Akshay Mishra

011476673

**Javascript**

**Variables**

**Introduction:** Variables are placeholders for temporary data or values. This data can be used by referencing it from the other parts of the program using variable names. In Javascript, one doesn’t need to define the data type such as String, Integer as Javascript is a dynamic language.

**Question:** Calculate um of the elements

**Program: Variable.html**

<!DOCTYPE html>

<html>

<head>

<script>

function addItems(){

var one=document.getElementById("one").value;

var two=document.getElementById("two").value;

var sum= +one + +two;

document.getElementById("total").innerHTML=sum;

}

</script>

<meta charset="UTF-8">

<title>Variable</title>

</head>

<body>

Number #1 : <input type="Number" id="one"/></br>

Number #2 : <input type="Number" id="two"/></br></br>

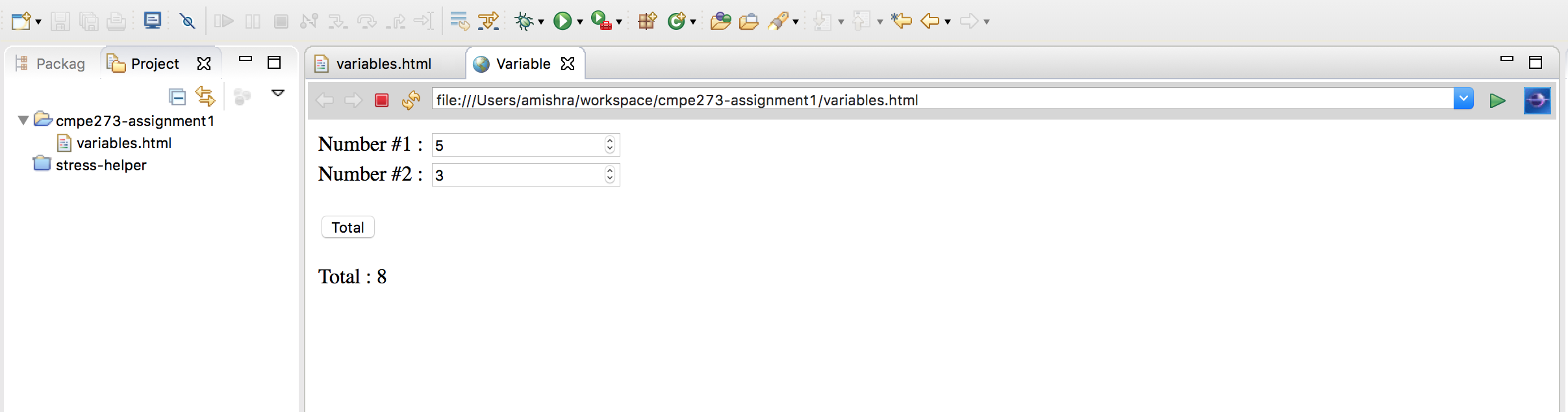
<input type="Button" onclick="addItems()" value="Total"/></br></br>

Total    : <label id="total"></label>

</body>

</html>

**Output:**

****

**Objects**

**Introduction:** Objects in JavaScript are standalone entities with properties. A property is a simple key value pair. If value in this pair can also be a function, then the property is called as a method of the object. Objects are inherited from objects which are referred to as ‘Prototype’.

**Question: Compose a box from the properties supplied**

**Code:**

**Objects.html**

<!DOCTYPE html>

<html>

<head>

<script>

function createBox(){

var box = new Object();

box.height = document.getElementById("Height").value;

box.length = document.getElementById("Length").value;

box.breadth = document.getElementById("Breadth").value;

box.color = document.getElementById("Color").value;

document.getElementById("Box").innerHTML='height-' + box.height + ', breadth-' + box.breadth + ', length-' + box.length + ', color-' + box.color;

//$('.Box').html(color);

}

</script>

<meta charset="UTF-8">

<title>Objects</title>

</head>

<body>

Height   : <input type="Number" id="Height" value=3 /></br>

Length   : <input type="Number" id="Length" value=4 /></br>

Breadth  : <input type="Number" id="Breadth"/ value=5 ></br>

Color    : <input type="Text" id="Color" value="Red" /></br></br>

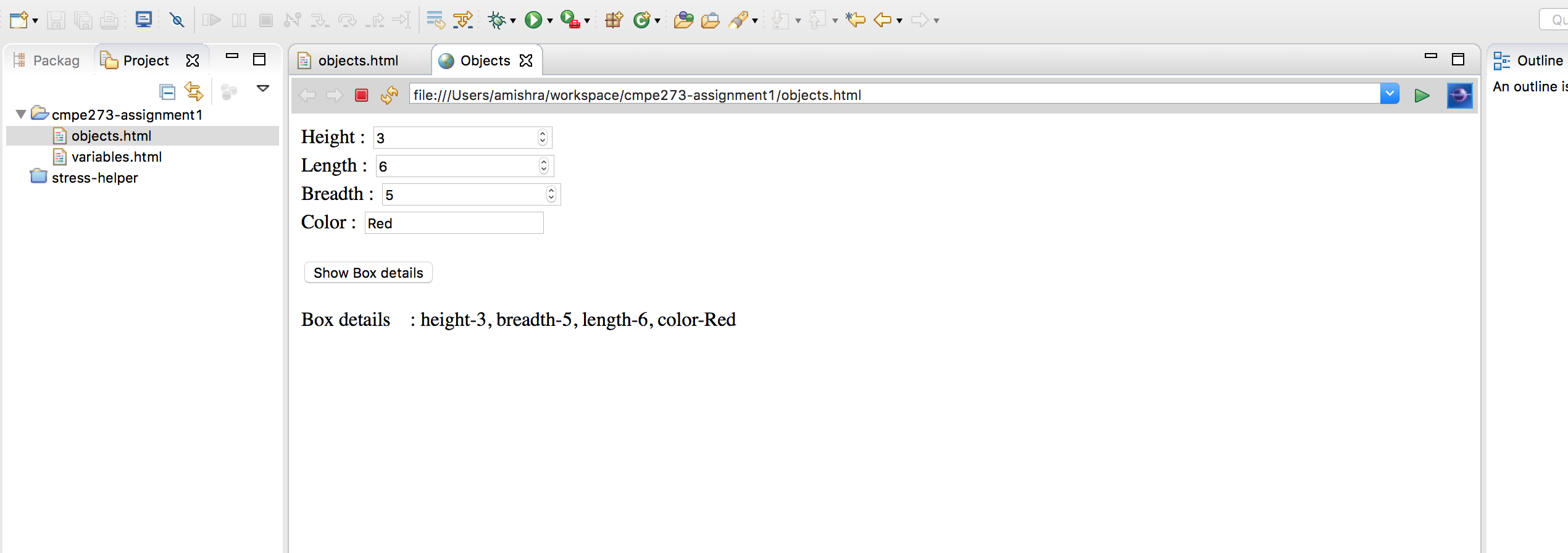
<input type="Button" onclick="createBox()" value="Show Box details"/></br></br>

Box details    : <label id="Box" class="Box" ></label>

</body>

</html>

**Output:**

****

**Functions:**

**Introduction:** Functions in JavaScript are similar to the functions in other languages. It groups set of actions together in order to be called from other parts of the program. In JavaScript, functions can be called by passing them parameters, and can return value just like other programming languages.

**Question: Write a function that randomly changes color of a shape**

**Code:**

<!DOCTYPE html>

<html>

<head>

<script>

function changeColor(){

var colorArray = ["red", "blue", "green", "yellow", "orange", "black"];

    var color = colorArray[(Math.random()\*colorArray.length)|0];

document.getElementById("circle").style.fill=color;

}

</script>

</head>

<body>

<svg height="100" width="100">

  <circle id="circle" cx="50" cy="50" r="40" stroke="black" stroke-width="3" fill="red" />

