Refresher Assignment

JavaScript:

Variables: like any other languages, variables are used to store the data(i.e. strings or numbers) to perform some arithmetic operations or string operations that can be used in Java script programming.

Question: Write a program to calculate the area of a rectangle.

<!DOCTYPE html>

<html>

<head>

<meta charset=*"ISO-8859-1"*>

<title>Variables</title>

</head>

<body>

<h2> Area of a rectangle </h2>

<p id=*"demo"*> </p>

<script>

**var** length = 9;

**var** breadth = 10;

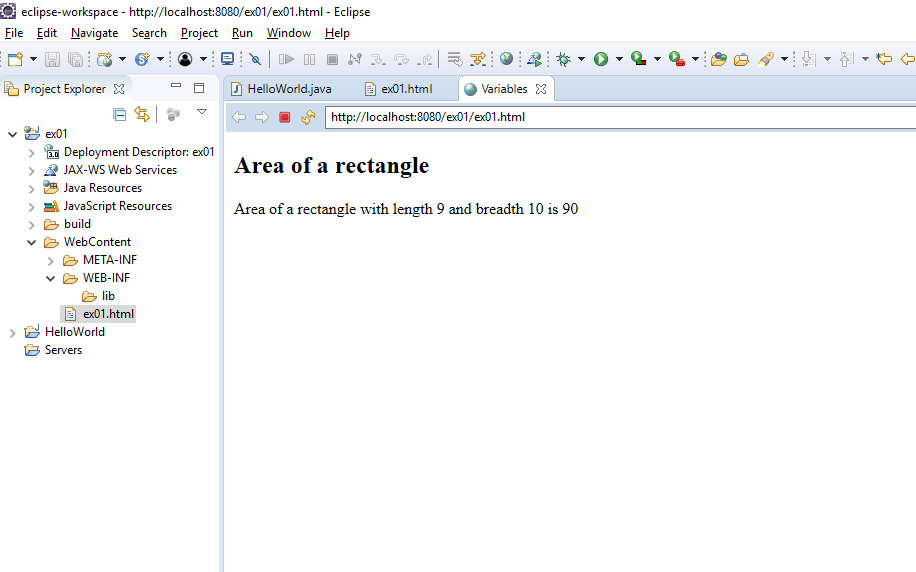
**var** area = length \* breadth;

document.getElementById("demo").innerHTML = "Area of a rectangle with length 9 and breadth 10 is " + area;

</script>

</body>

</html>



Objects: Objects are also variables that can have more than one name, value pairs.

Question: write a program to display my address from the object.

<!DOCTYPE html>

<html>

<head>

<meta charset=*"ISO-8859-1"*>

<title>Objects</title>

</head>

<body>

<h2>Display my address</h2>

<p id=*"demo"*></p>

<script>

**var** home = {

Street: "1180 Lochinvar Avenue",

Apt : 134,

City : "Sunnyvale",

State : "California",

Pincode: 94087

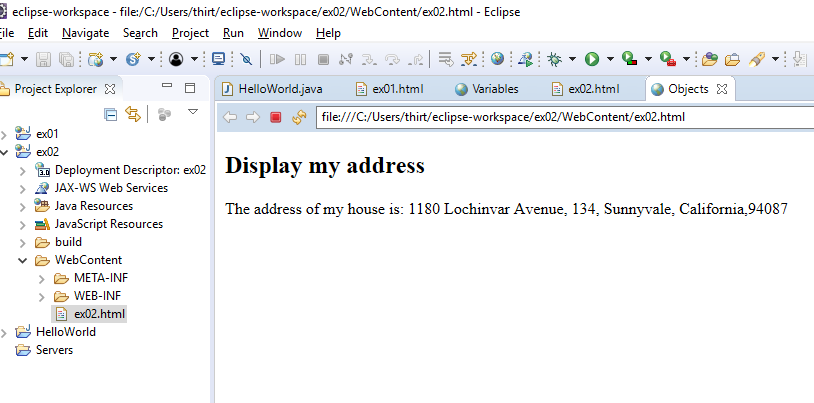
};

document.getElementById("demo").innerHTML = "The address of my house is: "+home.Street+ ", " + home.Apt+ ", " +home.City+ ", " + home.State + "," + home.Pincode;

</script>

</body>

</html>



Functions: Function is a set of instructions to be executed, when it is called or invoked by the code or invoked by itself.

Question: Often when we are buying a car or leasing it, car dealers play with the numbers (i.e increase down payment decrease monthly lease). write a program with a function where the function calculates effective lease price for a car every month considering the down payment paid.

<!DOCTYPE html>

<html>

<head>

<meta charset=*"ISO-8859-1"*>

<title>Function</title>

</head>

<body>

<h2>My effective lease price per month</h2>

<p id=*"demo"*></p>

<script>

**function** myFunction(dp, lease, numOfMons)

{

**var** effLease = lease+ dp / numOfMons;

**return** effLease;

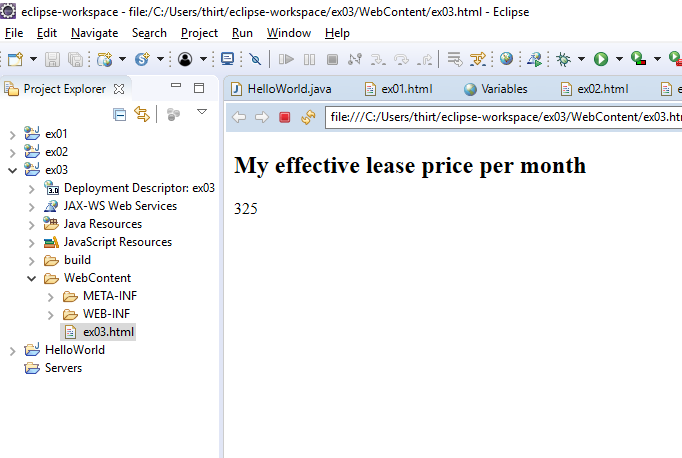
}

document.getElementById("demo").innerHTML = myFunction(2500, 200, 20);

</script>

</body>

</html>



Events: We can write code in Javascript and let it execute when an action is taken on HTML elements(i.e called an HTML event). Some of the HTML events are onclick, onmouseover, etc.

Question: Write a program using onclick and onmouseover to display the sum of two numbers and to demonstrate the difference between them in the output of the code.

<!DOCTYPE html>

<html>

<head>

<meta charset=*"ISO-8859-1"*>

<title>Events</title>

</head>

<body>

<h2>Onclick and Onmouseover</h2>

<button onclick="document.getElementById('demo').innerHTML=myfunction(2,3)">Click to find sum</button>

//displays sum of 2 and 3 when clicked

<p id=*"demo"* onmouseover="document.getElementById('demo').innerHTML=myfunction(3,4)"

onmouseout ="document.getElementById('demo').innerHTML='Move over to find sum'">Move over to find the sum</p>

//displays sum of 3 and 4 when moved over and comes back displaying message when moved out

<script>

**function** myfunction(a,b)

{

**return** a+b;

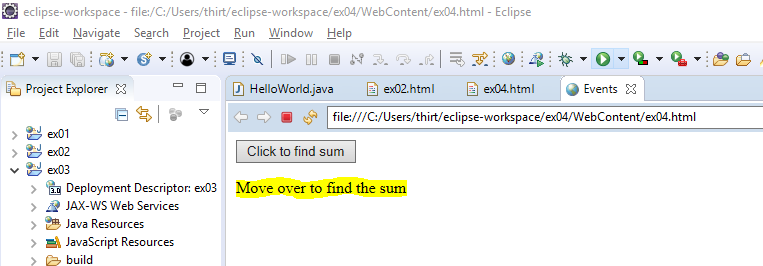
}

</script>

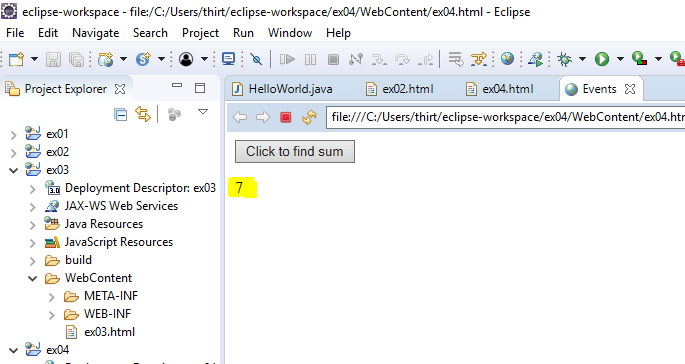
</body>

</html>

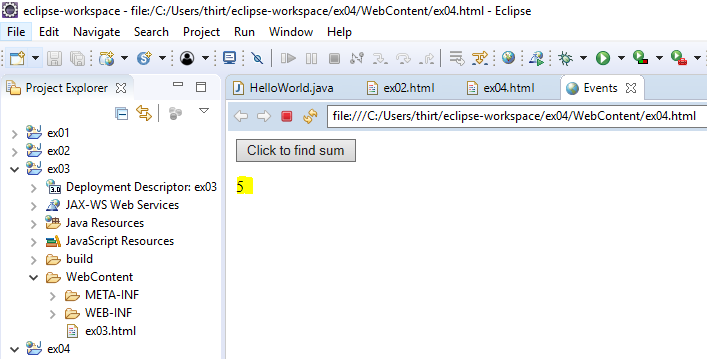
Output for onmouseover(before moving mouse over field)



Output for onmouseover(after moving mouse over field)



Output for onclick



Arrays: Arrays are also variables like objects that can have more than one different types of values at a time. Arrays can store a function, an object or an array itself.

