending on whether one displays the toolbar or not.
- ✓ **menubar** -presenting a menu or not.

- ✓ **resizable** -to be able to change the size or not. The lower right corner is designed accordingly.
- ✓ **directories** -including buttons for favorites.

Attributes of window

- ✓ These attributes can be read only for some, their value can be assigned otherwise.
- ✓ **frames[]**-The frames in the window. Read only.
- ✓ **Length**-Number of frames. Read only.
- ✓ **Name**-Name of the window.
- ✓ **Status**-Text of the status bar
defaultStatus-Default text in statusbar.
- ✓ **Closed**-State closed or not.
- ✓ **Opener**-Reference on the window that opened this window. Example: x = window.opener;
- ✓ **Parent**-The window parent of a window. Example: x = mywin.parent;
- ✓ **Top**-The parent of the highest level.

Objects in window

- ✓ These objects have their own attributes and methods that are not detailed here but they have their own page.
- ✓ **document**-Refers to a page, that is contained in the window. [*Document*](#).
- ✓ **history**-List of pages previously viewed in the same window. [*History*](#).
- ✓ **location**-Designates the URL of a page that contains the window. [*Location*](#).
- ✓ **screen**-The screen and its properties: width, height, availWidth, availHeight, colorDepth.

Window Creation

- The **Window** object methods **open()** and **close()** are used to create and destroy a window, respectively.
- When you open a window, you can set its URL, name, size, buttons, and other attributes, such as whether or not the window can be resized.

Syntax:

- **window.open**(url, name, features, replace) ;
- Where
 - *url* is a URL that indicates the document to load into the window.
 - *name* is the name for the window
 - *features* is a comma-delimited string that lists the features of the window.
 - *replace* is an optional Boolean value (**true** or **false**) that indicates if the URL specified should replace the window's contents or not.

Example

```
secondwindow = open("http://www.yahoo.com", "yahoo",  
"height=300,width=200, scrollbars=yes");
```

Example Program

```
<html>  
  
  <head>  
  
    <script language="JavaScript">  
  
      function openWin()  
  
        {  
  
          myWin = open("window.html");  
  
        }  
  
      function closeWin()  
  
        {
```

```
        myWin.close();
    }
</script>
</head>
<body>
<form>
    <input type = "button" value = "Open new Window" onClick="openWin()">
    <input type = "button" value = "Close Window" onClick="closeWin()">
</form>
</body>
</html>
```

Forms and Validation

```
<html>
<head><title>Form validation</title>
<script language="javascript">
var s1,s2,s3,s4,s5,s6,s7,s8;
function fun1()
{
if(form1.name.value=="")
{
alert("enter the name");
form1.name.focus();
}
else
{
var l=form1.name.value.length;
s1=new String(form1.name.value);
for(i=0;i<l;i++)
```

```
{
if(!((s1.charAt(i)>='a'||s1.charAt(i)>='A') && (s1.charAt(i)<='z'||s1.charAt(i)<='Z'))))
{
alert("Please enter the correct name");
form1.name.focus();
break;
}
}
}
}
function fun2()
{
if(form1.email.value=="")
{
alert("enter the email");
form1.email.focus();
}
else
{
var c1=0;
var c2=0;
var l=form1.email.value.length;
s2=new String(form1.email.value);
for(i=0;i<=l;i++)
{
if(s2.charAt(i)=='@')
{
c1=c1+1;
var j=i;
}
}
if(c1==1)
{
for(i=j;i<=l;i++)
{
if(s2.charAt(i)=='.')
c2=c2+1;
}
}
```

```
}
if(!((c1==1) && (c2==1)))
{
alert("enter the correct e-mail id");
form1.email.focus();
}
}
}
function fun3()
{
s3=form1.age.value;
if((s3=="")||isNaN(s3)) //isNaN() function determines whether a value is an illegal number
{
alert("enter the age");
form1.age.focus();
}
}
function fun4()
{
s4=form1.addrs.value;
if(s4=="")
{
alert("enter the address");
form1.addrs.focus();
}
}
function fun5()
{
s5=form1.phno.value;
if((s5=="") || isNaN(s5))
{
alert("Invalid phone number");
form1.phno.focus();
}
}
function fun6()
{
var tag1=0;
```



```
for(var i=0;i<form1.bank.length;i++)
{
if (form1.bank[i].checked==true)
{
s6=form1.bank[i].value;
tag1=1;
}
}
if(!(tag1==1))
{
alert("select ur Bank");
}
}
function fun7()
{
s7=form1.card.value;
if((s7=="") || isNaN(s7))
{
alert("Invalid card number");
form1.card.focus();
}
}
function fun8()
{
s8=form1.pin.value;
var l=form1.pin.value.length;
if(((s8=="") || isNaN(s8))&&(l==3))
{
alert("Invalid card number");
form1.pin.focus();
}
}
function fun9()
{
w1=window.open('Order Conformation.html');
w1.document.writeln("<h1><center>");
w1.document.writeln("Order Conformation Details");
w1.document.writeln("</h1></center>");
```