</svg>

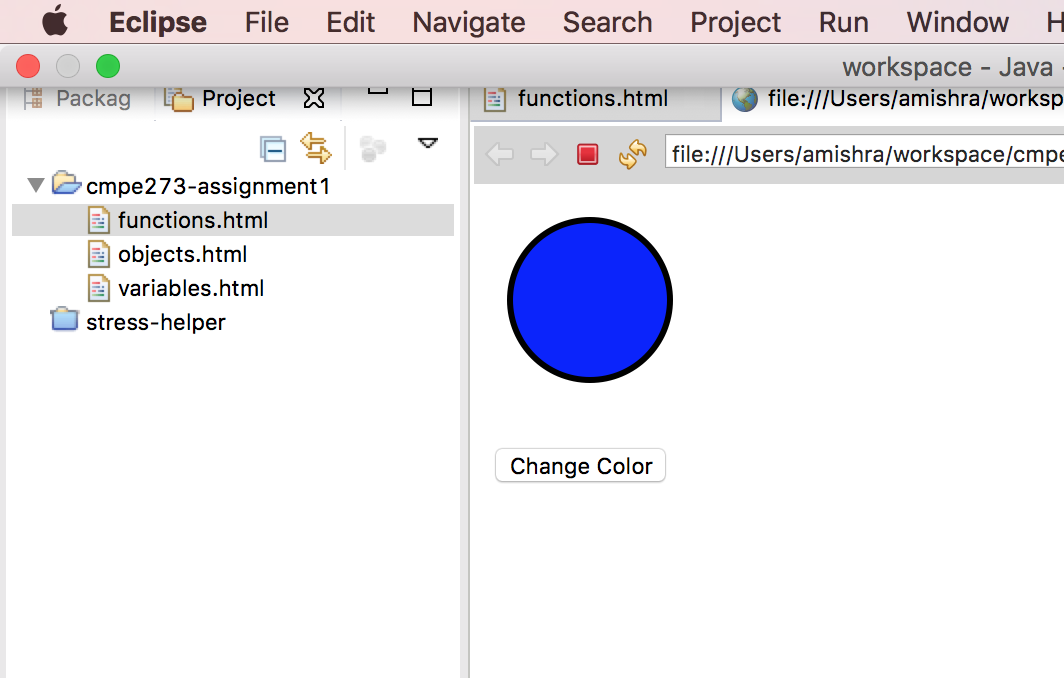
</br></br>

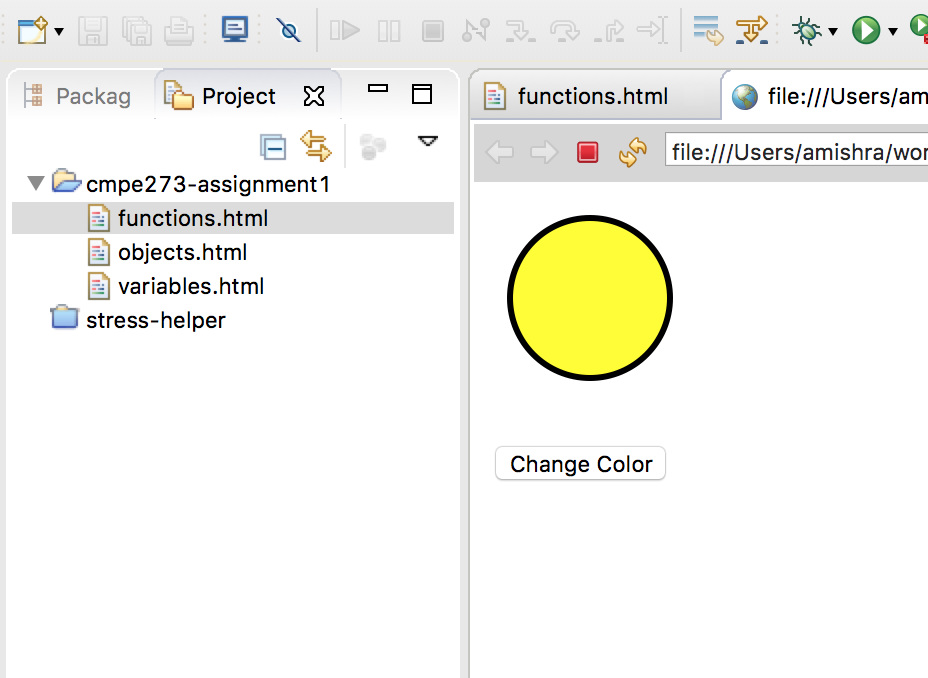
 <input type="Button" onClick=changeColor() value="Change Color">

</body>

</html>

**output:**





**Events:**

**Introduction:** Events in JavaScript are used to manage interaction with HTML effectively. When the DOM loads, it’s an event. When a user clicks somewhere, that can be counted as an event. Once an event is detected, then it can be handled or responded by the page.

**Question:**

Write a function to detect change the color of a shape based on position of the mouse relative to its area

**Code:**

<!DOCTYPE html>

<html>

<head>

<script>

function on(elem){

elem.style.fill="green";

console.log("here");

}

function out(){

document.getElementById("circle").style.fill="red";

}

</script>

</head>

<body>

<svg height="100" width="100">

  <circle id="circle" cx="50" cy="50" r="40" stroke="black" stroke-width="3" fill="blue" onmouseover="on(this);" onmouseout="out(this);"/>

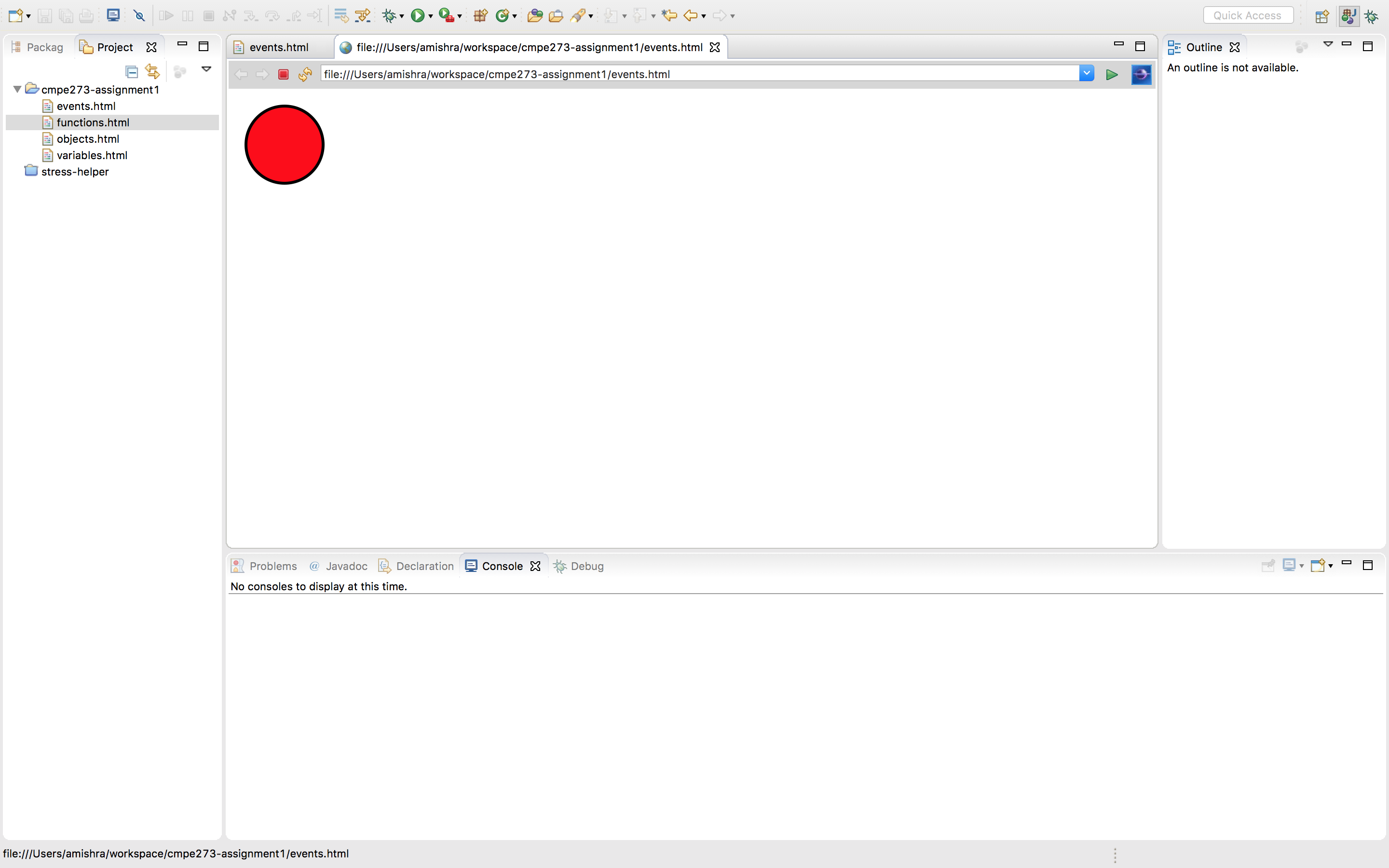
</svg>

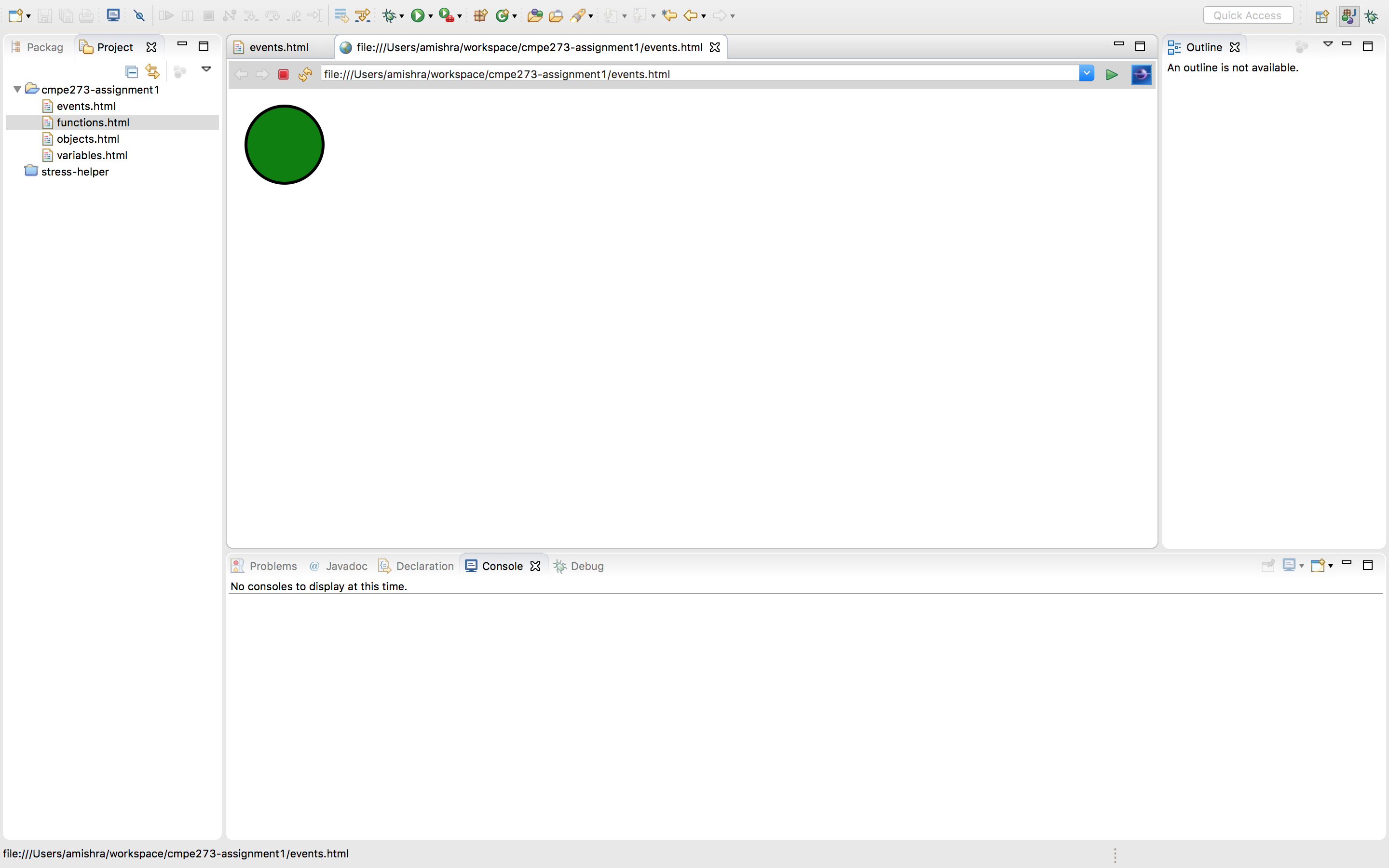
</br></br>

</body>

</html>

**Output:**





**Arrays:**

**Introduction:** Just like other programming languages, arrays in Javascript are used to store multiple values of same data type in a single variable. Each value can be accessed using an index associated with every element.