Question: Write a program of array that has objects name and salary of employer. Display the salaries of the employers and sort them in ascending order.

<!DOCTYPE html>

<html>

<head>

<meta charset=*"ISO-8859-1"*>

<title>ARRAYS</title>

</head>

<body>

<h2>Calculating the salaries of the employers</h2>

<p>click the button to sort the salaries</p>

<button onclick="sortsalaries()">sort</button>

<p id=*'demo'*></p>

<script>

**var** sal = [

{Name:"Joe", Salary:1800},

{Name:"Tom", Salary:800},

{Name:"Kyle", Salary:100},

{Name:"Jim", Salary:1100}

]

displaysalaries();

**function** sortsalaries()

{

sal.sort(**function**(a,b){**return** a.Salary-b.Salary});

displaysalaries();

}

**function** displaysalaries()

{

document.getElementById("demo").innerHTML =

sal[0].Name + " " + sal[0].Salary + "<br>" +

sal[1].Name + " " + sal[1].Salary + "<br>" +

sal[2].Name + " " + sal[2].Salary + "<br>" +

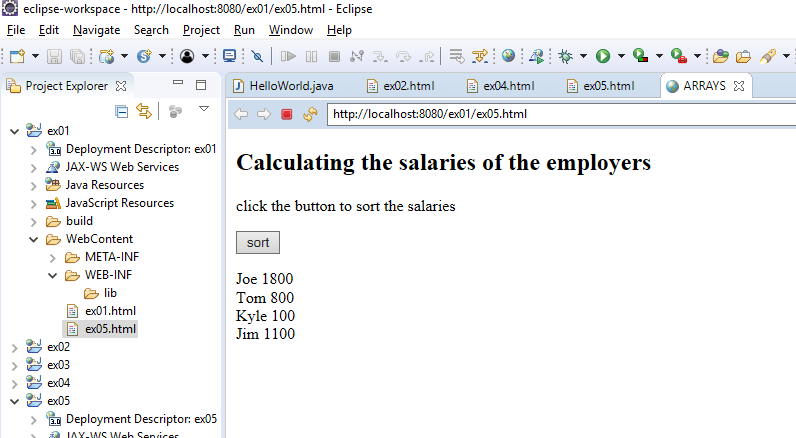
sal[3].Name + " " + sal[3].Salary;

}

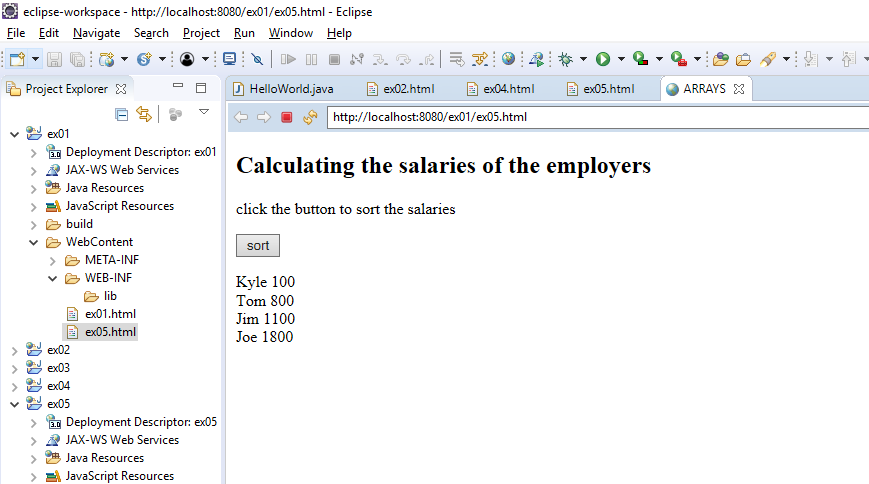
</script>

</body>

</html>



After sorting



Inheritance: In java script, prototypal inheritance is when an object is based on another object and class-based inheritance is based on another class. using the same implementation (inheriting from an object or class) specifying implementation to maintain the same behavior (realizing an interface; inheriting behavior).

Question: write a program to demonstrate the prototypical inheritance. Create object student and teacher and inherit the student into the teacher. Display them using a function.

<!DOCTYPE html>

<html>

<head>

<meta charset=*"ISO-8859-1"*>

<title>Insert title here</title>

</head>

<body>

<script>

**var** student = **function**()

{

**this**.name = "studentA from parent"; //creating object student

}

student.prototype.print = **function**() {

document.write(**this**.name+ "<br>"); //display student

}

**var** student1 = **new** student();

student1.print();

**var** inherit = **function** (child, parent){

child.prototype = Object.create(parent.prototype); //prototypical inheritance

};

**var** teacher = **function** ()

{

**this**.name = "course SE"

**this**.source = "derived from student";

}

inherit(teacher, student);

**var** teacher1 = **new** teacher();

teacher1.print(); // print the course name of teacher

teacher.prototype.print = **function**() {

student.prototype.print.call(**this**);

document.write(**this**.source+ "<br>"); //inheritance of student into teacher

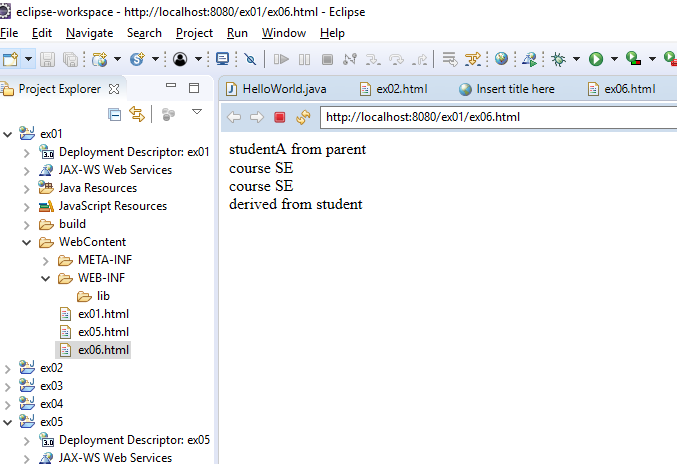
}

teacher1.print();

student1.print();

</script>

</body>

</html>

Conditions: conditional statements are used to specify the JS code to execute a particular piece of code when a condition is met. Like any other languages, if, else, else if statements are used in Javascript.

Question: Write a program to check if the salary of the employee is greater than 1000. If yes, calculate the tax as 30% of salary else the tax is 0%.

<!DOCTYPE html>

<html>

<head>

<meta charset=*"ISO-8859-1"*>

<title>Conditions</title>

</head>

<body>

<h2>Calculating the tax if the salary is greater than 1000</h2>

<p> Enter the salary </p>

<input id=*"sal"*>

<button type=*"button"* onclick="CalculateTax()">Calculate Tax</button>

<p id=*"demo"*></p>

<script>

**function** CalculateTax()

{

**var** x, tax;

x = document.getElementById("sal").value;

**if** (x > 1000)

{

tax = 0.3 \* x ;

}

**else**

{

tax = 0;

//document.getElementById("demo").innerHTML = tax;

}

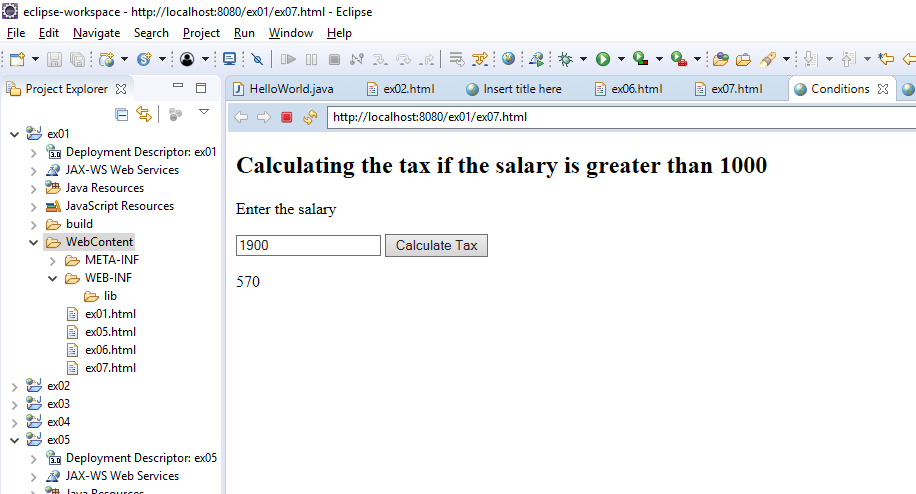
document.getElementById("demo").innerHTML = tax;

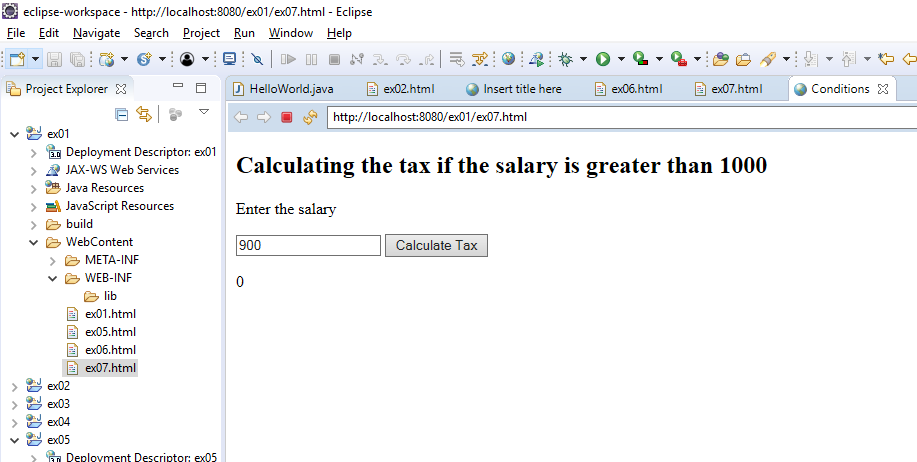
}

</script>

</body>

</html>





Regular Expressions are patterns that are formed to perform various search and replace operations and the advantage of using it is that it can be case insensitive and perform multiline matching making it easier for coder. The various pattern includes brackets, metacharacters and quantifiers.