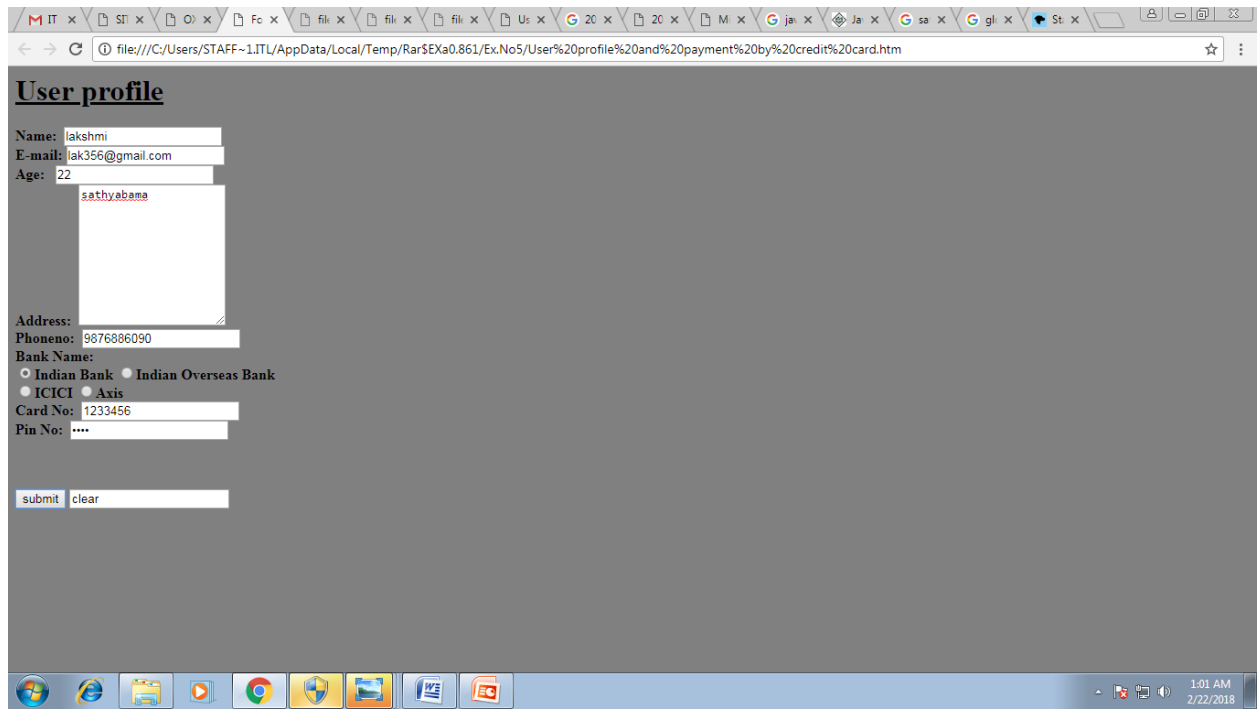
```

w1.document.writeln("<br>");
w1.document.writeln("Name:",s1);
w1.document.writeln("<br>");
w1.document.writeln("Email:",s2);
w1.document.writeln("<br>");
w1.document.writeln("age:",s3);
w1.document.writeln("<br>");
w1.document.writeln("Address:",s4);
w1.document.writeln("<br>");
w1.document.writeln("Phone:",s5);
w1.document.writeln("<br>");
w1.document.writeln("Bank Name:",s6);
w1.document.writeln("<br>");
w1.document.writeln("Card Number:",s7);
w1.document.writeln("<br>");
}
</script></head>
<body bgcolor="gray" >
<h1><b><u> User profile </u></b></h1>
<form name="form1" method="get">
<b>
Name:&nbsp;&nbsp;&nbsp;<input type="text" name="name" maxlen="5"><br>
E-mail:&nbsp;&nbsp;&nbsp;<input type="text" name="email" onfocus="fun1()"><br>
Age:&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<input type="text" name="age" onfocus="fun2()"><br>
Address:&nbsp;&nbsp;&nbsp;<textarea onfocus="fun3()" rows=10 cols=20
name=addr></textarea><br>
Phoneno:&nbsp;&nbsp;&nbsp;<input type="text" name="phno" onfocus="fun4()"><br>
Bank Name:<br>
<input type="radio" onfocus="fun5()" name="bank" value="Indian Bank">Indian Bank
<input type="radio" onfocus="fun5()" name="bank" value="Indian Overseas Bank">Indian
Overseas Bank <br>
<input type="radio" onfocus="fun5()" name="bank" value="ICICI">ICICI
<input type="radio" onfocus="fun5()" name="bank" value="Axis">Axis <br>
Card No:&nbsp;&nbsp;&nbsp;<input type="text" name="card" onfocus="fun6()"><br>
Pin No:&nbsp;&nbsp;&nbsp;<input type="password" name="pin" onfocus="fun7()"
"><br><br><br><br>
<input type="button" value="submit" onfocus="fun8()" onclick="fun9()">
<input type="reset" value="clear">
</b>

```

```
</form>
</body>
</html>
```

Output



The screenshot shows a web browser window with a single tab. The address bar contains the file path: `file:///C:/Users/STAFF~1/ITL/AppData/Local/Temp/Rar$EXa0.861/Ex.No5/User%20profile%20and%20payment%20by%20credit%20card.htm`. The page title is "User profile". The form contains the following fields and options:

- Name:
- E-mail:
- Age:
- (This field is highlighted with a red border)
- Address:
- Phoneno:
- Bank Name:
 - Indian Bank
 - Indian Overseas Bank
 - ICICI
 - Axis
- Card No:
- Pin No:
- submit

The Windows taskbar at the bottom shows the system tray with the time 1:01 AM and date 2/22/2018.