**Question:** Write a program to add elements to an array

**Code:**

<!DOCTYPE html>

<html>

<head>

<script>

var fruitArray=[];

function addToArray(elem){

fruitArray.push(elem.value);

}

function printFruitList(){

document.getElementById("list").innerHTML=fruitArray.toString();

}

</script>

<meta charset="UTF-8">

<title>Arrays</title>

</head>

<body>

Add To Cart</br></br>

<input type="Button" name="one" value="Mango" onClick="addToArray(this);">

<input type="Button" name="two" value="Banana" onClick="addToArray(this);">

<input type="Button" name="three" value="Orange" onClick="addToArray(this);">

<input type="Button" name="four" value="Guava" onClick="addToArray(this);"></br>

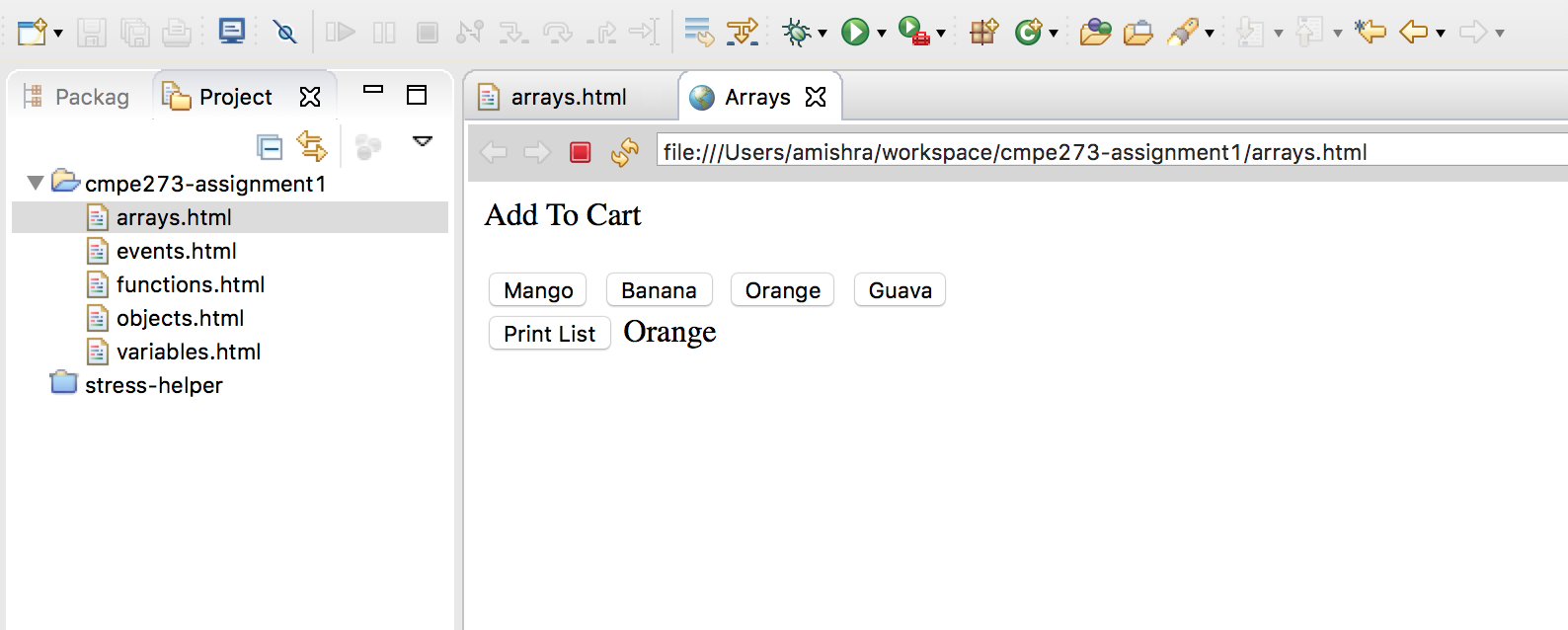
<input type="Button" onClick="printFruitList();" value="Print List">

<label id="list"></label>

</body>

</html>

**Output:**

****

**Inheritance**

**Introduction:** Inheritance, just like in other many programming languages, in Javascript allows us to reuse the code and functionality. It makes available the properties or methods to the subclass by inheriting parent class. In Javascript, functionality is inherited through prototypical inheritance rather than the usual inheritance. What it means that, eg. Methods to a sub class are linked through a prototype object than directly to the subclass.

**Question:** Make dogs bark in Javascript using prototypical inheritance.

**Code:**

<!DOCTYPE html>

<html>

<head>

<script>

function Cat(){

}

var katie = new Cat();

Cat.prototype.scream= function() {

alert("meow!!");

}

katie.scream();

</script>

<meta charset="UTF-8">

<title>Inheritance</title>

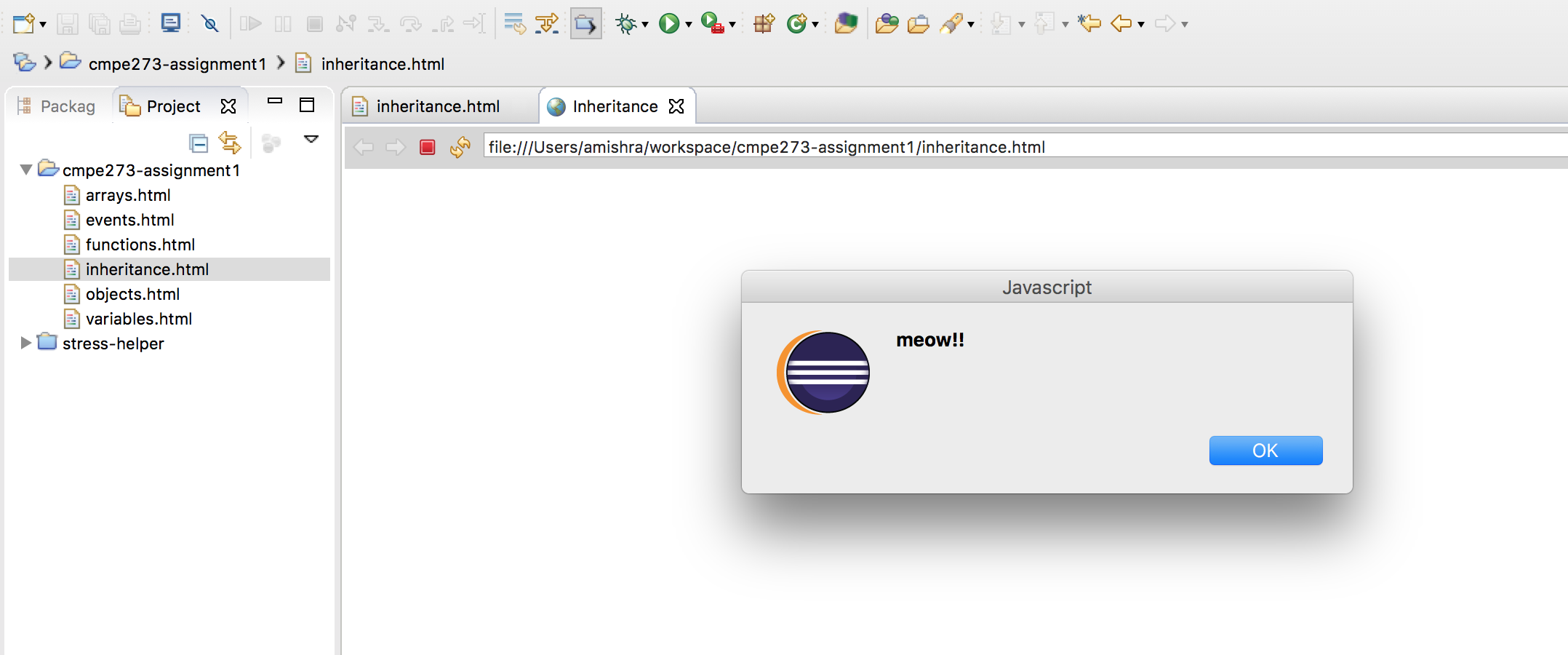
</head>

<body>

</body>

</html>

**output:**

****

**Conditions**

**Introduction:** Conditions in Java script behave the similar way to the conditional statements in other programming languages. Conditional statements help us to modify or manipulate the flow of the program based on inputs. A lot of options like while statements, if-else statements, case statements are available for the use in Javascript.

**Program:** Decide the color of the circle based on hour of the day

**Code:**

<!DOCTYPE html>

<html>

<head>

<script>

function getStatus(){

var today= new Date();

var hours= today.getHours();

if(hours>6 && hours<=18){

document.getElementById("circle").style.fill="white";

}

else {

document.getElementById("circle").style.fill="black";

}

}

</script>

</head>

<body>

<svg height="100" width="100">

  <circle id="circle" cx="50" cy="50" r="40" stroke="black" stroke-width="3" fill="blue"/>