Question: Write a program to do a global search and replace more than one string “it” with “Javascript”.

<!DOCTYPE html>

<html>

<head>

<meta charset=*"ISO-8859-1"*>

<title>Regular Expressions</title>

</head>

<body>

<h2>JavaScript Regular Expressions</h2>

<p>Replace the word "it" with "JavaScript" using regular expressions</p>

<button onclick="replace()">Click to replace</button>

<p id=*demo*>It is used to make webpages interactive. It also provides online programs including video games.</p>

<script>

**function** replace()

{

**var** str, restr;

str = document.getElementById("demo").innerHTML;

restr = str.replace(/it/ig,"JavaScript"); //global search

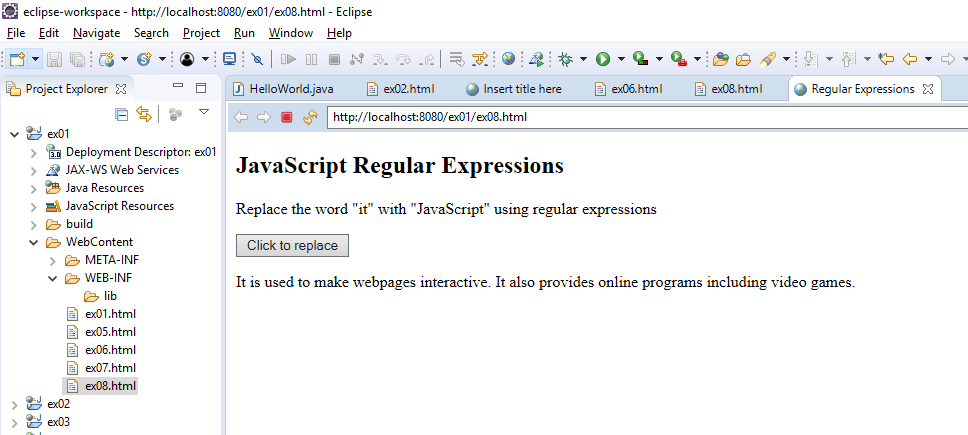
document.getElementById("demo").innerHTML = restr;

}

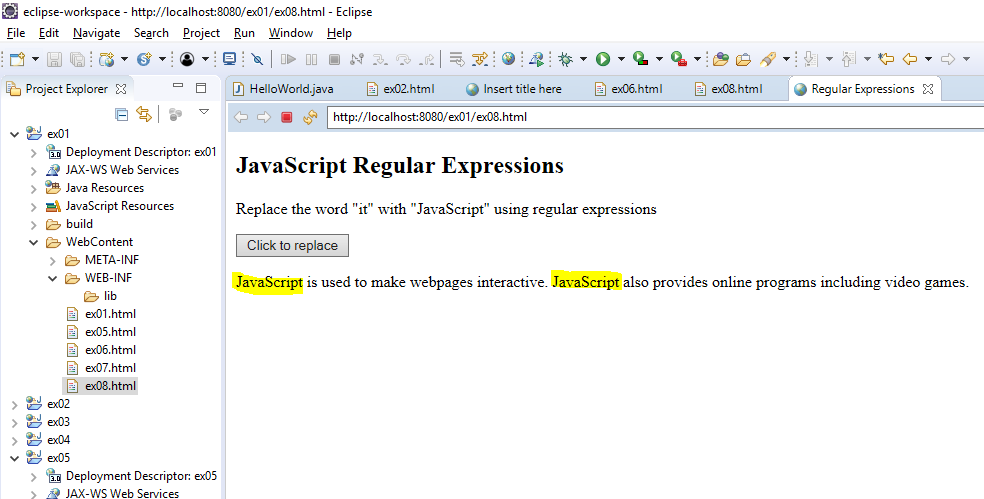
</script>

</body>

</html>



After clicking the replace button:



Strict mode is used inside the script block to restrict the use of undeclared variables. This mode is available in the newer version of the Javascript which helps the programmer to avoid various syntax errors and securely write the code.

Write a program to demonstrate the use of strict mode. Using function and duplicating the parameters.

<!DOCTYPE html>

<html>

<head>

<meta charset=*"ISO-8859-1"*>

<title>strict mode</title>

</head>

<body>

<h2>Script mode is ON</h2>

<p id=*"demo"*></p>

<script>

"use strict";

**function** install(a, a)

{

**return** a+a;

} //duplicating the parameters of function is not allowed when strict mode is on

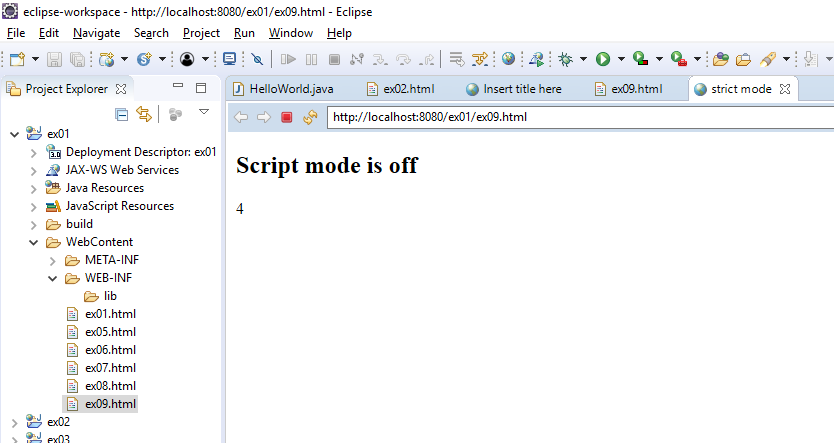
document.getElementById("demo").innerHTML = install(2,2);

</script>

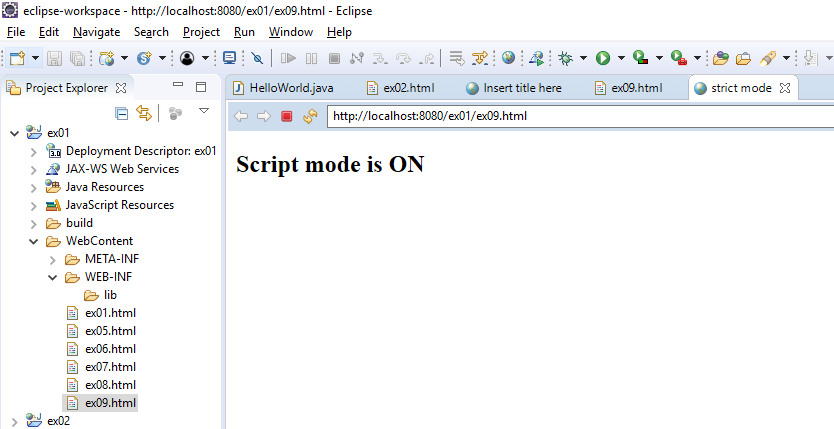
</body>

</html>

When script mode is off, you get an output.



When script mode is on, it does not allow duplicating the parameters of the function.



Errors: Java script uses try, catch, throw and finally statements to handle the different errors that occur when executing the code. The try statement is where you write your code and in catch statement you write a piece of code to handle the error that occurred while executing the code written inside try block. JavaScript has some built in errors like EvalError, RangeError, SyntaxError, etc, In addition to these, when you want to create your error messages, throw statement is used. Finally statement is used when you want to execute a code after try and catch block.

Write a program to do string validation. The program should throw an error if the string length is not between 5 and 10.