</svg>

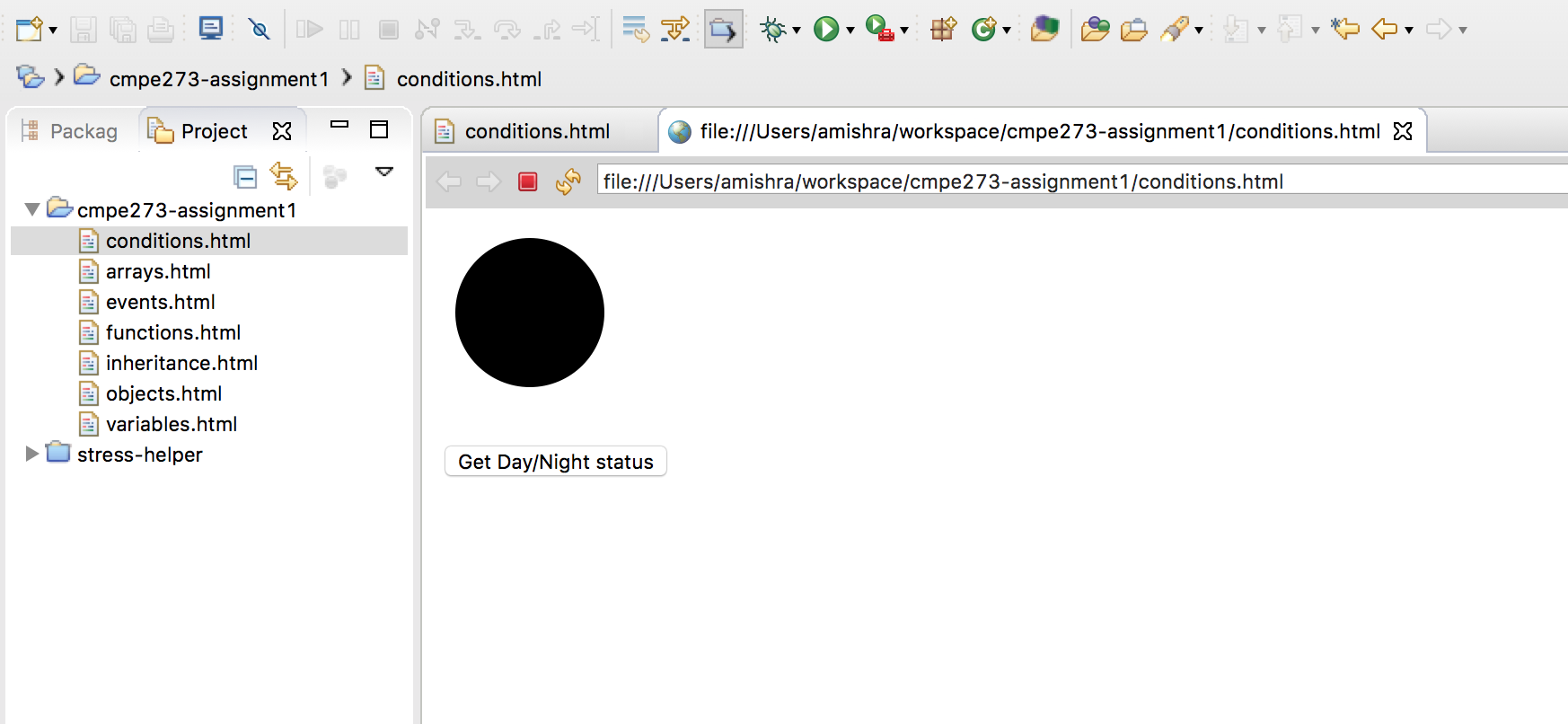
</br></br>

 <input type="Button" onClick=getStatus() value="Get Day/Night status">

</body>

</html>

**Output:**

****

**Regular expressions:**

**Introduction:** Regular expressions are used to find patterns in a string. It consists of wildcards characters following a standard syntax and it tries to match it against a given string.

**Question:** Find whether a phone number is valid or not

**Code:**

<!DOCTYPE html>

<html>

 <head>

  <script type="text/javascript">

   function handleIt(phone) {

  if (/^\(\d\d\d\) \d\d\d-\d\d\d\d$/g.exec(phone)) {

  document.getElementById("status").innerHTML="valid"

  }

  else {

  document.getElementById("status").innerHTML="invalid"

  }

   }

  </script>

 </head>

 <body>

  <form name="myform" action="javascript:handleIt(phone.value)">

   <input type="text" name="phone" id="phone" value="(012) 345-6789">

   <input name="Submit" type="submit" value="Check"/>

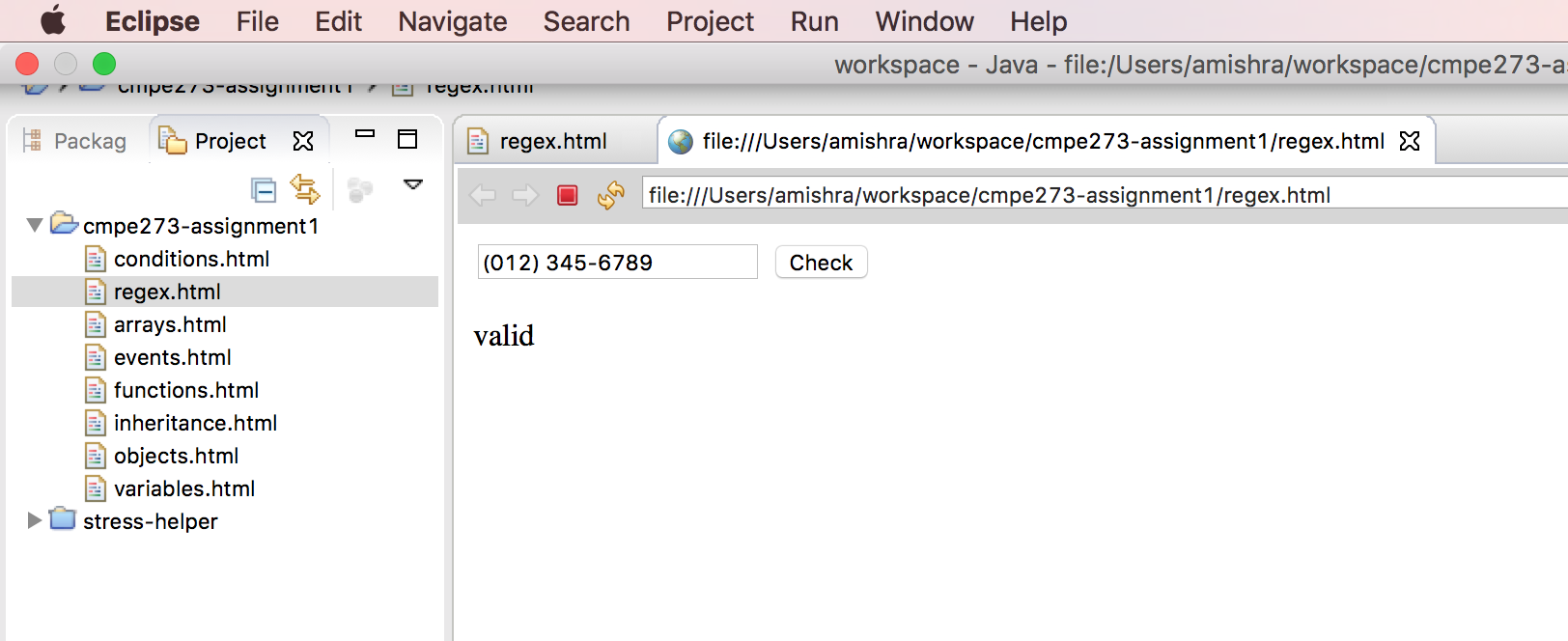
  </form></br>

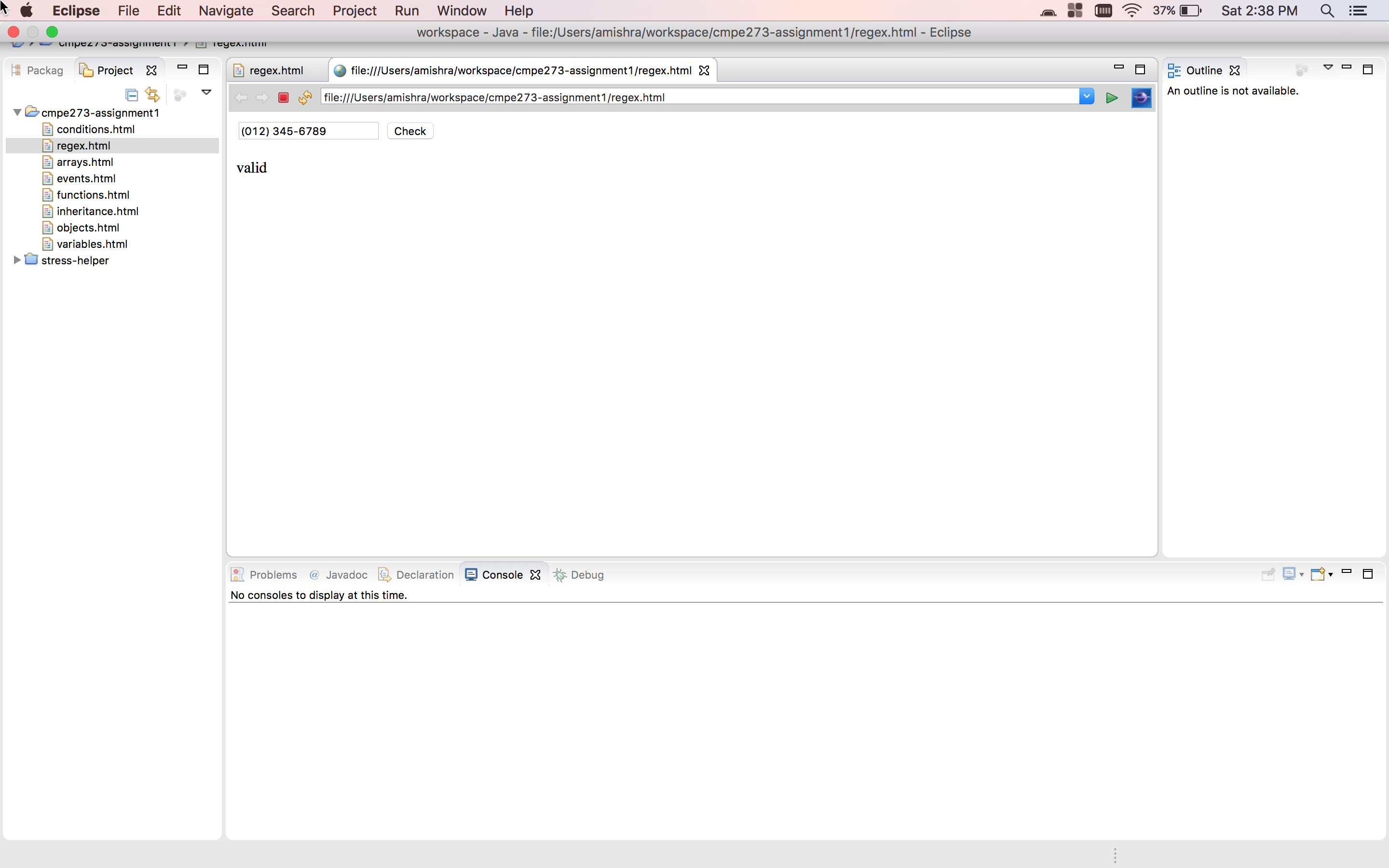
  <label id="status">Enter a phone #</label>

 </body>

</html>

**output:**

****

****

**use strict**

**Introduction:** The ‘use strict’ is a directive. It is not a statement but a literal expression that is ignored by the compiler and is only used to convey to the compiler that the code should be executed in the strict mode.

**Question:** Find whether a script is invalid or not when compared to the strict mode rules

**Code:**

<!DOCTYPE html>

<html>

<head>

<script>

function strict(){

"use strict";

document.getElementById("mode").innerHTML = "Strict Mode on";

try {

pi=3.14;

}

catch(err) {

    document.getElementById("compiler\_message").innerHTML = err.message;

}

}

function nonstrict(){

document.getElementById("mode").innerHTML = "Regular Mode on";

try {

pi=3.14;

}

catch(err) {

    document.getElementById("compiler\_message").innerHTML = err.message;

}

document.getElementById("compiler\_message").innerHTML = "all good!";

}

</script>

<meta charset="UTF-8">

<title>Strict mode</title>

</head>

<body>

<input type="button" id="strict" value="Strict mode" onClick="strict()" >

<input type="button" id="regular" value="Regular mode" onClick="nonstrict()"></br></br>

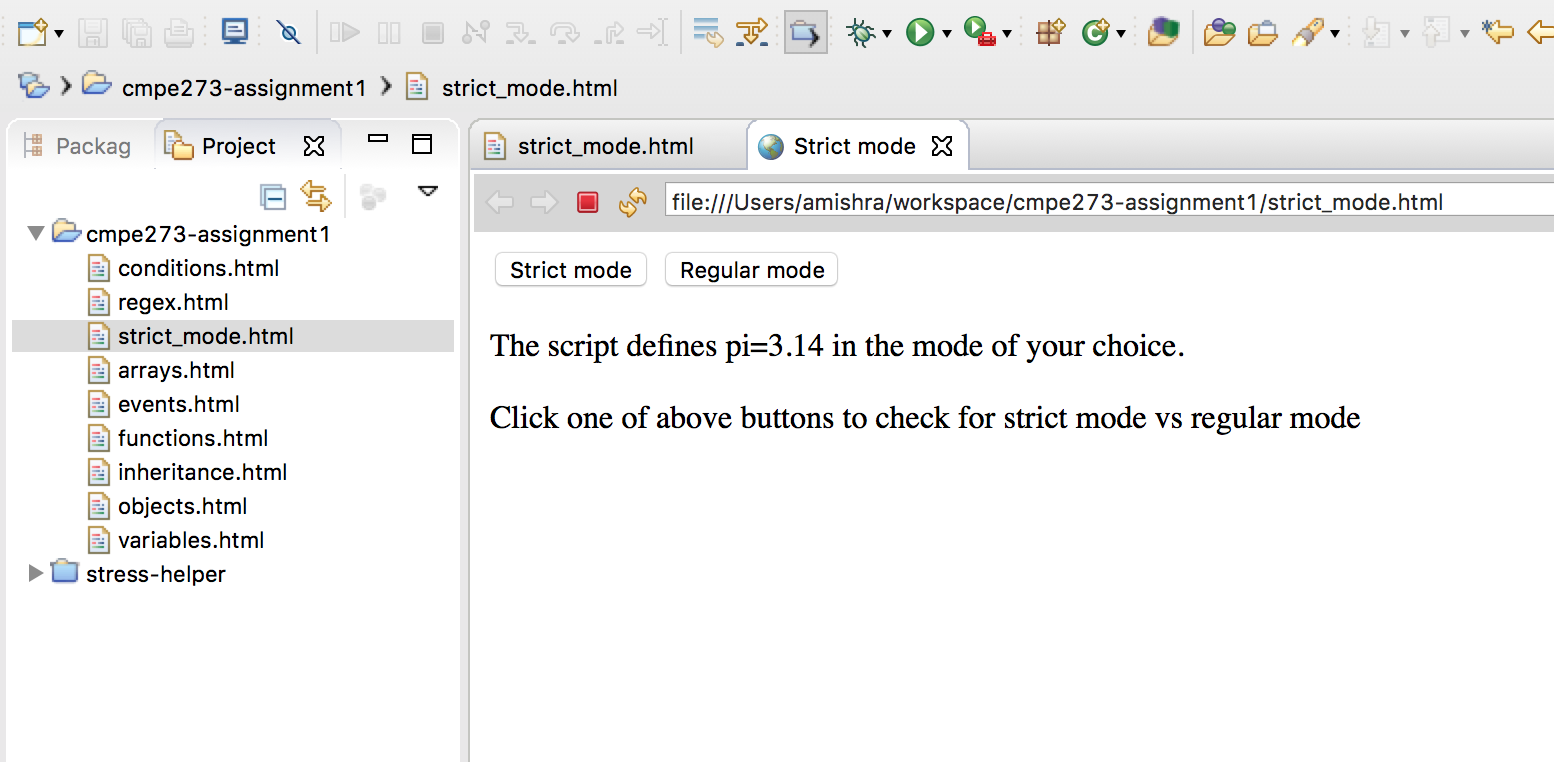
<label id="mode">The script defines pi=3.14 in the mode of your choice.</label></br></br>

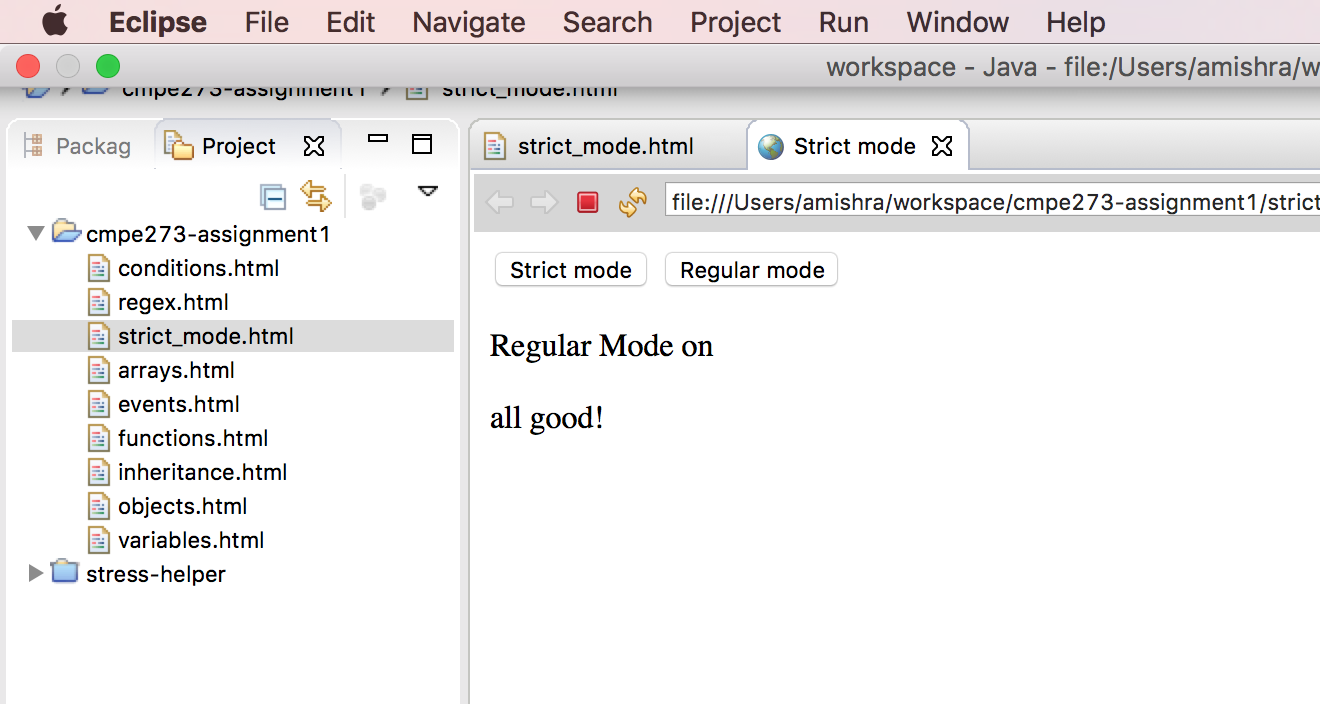
<label id="compiler\_message">Click one of above buttons to check for strict mode vs regular mode</label>

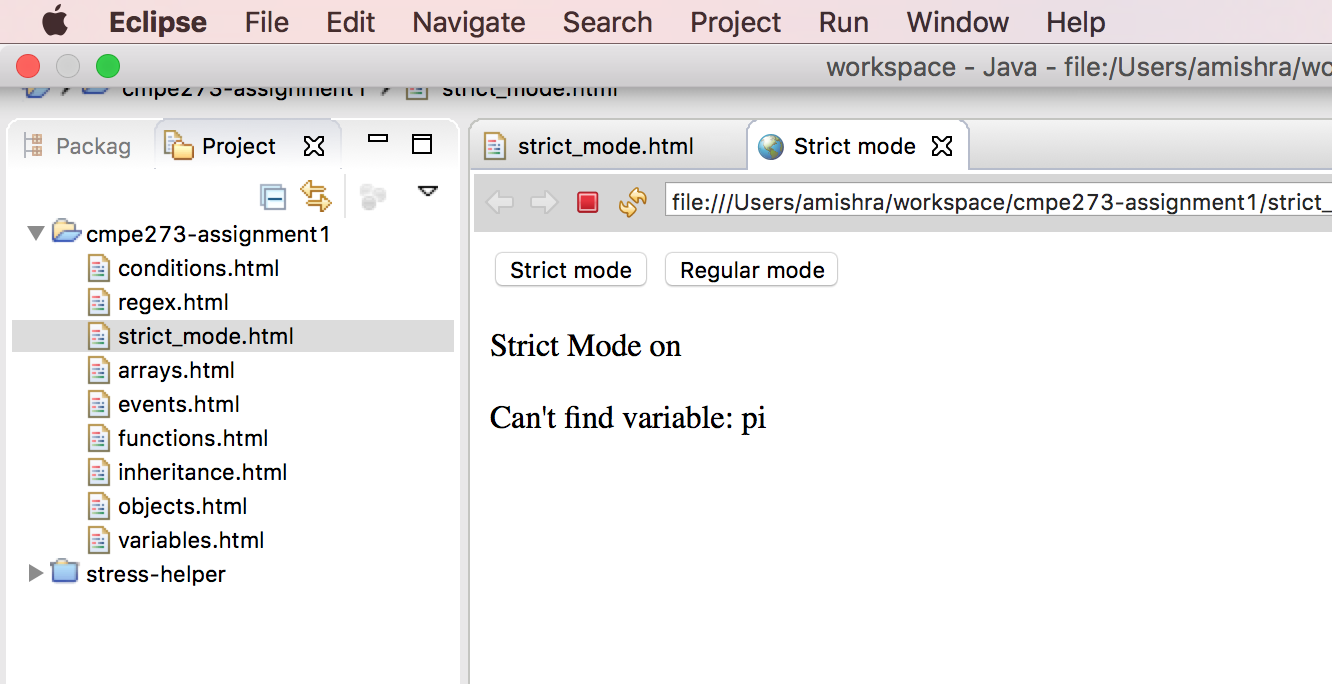
</body>

</html>

**output:**

****

****

****

**Errors:**

**Introduction:** Errors are the instances of objects created and thrown when an error occurs during compilation or during runtime. In Javascript, one can also define custom errors, apart from being able to use already defined errors. There are multiple types of errors. Eg. EvalError etc.