<!DOCTYPE html>

<html>

<head>

<meta charset=*"ISO-8859-1"*>

<title>Errors</title>

</head>

<body>

<h2>JavaScript Errors</h2>

<p>Please input a string of minimum five characters and maximum ten characters:</p>

<input id=*"demo"* type=*"text"*>

<button type=*"button"* onclick="TestLength()">Test Input</button>

<p id=*"message"*></p>

<script>

**function** TestLength() {

**var** message, x, y;

message = document.getElementById("message");

x = document.getElementById("demo").value;

**try** {

x = String(x);

y = x.length;

**if**(y == "") **throw** "empty";

**if**(y < 5) **throw** "too low";

**if**(y > 10) **throw** "too high";

}

**catch**(e) {

message.innerHTML = "The string is " + e;

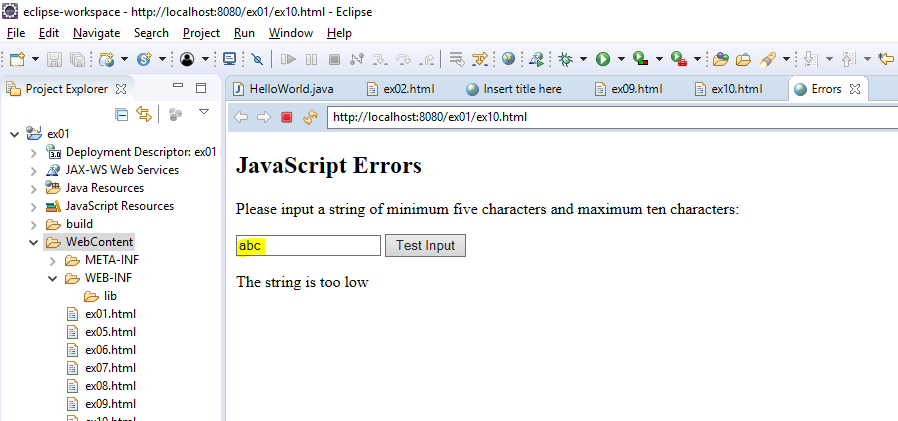
}

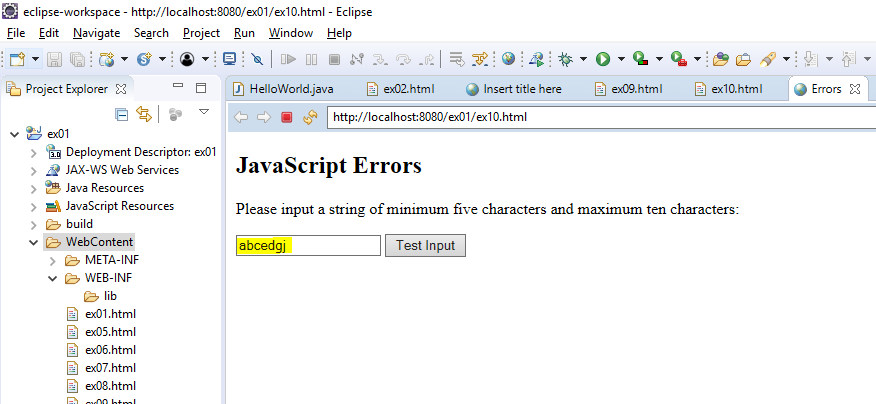
}

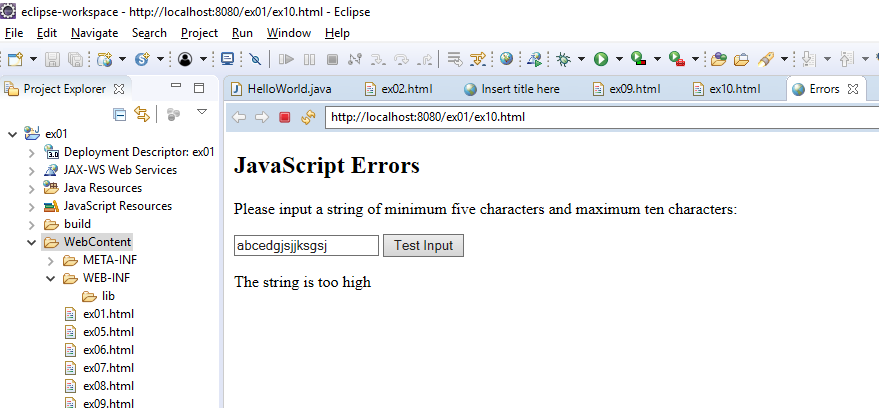
</script>

</body>

</html>







Type Conversions: are used to convert a data type into a different data type. Java script tends to do auto type conversion i.e. correct the wrong data types in some operations like sum, multiplication, etc that might give an output that is not expected.

Write a program to demonstrate the type of operator and some examples of type conversion.

<!DOCTYPE html>

<html>

<head>

<meta charset=*"ISO-8859-1"*>

<title>Insert title here</title>

</head>

<body>

<h2>The JavaScript Type Conversion</h2>

<p>The typeof operator returns the type of a variable or expression.</p>

<button onclick="typeOfOp()">Click</button>

<p id=*"demo"*></p>

<script>

**function** typeOfOp() {

**var** y = "55";

**var** x = + y;

**var** a = y + x;

**var** b = Number(**false**);

**var** c = y + **false**; //since value of false is 0

**var** d = x + **false**;

document.getElementById("demo").innerHTML =

"value of x " + x + ' ' + "and data type is " + **typeof** x + "<br>" +

"value of y " + y + ' ' + "and data type is " + **typeof** y + "<br>" +

"value of a " + a + ' ' + "and data type is " + **typeof** a + "<br>" +

"value of b " + b + ' ' + "and data type is " + **typeof** b + "<br>" +

"value of c " + c + ' ' + "and data type is " + **typeof** c + "<br>" +

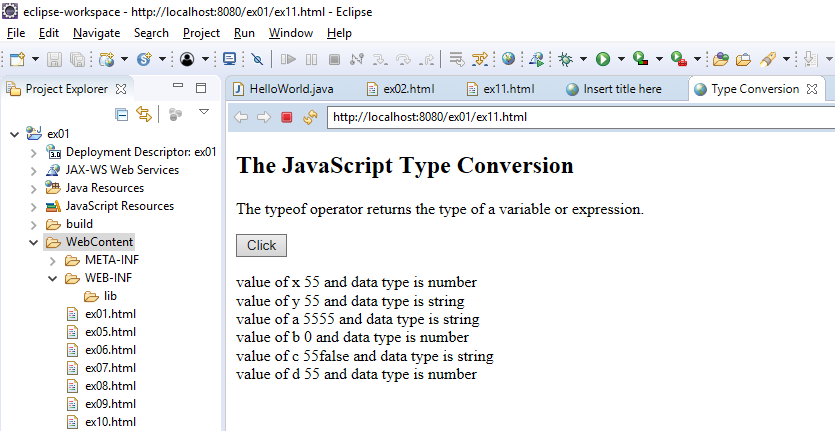
"value of d " + d + ' ' + "and data type is " + **typeof** d ;

}

</script>

</body>

</html>



JSON is an abbreviation for JavaScript Object Notation and is language independent. JSON is used to store and move data from server to web page.

Create a JSON object of students in a class and create a function to display their name, ID and course as a string.

<!DOCTYPE html>

<html>

<head>

<meta charset=*"ISO-8859-1"*>

<title>JSON</title>

</head>

<body>

<h2>Using JSON Object</h2>

<p id=*"demo"*></p>

<script>

**var** str = '{"students":[' +

'{"firstName":"Harvey","lastName":"Specter","ID":"01", "course": "SE" },' +

'{"firstName":"Mike","lastName":"Ross" ,"ID":"02", "course": "CE" },' +

'{"firstName":"Rachel","lastName":"Zane","ID":"03", "course": "CS" }]}';

A = JSON.parse(str);

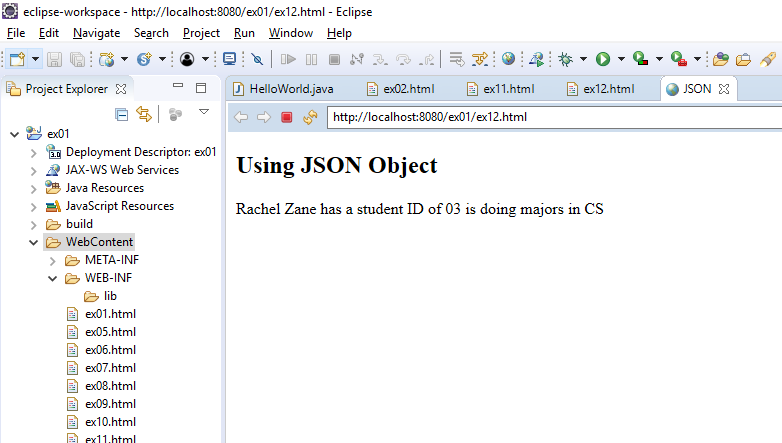
document.getElementById("demo").innerHTML =

A.students[2].firstName + " " + A.students[2].lastName + " has a student ID of " + A.students[2].ID + " is doing majors in " + A.students[2].course;

</script>

</body>

</html>



**HTML5:**

Local Storage: The data is stored locally in our browser and is never sent to server. Unlike cookies, we can store more data locally and securely without interrupting the performance of the site.

Write a program to save my last name and favorite destination spot.

<!DOCTYPE html>

<html>

<body>

<div id=*"result"*></div>

<script>

// Check browser support

**if** (**typeof**(Storage) !== "undefined") {

// Store

localStorage.setItem("lastname", "Smith");

localStorage.setItem("Spot", "Hawaii");

localStorage.setItem("Address", "SJSU University");

localStorage.setItem("Phone", "0123456789");

console.log(localStorage.length);

// Retrieve

// Alert( "Hi " + localStorage.getItem("lastname"));

document.getElementById("result").innerHTML = "Hi " + localStorage.getItem("lastname") + " I remember that your favourite destination is " + localStorage.getItem("Spot") + "<br>" + "Using your address and contact to look for hotels " + "<br>" +

localStorage.getItem("Address") + " and contact " + localStorage.getItem("Phone");

} **else** {

document.getElementById("result").innerHTML = "Sorry, your browser does not support Web Storage...";

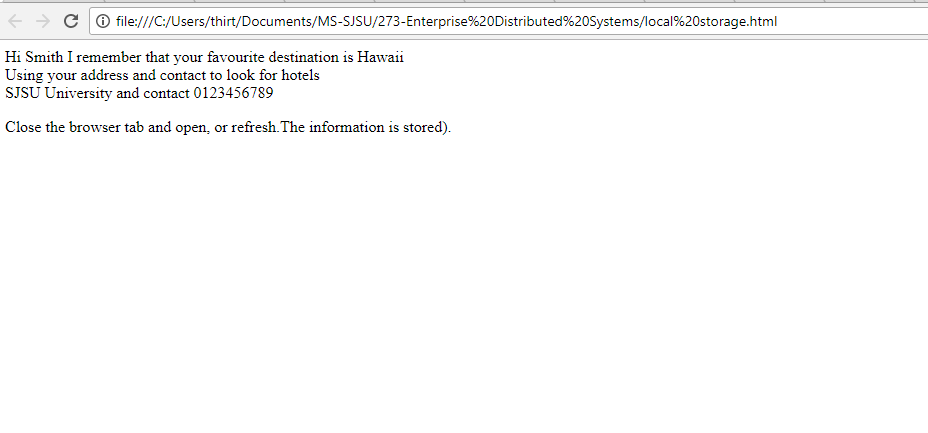
}

</script>

<p>Close the browser tab and open, or refresh.The information is stored.</p>

</body>

</html>



Note: using browser for HTML5 since output was not displayed in eclipse.

Media (Video and Audio):

Anything that we see or hear on web is called Multimedia. These multimedia files are stored in media in various formats. The type of format is identified by the extensions at the end of the file name.

Audio : There are some Audio controls that are used to play the media on the browser.

Write a program to play the audio

<html>

<head> click to play you favourite</head>

<body>

<audio controls>

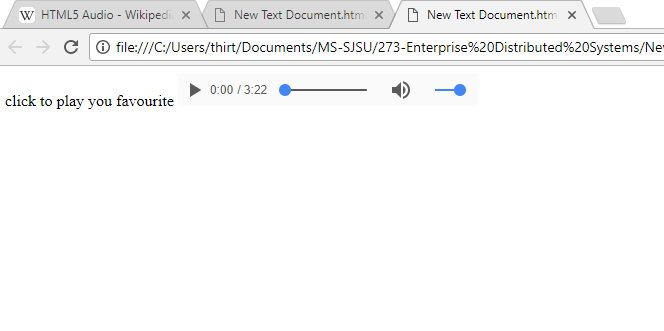
<source src="C:\Users\thirt\Downloads\audio1.mp3" type="audio/mpeg">

Your browser does not support the audio element. //this will only be displayed when the browser doesnt support.

</audio>

</body>

</html>



Video: the HTML5 <video> elements lets standard way to play the video on browser.

Write a program to embedded a video into browser.

<!DOCTYPE html>

<html>

<h2> Video </h2>

<body>

<video width="700" controls>

<source src="C:\Users\thirt\Downloads\SOMETIMES YOURE THE PROBLEM.mp4" type="video/mp4">

Your browser does not support HTML5 video.

</video>

<p>

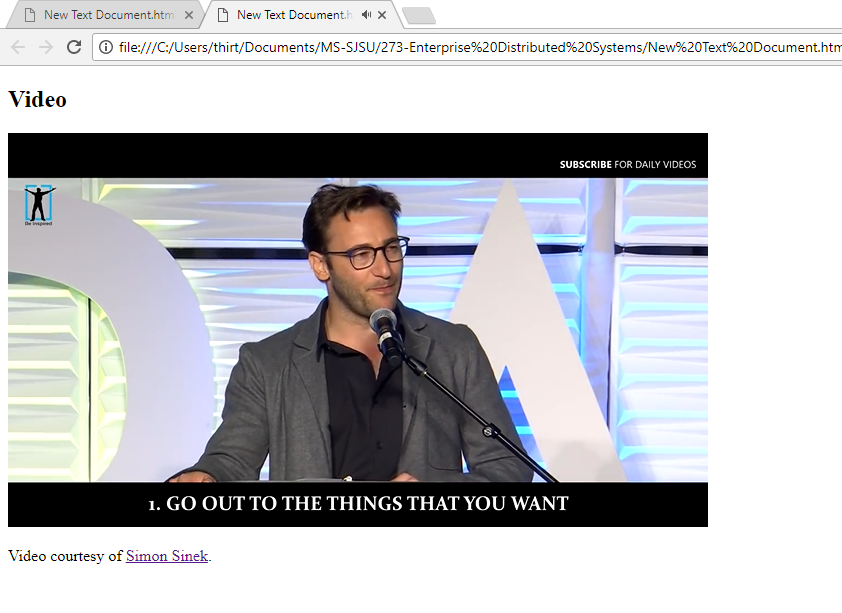
Video courtesy of

<a href="https://www.youtube.com/watch?v=XfP-SBD00hY&t=4s"\_blank">Simon Sinek</a>.

</p>

</body>

</html>



Input Type (make use of different input property options in HTML5 like

patterns,

autofocus, required, email etc. Place types you want, mention the properties used in your

Introduction to Topic section) :

Text: defines text. Lets the user enter single line text.

Password: lets user enter password. The characters are masked with circles.

Requires: lets user know that entering the field is mandatory.

Submit: defines a button for submitting.

Radio: defines the radio button where it lets the user select only one option.

Button: defines a button.

Date: lets user enter date field.

Email: lets user enter email id.

Tel: lets user enter phone number. But I have used pattern and let it accept pincode.

Pattern: is a regular expression that the input is checked against and alerts the user to enter the value that’s allowed.

Autofocus: focuses the attribute as soon as the page loads.

Formtarget: lets you specify where you want to display the output received after loading the page.

Formaction: lets you specify a URL that it should be directed to when the form is submitted.

Write a program to create a signup page for an online shopping website.

<!DOCTYPE html>

<html>

<body>

<form action="input.html">

Enter the user name: <br>

<input type="text" name="username" autofocus><br> //autofocus, text

Enter the password: <br>

<input type="password" name="pswd" required><br> //password, required

Enter your date of birth:<br>

<input type="date" name="DOB"><br> //date

Enter the pincode: <br>

<input type="tel" name="country\_code" pattern="[0-9]{5}" title="Five letter Pincode"><br><br> //pattern

<input type="submit" value="Submit" > <br> //submit

<input type="submit" formtarget="\_blank" value="Submit to a open in a new tab"><br>

<input type="submit" formaction="https://en.wikipedia.org/wiki/HTML5\_Audio" value="Click to learn about HTML5 Audio"><br>

Enter the email ID: <br>

<input type="email" name="email"><br> //email

<input type="button" onclick="alert('email is entered')" value="Click Me!"> <br>

<input type="radio" name="subscribe" value="yes"> I subscribe for newsletters<br>

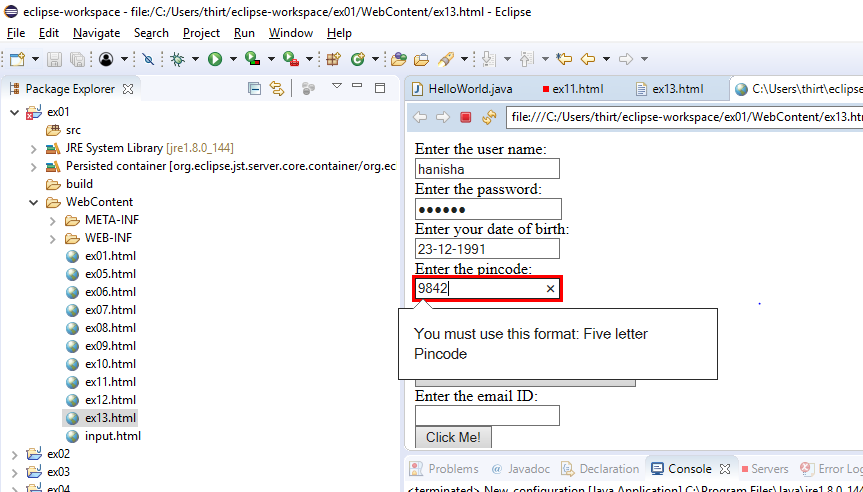
<input type="radio" name="unsubscribe" value="No"> No, thank you! //radio button