**Question:** Throw an error with a wrong script. Print an error message on the screen.

**Code:**

<!DOCTYPE html>

<html>

<head>

<script>

function strict(){

document.getElementById("mode").innerHTML = "error message";

try {

null.foobar();

}

catch(err) {

    document.getElementById("compiler\_message").innerHTML = err.message;

}

}

</script>

<meta charset="UTF-8">

<title>Strict mode</title>

</head>

<body>

<input type="button" id="strict" value="Strict mode" onClick="strict()" ></br></br>

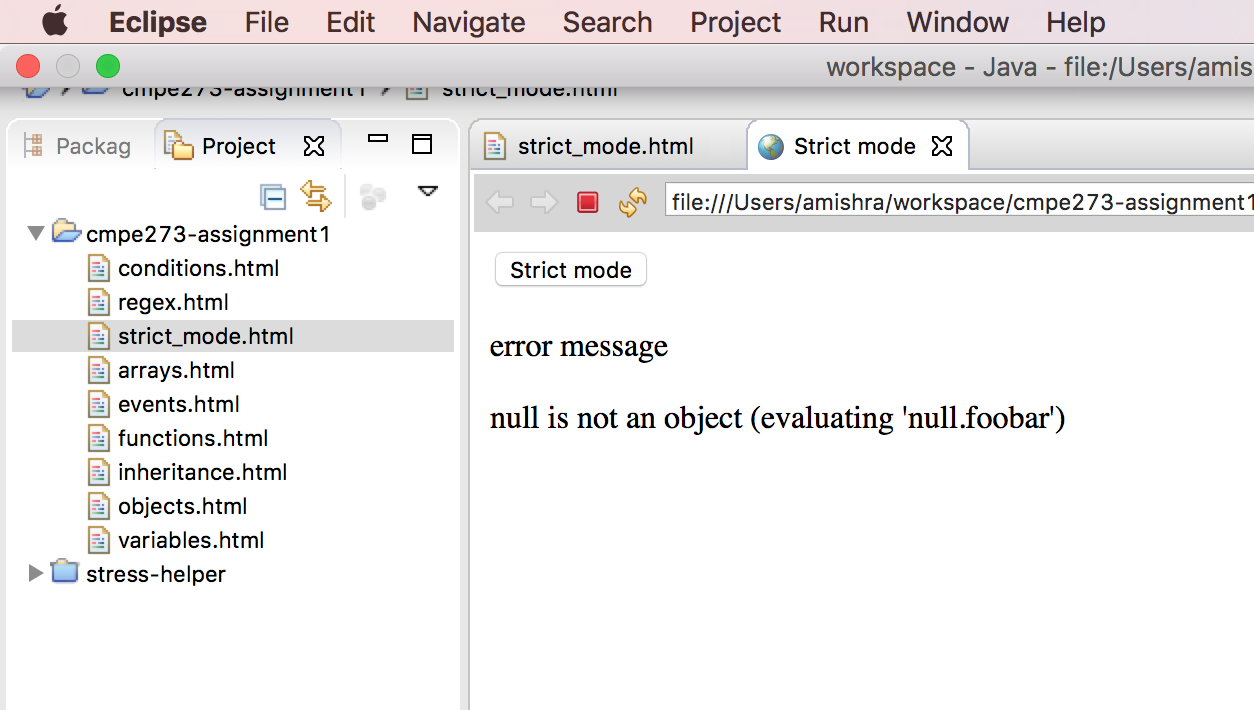
<label id="mode">The script defines a variable on an invalid type.</label></br></br>

<label id="compiler\_message">Click button to runs the script</label>

</body>

</html>

**output:**

****

**Type conversions:**

**Introduction:** Javascript is a loosely-typed language in a sense that datatype doesn’t need to be declared at the time of the declaration of the variable. Also, a Javascript variable can hold a data of more than one type at a time since a variable is not bound to follow a particular type. Javascript can also implicitly convert a variable type based on required by the context, apart from option of explicit type conversion.

**Code:**

<!DOCTYPE html>

<html>

<head>

<script>

function addNumber(){

document.getElementById('resultNum').innerHTML=null+document.getElementById("String").value;

}

</script>

<meta charset="UTF-8">

<title>Variable</title>

</head>

<body>

Input : <input type="String" id="String" value="5"/>

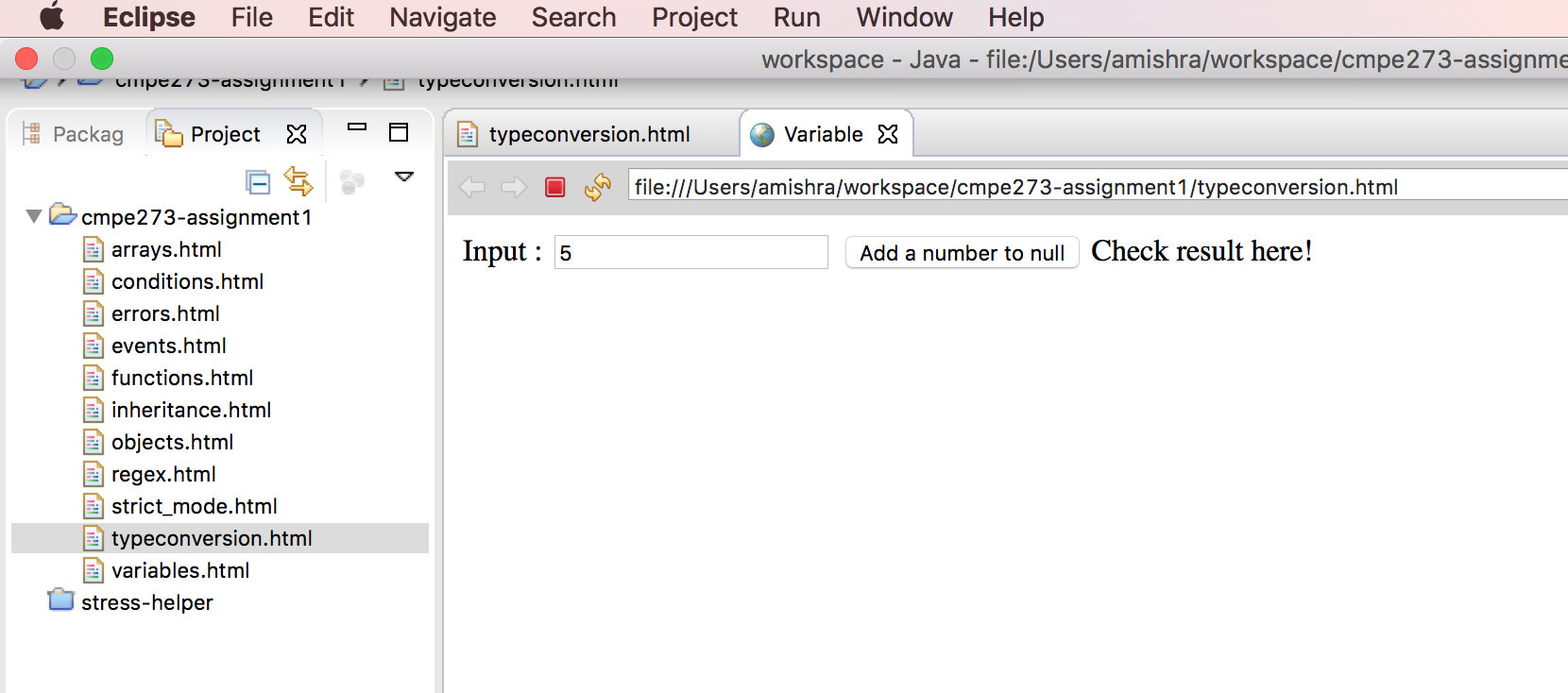
<input type="Button" onclick="addNumber()" value="Add a number to null" />

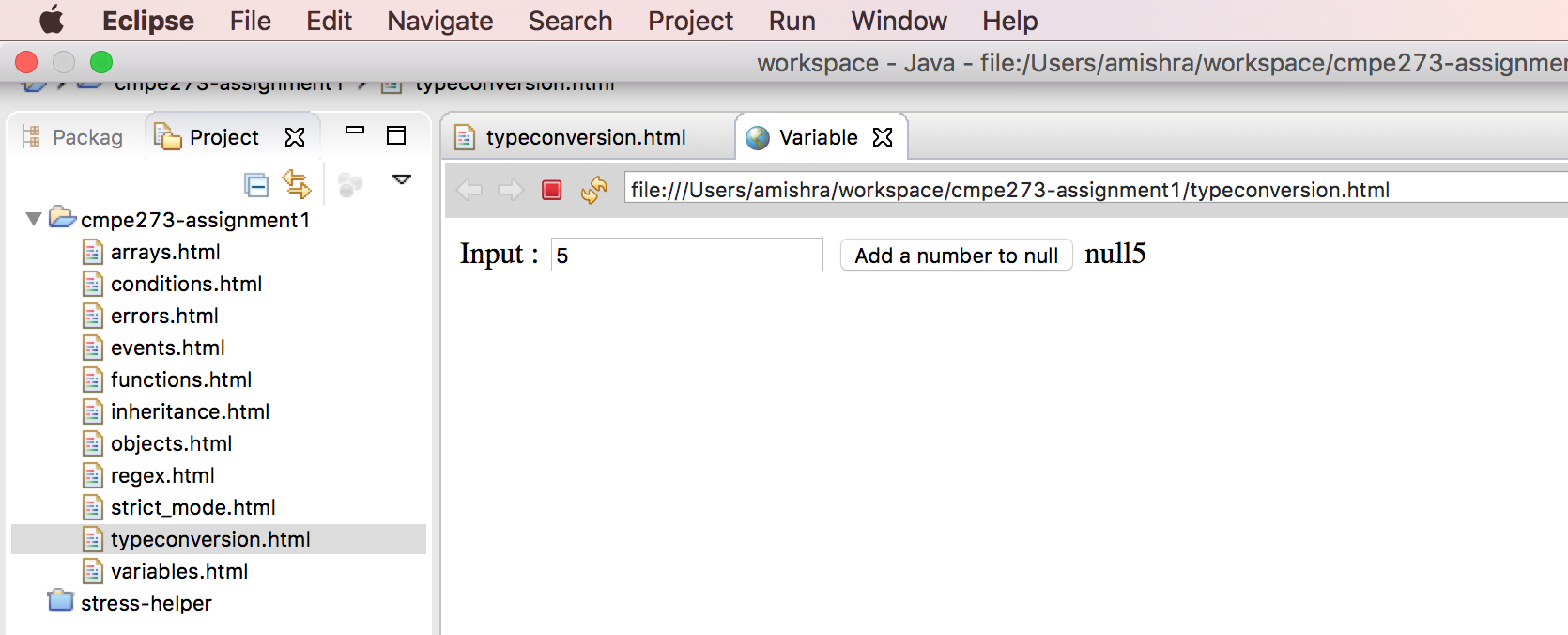
<label id="resultNum"> Check result here! </label> </br></br>

</body>

</html>

**Output:**

****

****

**JSON:**

**Introduction:**

JSON is a format in which data can be sent/received without much overhead unlike XML. It is based on ECMA 3 standard and is accepted by multiple languages like C, C++, Java, Perl, Ruby etc. Parsers are easily available in all programming languages.

**Question:** Create a simple HTML page to store book details in JSON format.

**Code:**

<!DOCTYPE html>

<html>

<head>

<script>

var bookJSON;

function store(){

var name = document.getElementById("name");

var author = document.getElementById("author")

var price = document.getElementById("price")

bookJSON = {

"name" : name.value,

"author":author.value,

"price": price.value

};

document.getElementById("jsonObject").innerHTML=JSON.stringify(bookJSON);

}

</script>

<meta charset="UTF-8">

<title>JSON</title>

</head>

<body>

Enter book details:</br>

Name:<input type="text" id="name" value="Wings of fire"></br>

Author:<input type="text" id="author" value="APJ Abdul Kalam"></br>

Price:<input type="Number" id="price" value="45"></br>

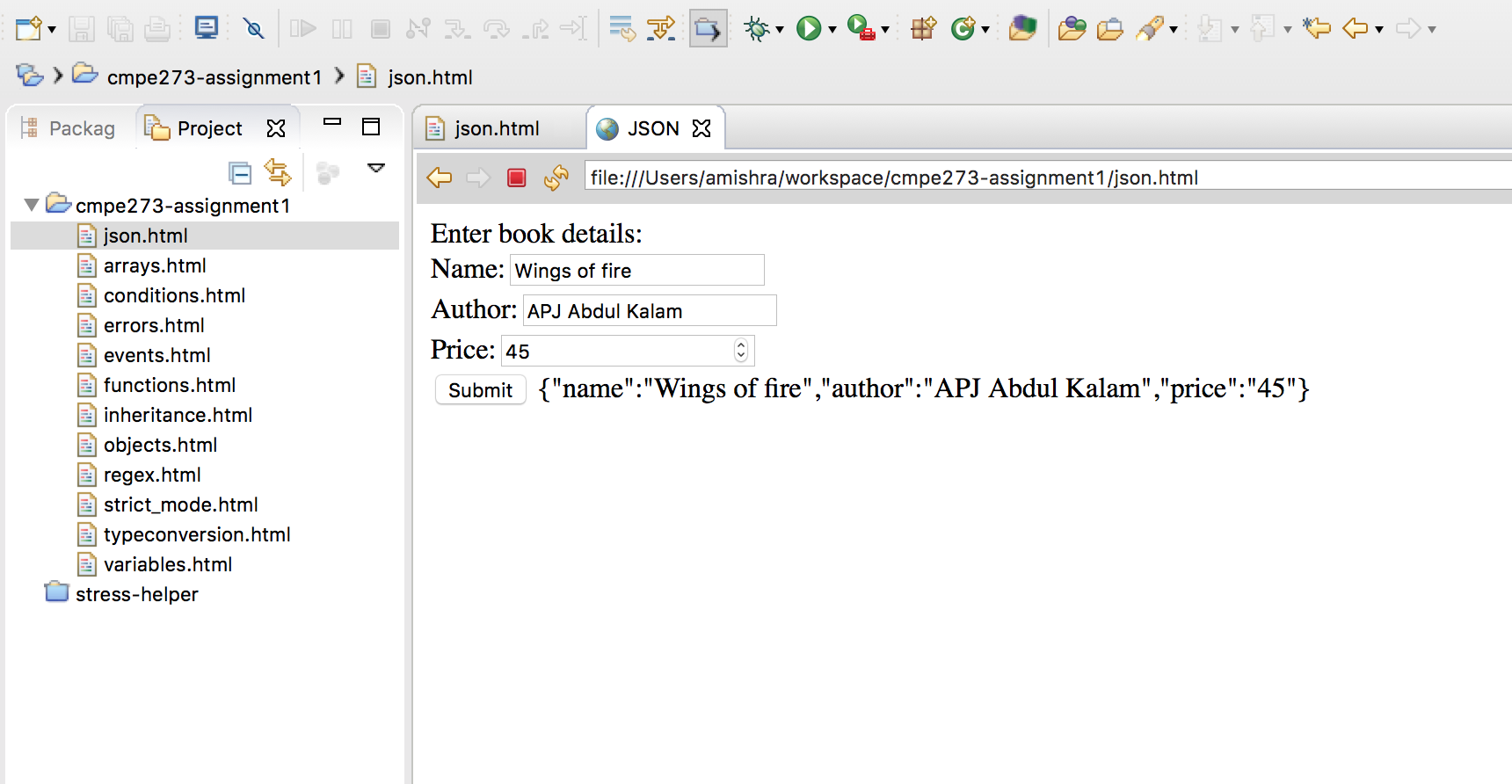
<input type="Submit" value="Submit" onClick="store()">

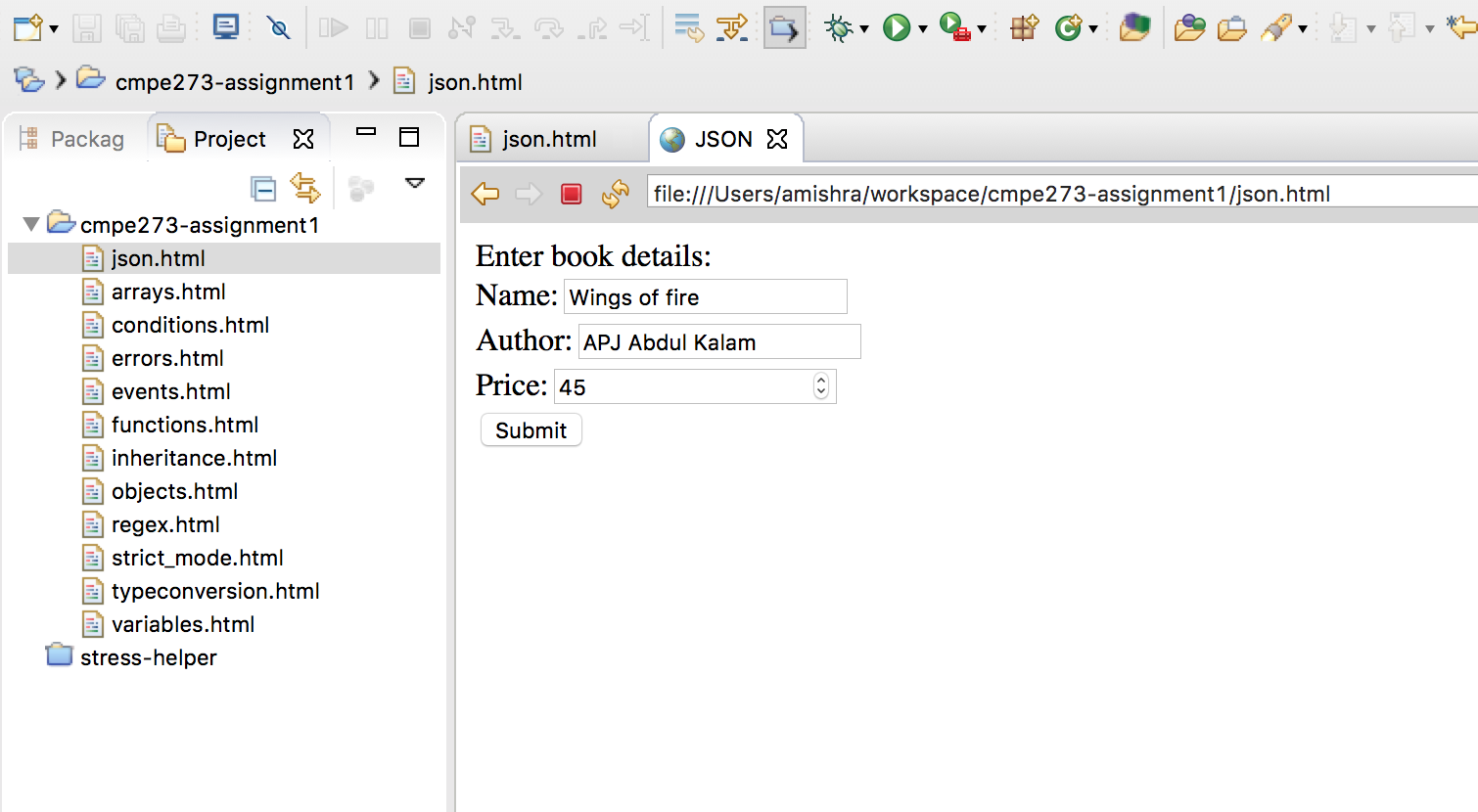
<label id="jsonObject"></label>

</body>

</html>

**Output:**

****

****

**HTML5**

**Video:**

**Introduction:** Before video tag was introduced in HTML5, it needed 3rd party applications like Adobe Flash Player etc in order to embed and play a video within HTML. HTML doesn’t have that feature.