</form>

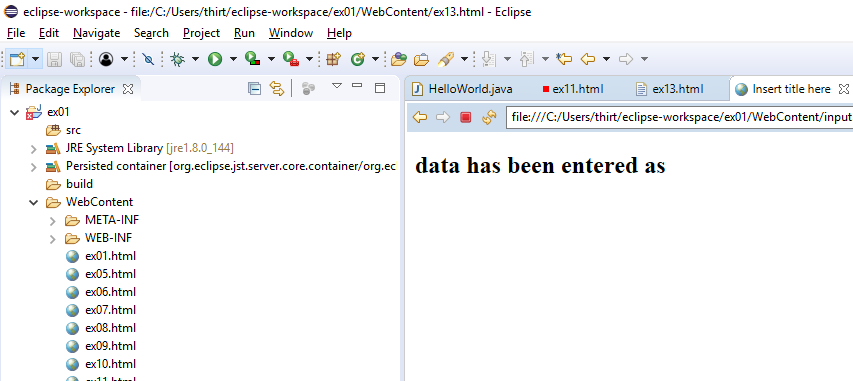
<p><strong>Note:</strong> type= Do not use Internet Explorer 11 and earlier versions.</p>

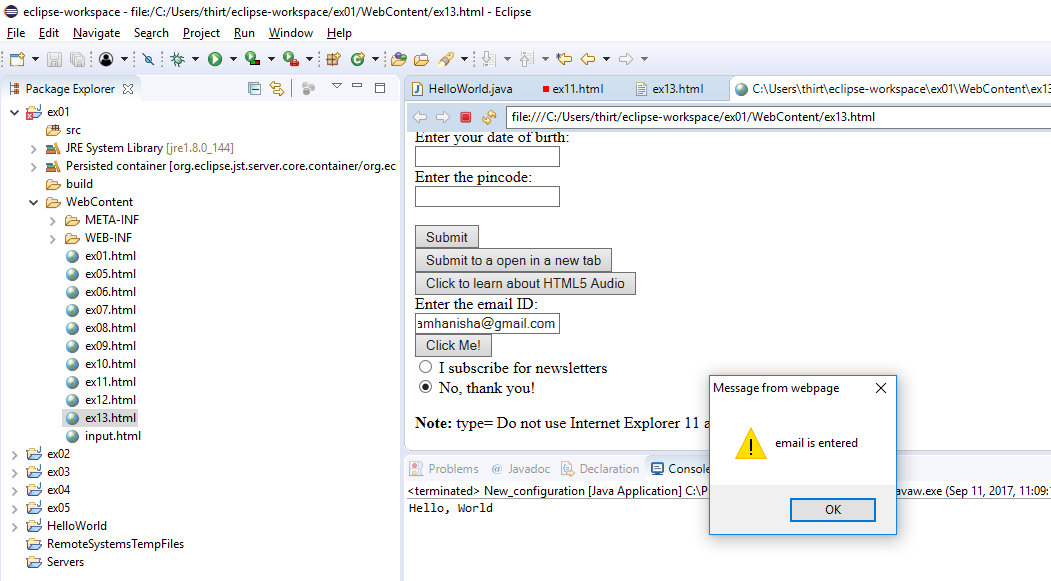
</body>

</html>



After submitting goes to input.html





Geolocation: is a HTML API used to find the current geographical location of a user. It’s used with the method getCurrentPosition().

<!DOCTYPE html>

<html>

<body>

<p>Click the button to find the location.</p>

<button onclick="getCoords()">Track</button>

<p id="demo"></p>

<script>

var x = document.getElementById("demo");

function getCoords() {

if (navigator.geolocation) {

navigator.geolocation.getCurrentPosition(DisplayPos, DisplayErr);

} else {

document.getElementById("demo").innerHTML = "Geolocation is not supported by this browser.";

}

}

function DisplayPos(POS) {

document.getElementById("demo").innerHTML = "Latitude: " + POS.coords.latitude +

"<br>" + "Longitude: " + POS.coords.longitude;

}

function DisplayErr(err)

{

switch(error.code) {

case error.PERMISSION\_DENIED:

document.getElementById("demo").innerHTML = "User didnot approve the Geolocation."

break;

case error.UNKNOWN\_ERROR:

document.getElementById("demo").innerHTML = "An unknown error occurred."

break;

}

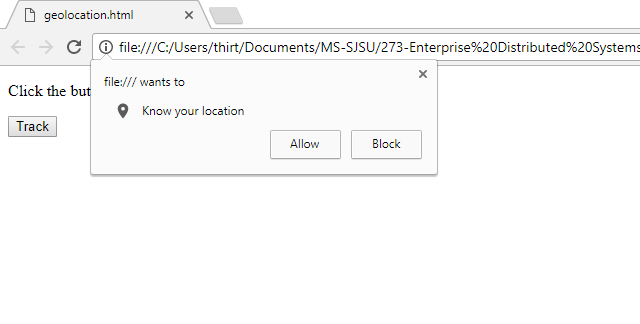
}

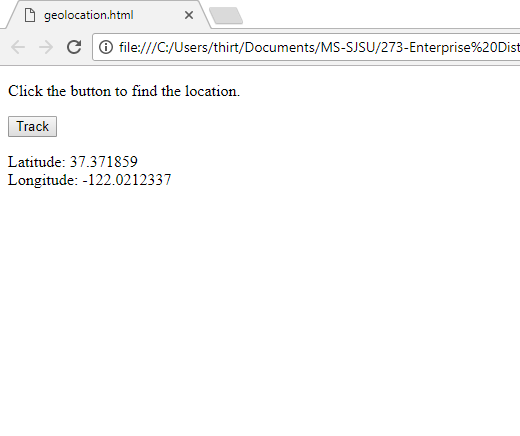
</script>

</body>

</html>

Only after the user gives permission, it accesses the location.





**Java**

Queues: are abstract data types that follow the logic of FIFO (first in first out). This is implemented in various ways in different programming languages. In java, we use head and tail to point to the first and last elements in the queue respectively and perform functions like adding a node and deleting a node.

Write a program to implement queue in linked list and write functions to insert and delete the elements from queue.

**package** javaex02;

**class** Node{

**int** item;

Node next;

Node prev;

}

**public** **class** ex04 **extends** Node{

Node head; //head node defining node that is pointed towards the first element in queue

Node tail; // tail node defining node that is pointed towards the last element in queue

**public** **void** queue()

{

Node a = **new** Node(); //creating queue

Node b = **new** Node();

Node c = **new** Node();

c.item = 1;

c.next = **null**;

c.prev = b;

// Node b = new Node();

b.item = 2;

b.next = c;

b.prev = a;

a.item = 3;

a.next = b;

a.prev = **null**;

head = a;

tail = c;

}

**public** **int** insert(**int** n) //inserting a new node

{

Node d = **new** Node();

d.item= n;

d.next = **null**;

d.prev = tail;

tail = d;

**return** tail.item;

//return top.next.item;

}

**public** **int** delete()

{

**int** i = head.item;

head = head.next;

head.prev = **null**;

**return** i;

//return top.item;

}

**public** **static** **void** main(String[] args) {

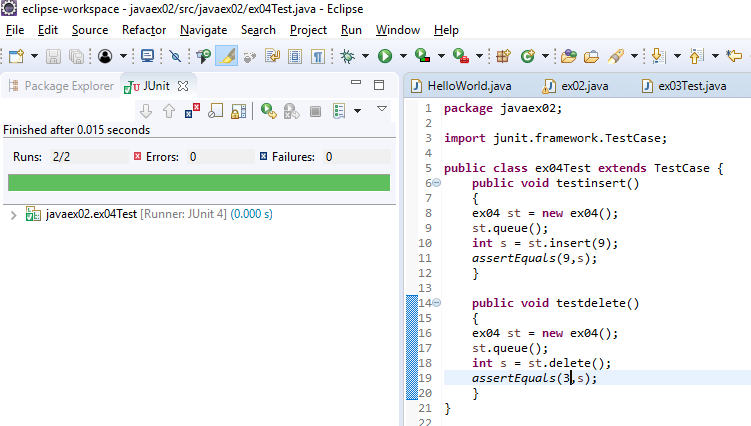
// **TODO** Auto-generated method stub

ex04 st1 = **new** ex04();

st1.queue();

}

}



**package** javaex02;

**import** junit.framework.TestCase;

**public** **class** ex04Test **extends** TestCase {

**public** **void** testinsert()

{

ex04 st = **new** ex04();

st.queue();

**int** s = st.insert(9);

*assertEquals*(9,s);

}

**public** **void** testdelete()

{

ex04 st = **new** ex04();

st.queue();

**int** s = st.delete();

*assertEquals*(3,s);

}

}

Stacks: are abstract data types that follow logic of LIFO(Last in First out). This is implemented in various ways in different programming languages. In java, we use top node to point to the first element in the stack and perform functions like push and pop.

Question: write a program to implement stack using linked lists. Write functions for push and pop

**class** Node {

**int** item;

Node next;

} //defining stack

/\* class stack

{

Node top;

}

\*/

**public** **class** ex03 **extends** Node{

Node top = **null**; //top node defining node that is pointed towards the first element in stack

**public** **void** stack()

{

Node a = **new** Node(); //creating stack

a.item = 1;

a.next = **null**;

Node b = **new** Node();

b.item = 2;

b.next = a;

Node c = **new** Node();

c.item = 3;

c.next = b;

top = c;

}

**public** **int** push(**int** n) //inserting a new node

{

Node d = **new** Node();

d.item= n;

d.next = top;

top = d;

**return** top.item;

//return top.next.item;

}

**public** **int** pop()

{

**int** i = top.item;

top = top.next;

**return** i;

//return top.item;

}

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

/\*Node a = new Node(); //creating stack

a.item = 1;

a.next = null;

Node b = new Node();

b.item = 2;

b.next = a;

Node c = new Node();

c.item = 3;

c.next = b; \*/

ex03 st1 = **new** ex03();

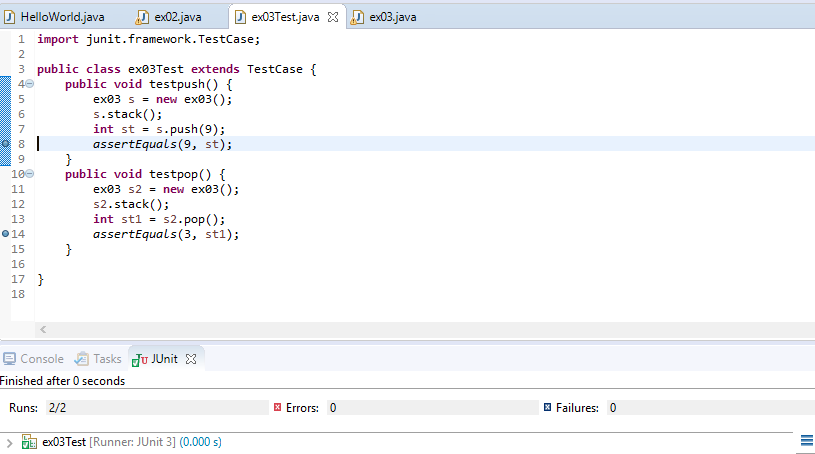
//st1.stack();

//st1.push(9);

//System.out.println(st1.pop());

}

}



**import** junit.framework.TestCase;

**public** **class** ex03Test **extends** TestCase {

**public** **void** testpush() {

ex03 s = **new** ex03();

s.stack();

**int** st = s.push(9);

*assertEquals*(9, st);

}

**public** **void** testpop() {

ex03 s2 = **new** ex03();

s2.stack();

**int** st1 = s2.pop();

*assertEquals*(3, st1);

}

}

Arrays: are a data structure that is declared a specific size carries sequential data of preferably same data type.

Question: Write a program to sort the values of the array and return the median of the array.

import java.util.Arrays;

import java.util.Collections;

public class ex02

{

public static int sort(){

int[] arr = {20, 23, 75, 89, 57, 98, 0};

int x = arr.length;

for (int i = 0; i<arr.length; i++)

{

for (int j = 0; j<arr.length; j++)

{

if (arr[i] < arr[j])

{

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

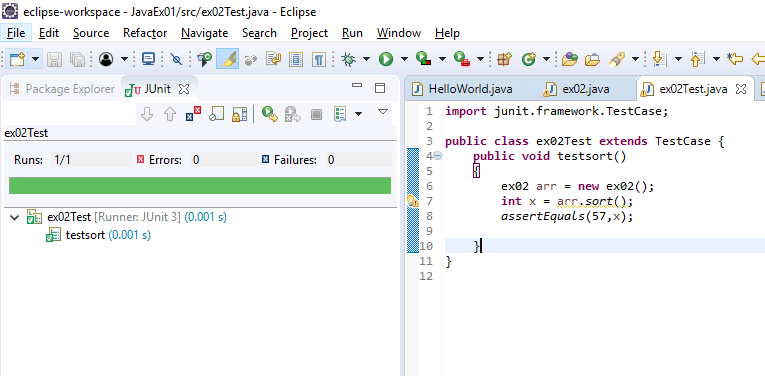
}

}

return arr[x/2]; // void methods cannot return value

// System.out.println("The function will return the mid element stored in the array" + " " + arr[x/2]);

}

}

**import** junit.framework.TestCase;

**public** **class** ex02Test **extends** TestCase {

**public** **void** testsort()

{

ex02 arr = **new** ex02();

**int** x = arr.*sort*();

*assertEquals*(57,x);

}

}

Interfaces: In Java, to reduce the complexity of creating more than one sub classes for one class, interface is another way to perform the multiple inheritance. Class implements the interface and interface extends an interface.

Question: Write a program to demonstrate interface and class to calculate the area of different shapes.

**import** java.util.\*;

**import** java.lang.\*;

**import** java.io.\*;

**interface** areaC {

**public** String ***base*** = "circle";

**public** **void** area(**int** r);

}

**interface** areaR **extends** areaC {

**public** String ***base*** = "rectangle";

**public** **void** area(**int** l, **int** b);

}

**class** area1 **implements** areaR{

// public String base = "shape3";

**public** **void** area(**int** r) {

**double** a = 3.14\*r\*r;

System.***out***.println("Drawing Circle here:" + ***base*** + ": " + a); //base gets overridden.

}

**public** **void** area(**int** l, **int** b) {

**double** a= l\*b;

System.***out***.println("Drawing Rectangle here:" + ***base*** + ": " + a);

}

}

**public** **class** ex01 {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

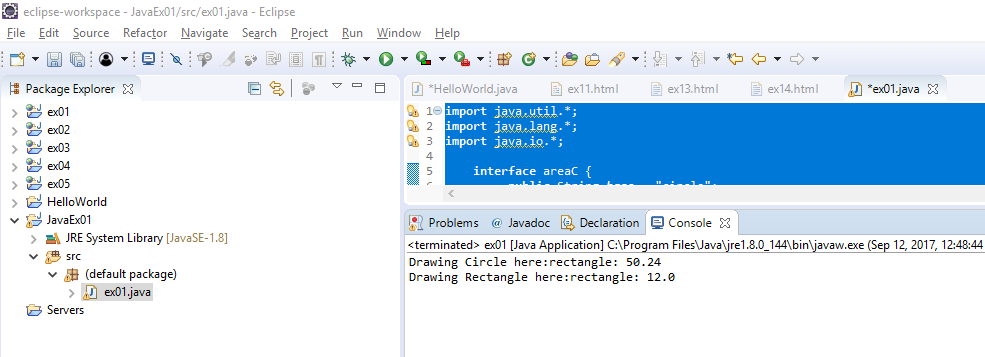
area1 ar = **new** area1();

ar.area(4);

ar.area(3,4); //the method is distinguished by different parameters.

}

}



**import** junit.framework.TestCase;

**public** **class** ex01Test **extends** TestCase {

**public** **void** testarea(){

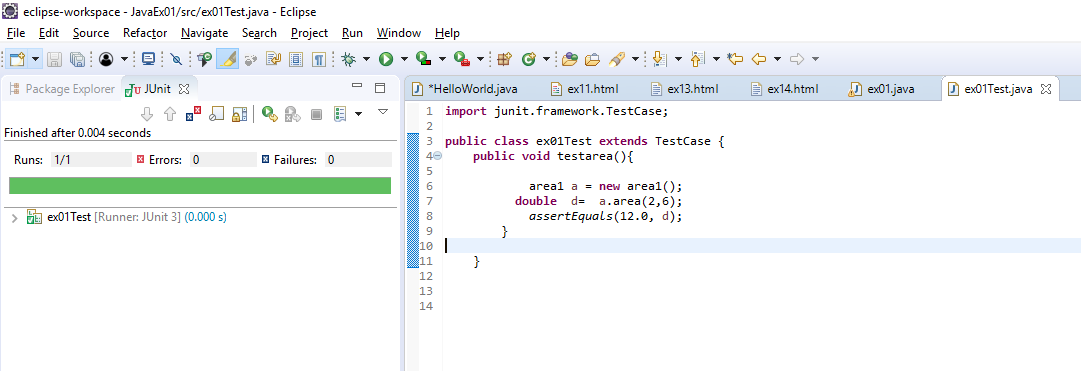
area1 a = **new** area1();

**double** d= a.area(2,6);

*assertEquals*(12.0, d);

}

}



Collections: The Java Collections Framework contains standard interfaces, classes and algorithms which helps in storing and manipulating the data. In addition, it also has map interfaces which stores key/value pair that integrate with collections.

Write a program to implement one of the collections i.e. ArrayList. Write functions to delete and add elements to the arraylist.

**package** javaex02;

**import** java.util.\*;

**public** **class** ex05 {

ArrayList arrL = **new** ArrayList();

// return al.size();

**public** **int** addArrLis() // display arraylist

{

// add elements to the array list

arrL.add("1");

arrL.add("2");

arrL.add("3");

arrL.add("4");

arrL.add("5");

arrL.add("6");

arrL.add(2, "A2");

**return** arrL.size();

}

**public** **int** removeArrLis() // Remove elements from the array list

{

arrL.remove("6");

arrL.remove(2);

**return** arrL.size();

}

**public** **int** ArrLisnum()

{

**return** arrL.size();

}

**public** **static** **void** main(String args[]) {

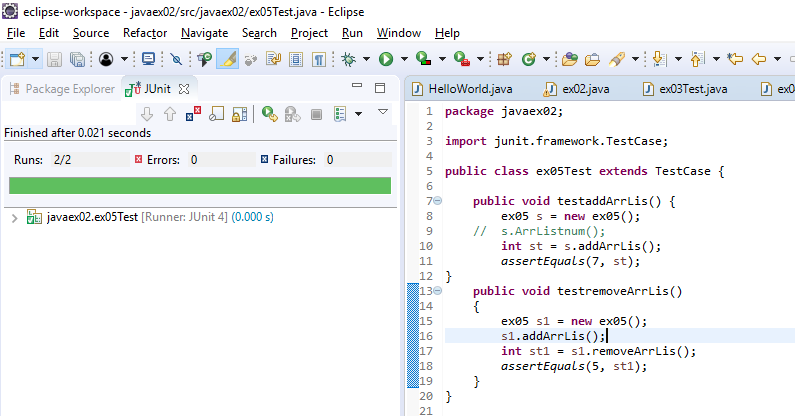
ex05 A = **new** ex05();

A.addArrLis();

A.removeArrLis();

}

}



**package** javaex02;

**import** junit.framework.TestCase;

**public** **class** ex05Test **extends** TestCase {

**public** **void** testaddArrLis() {

ex05 s = **new** ex05();

// s.ArrListnum();

**int** st = s.addArrLis();

*assertEquals*(7, st);

}

**public** **void** testremoveArrLis()

{

ex05 s1 = **new** ex05();

s1.addArrLis();

**int** st1 = s1.removeArrLis();

*assertEquals*(5, st1);

}

}

Generics: Generic methods let you define a set of related methods in a single method declaration or a set of related data types in a single class. This provides the programmer with compile time safety by catching invalid types at compile time.

Write a generic program to find the minimum of input. Take inputs as integer, double and string and test.

**package** javaex02;

**public** **class** ex06 {

// determines the largest of three Comparable objects

**public** **static** <M **extends** Comparable<M>> M minimum(M a, M b, M c) {

M min = a; //

**if**(b.compareTo(min) < 0) {

min = b; // if b is the largest

}

**if**(c.compareTo(min) < 0) {

min = c; // if c is the largest

}

**return** min; // returns the largest object

}

**public** **static** **void** main(String args[]) {

ex06 s = **new** ex06();

s.*minimum*(2, 3, 4);

s.*minimum*(3.3,4.4, 0.9);

s.*minimum*("apple", "banana", "cat");

/\* System.out.printf( minimum( 3, 4, 5 ));

System.out.printf( minimum( 6.6, 8.8, 7.7 ));

System.out.printf(minimum("pear", "apple", "orange")); \*/

}

}

**package** javaex02;

**import** junit.framework.TestCase;

**public** **class** ex06Test **extends** TestCase {

**public** **void** testmaximum()

{

ex06 s = **new** ex06();

**int** i = s.*minimum*(20, 30, 40);

*assertEquals*(i,20);

}

**public** **void** testmaximum1()

{

ex06 s = **new** ex06();

**double** i = s.*minimum*(2.3, 3.6, 4.9);

*assertEquals*(i,2.3);

}

**public** **void** testmaximum2()

{

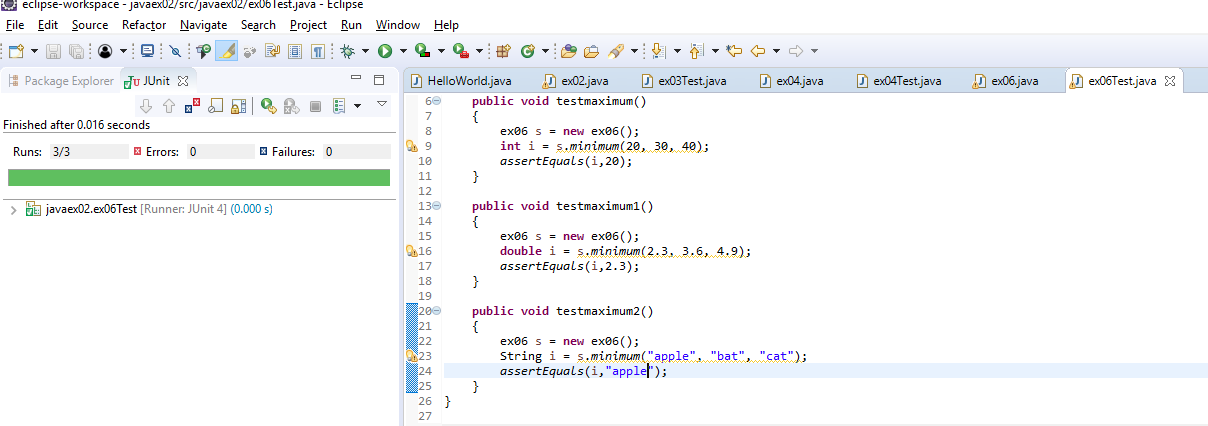
ex06 s = **new** ex06();

String i = s.*minimum*("apple", "bat", "cat");

*assertEquals*(i,"apple");

}

}



Multithreading programming is creating multiple processes for multiple tasks in the same program that can run parallelly on the processor, making optimal use of the resources i.e. CPU.

Write a program to show multithreading functionality for two threads “thread1” and “thread2” and check if the thread is alive.

**package** javaex02;

**class** ex07 **implements** Runnable {

**public** Thread thread;

**public** String tName;

ex07( String name) {

tName = name;

System.***out***.println("Creating " + tName );

}

**public** **void** run() {

System.***out***.println("Running " + tName );

**try** {

**for**(**int** i = 4; i > 0; i--)

{

System.***out***.println("Thread: " + tName + ", " + i);

// Let the thread sleep for a while.

Thread.*sleep*(50);

}

}

**catch** (InterruptedException e)

{

System.***out***.println("Thread " + tName + " interrupted." + e);

}

System.***out***.println("Thread " + tName + " exiting.");

}

**public** **void** start () {

System.***out***.println("Starting " + tName );

**if** (thread == **null**) {

thread = **new** Thread (**this**, tName);

thread.start ();

}

}

**public** **final** **boolean** TisAlive() //checking if the thread is alive.

{

// tName = name;

**boolean** y = thread.isAlive();

**return** y;

}

**public** **static** **void** main(String args[]) {

ex07 R1 = **new** ex07( "Thread1");

R1.start();

R1.run();

R1.TisAlive();

ex07 R2 = **new** ex07( "Thread2");

R2.start();

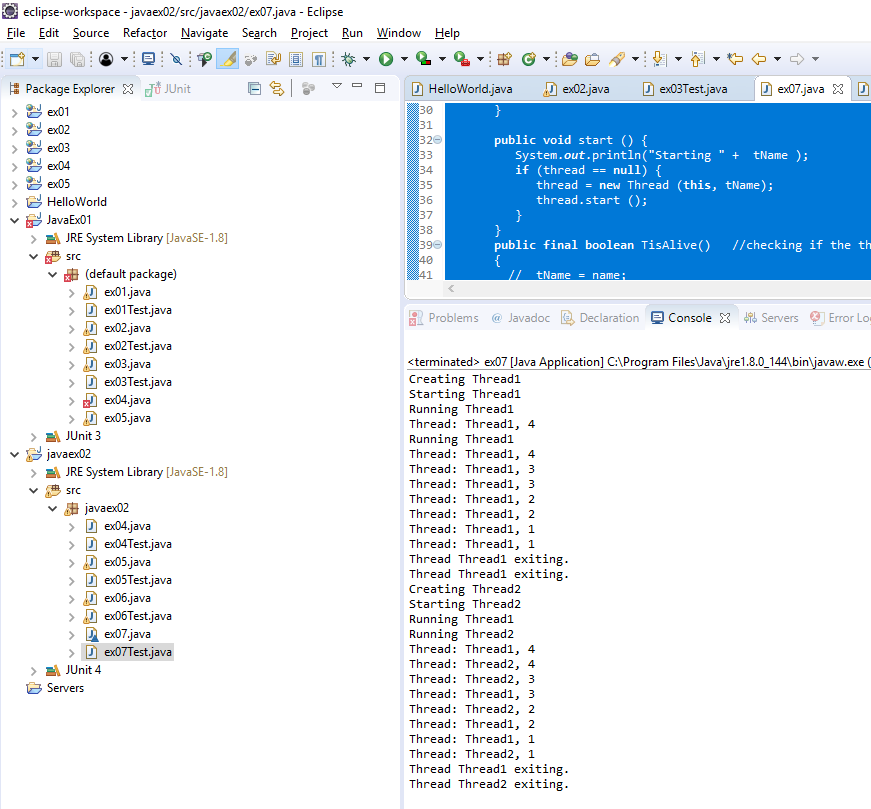
R1.run();

R2.TisAlive();

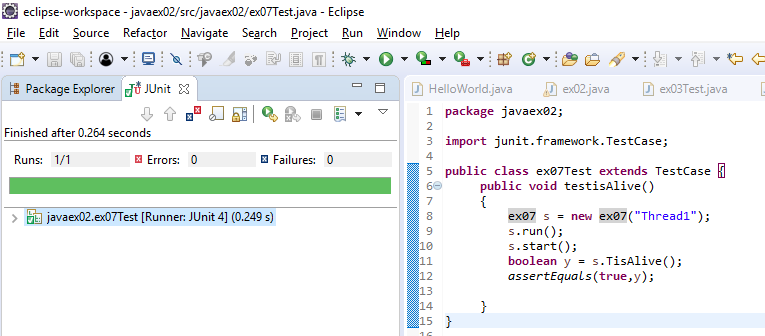
}

}

Output:



Junit testing:



**package** javaex02;

**import** junit.framework.TestCase;

**public** **class** ex07Test **extends** TestCase {

**public** **void** testisAlive()

{

ex07 s = **new** ex07("Thread1");

s.run();

s.start();

**boolean** y = s.TisAlive();

*assertEquals*(**true**,y);

}

}