**Question:** Create HTML page to show how video can be embedded and played inside HTML5 code.

**Code:**

<!DOCTYPE html>

<html>

<head>

<script>

</script>

<meta charset="UTF-8">

<title>Video</title>

</head>

<body>

<video width="480" controls=true>

  <source

    src="https://archive.org/download/WebmVp8Vorbis/webmvp8.webm"

    type="video/webm">

  <source

    src="https://archive.org/download/WebmVp8Vorbis/webmvp8\_512kb.mp4"

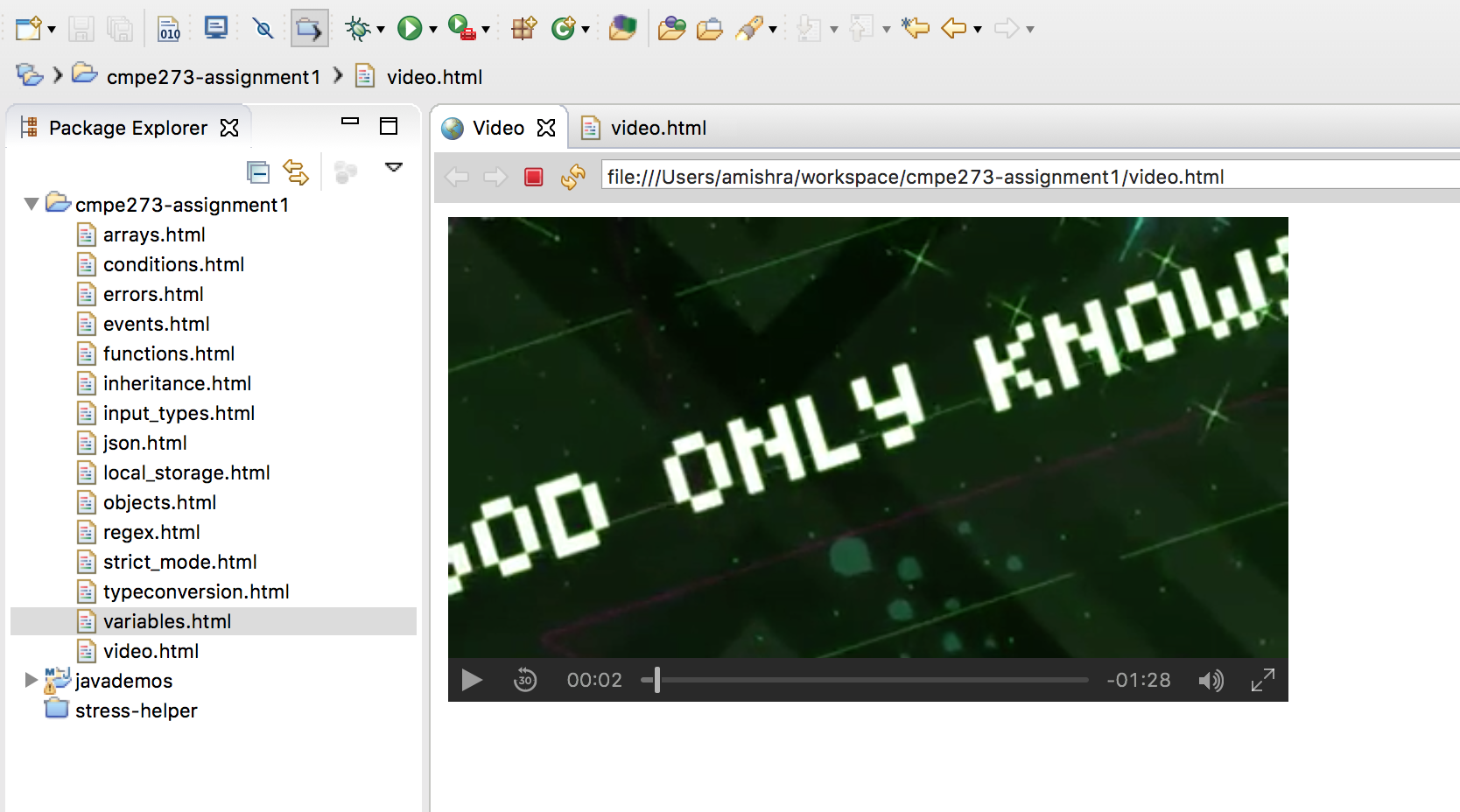
    type="video/mp4">

</video>

</body>

</html>

**Output:**

****

**Java**

Queue:

**Introduction:** Queue is a collection designed with an intent to store data before processing. It typically organizes element in FIFO order but this is not a limitation on this collection type. Queue is basically an interface which is implemented by classes like LinkedList and PriorityQueue. There are thousands of use cases of this data structure like messages stored in a queue for further processing, etc.

**Question:** Demonstrate how queue can be used to store data using a Java program. Print in the same order as the insertion.

**Code:**

package com.cmpe273.assignment1.javademos;

import java.util.LinkedList;

import java.util.List;

public class QueueDemo {

static List<String> queue = new LinkedList<String>();

static String queueString="";

public static List<String> setupQueue(){

queue.add("m1");

queue.add("m2");

queue.add("m3");

return queue;

}

public static String addToQueue(String msg){

queue.add(msg);

return QueueDemo.demoQueue();

}

public static String demoQueue(){

for(String msg : queue){

queueString+=msg+" ";

}

return queueString;

}

}

Junit Tests:

package com.cmpe273.assignment1.javademos;

import org.junit.Before;

import org.junit.Test;

import junit.framework.Assert;

public class QueueTest {

@Before

public void initQueue(){

QueueDemo.setupQueue();

}

@Test

public void testQueue(){

Assert.assertEquals(QueueDemo.demoQueue(), "m1 m2 m3 ");

}

}

**output:**



**Stacks**

**Introduction:** Stack is another useful collection type which implements LIFO i.e. Last In First Out strategy on the elements. It has critical use cases like in graph traversal etc. It has multiple operations like push(), pop(), peek() etc.

**Question:** Demonstrate how a stack can be used to store and pop up elements in LIFO fashion.

**Code:**

package com.cmpe273.assignment1.javademos;

import java.util.Stack;

public class StackDemo {

public static Stack<Integer> stack = new Stack<Integer>();

public static void initStack(){

for(int i=0; i<5; i++){

stack.push(i);

}

}

public static int popStack(){

return stack.pop();

}

public static int pushStack(int i){

stack.push(i);

return stack.peek();

}

public static int peekStack(){

return stack.peek();

}

}

Junit Test:

package com.cmpe273.assignment1.javademos;

import org.junit.Before;

import org.junit.Test;

import junit.framework.Assert;

public class StackTest {

@Before

public void initStack(){

StackDemo.initStack();

}

@Test

public void testPopStack(){

Assert.assertEquals(4, StackDemo.popStack());

}

@Test

public void testPushStack(){

Assert.assertEquals(5, StackDemo.pushStack(5));

}

@Test

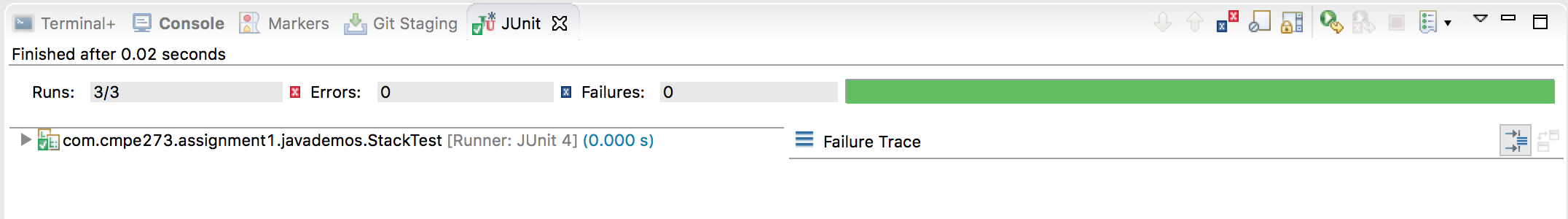
public void testPeekStack(){

Assert.assertEquals(5, StackDemo.pushStack(5));

}

}

**Output:**



Arrays

**Introduction:** Arrays in Java are very frequently used data structures and are known for its fast search time thanks to the indexes that are assigned to the elements starting from 0. Arrays have multiple use cases like storing data for accessing them randomly later.

**Question:** Demonstrate how arrays can be used to store data and accessing them by index.

**Code:**

package com.cmpe273.assignment1.javademos;

public class ArrayDemo {

static int[] array = new int[6];

static String arrayString="";

public static void initArray(){

for(int i=0; i<5; i++){

array[i]=i\*i;

}

}

public static String demoArray(){

for(int i=0; i<5; i++){

arrayString += array[i] + " ";

}

return arrayString;

}

public static int addToArray(int i){

array[array.length-1]=i;

return array.length;

}

}

Junit Tests:

package com.cmpe273.assignment1.javademos;

import org.junit.Before;

import org.junit.Test;

import junit.framework.Assert;

public class ArrayTest {

@Before

public void testInitArray(){

ArrayDemo.initArray();

}

@Test

public void testPrintArray(){

Assert.assertEquals(ArrayDemo.demoArray(), "0 1 4 9 16 ");

}

@Test

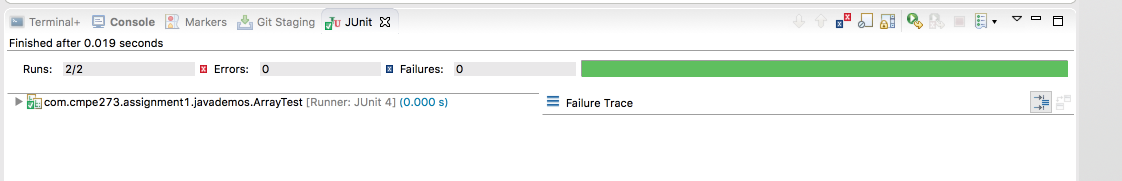
public void testAddToArray(){

Assert.assertEquals(6, ArrayDemo.addToArray(25));

}

}

**Output:**

****

**Interfaces**

**Introduction:** Interface in Java is like a contract that a class has to be abide by, if it is going to implement that interface. Interfaces are in some ways like classes, it is a collection of ‘abstract methods’. Abstract methods need not be implemented right there in the interface but have to be implemented by the classes implementing the interfaces. Only if the class is abstract, then it is free from implementing all methods of the interface.

**Question:** With an example of Animal hierarchy, show how to implement interface and organize the classes.

**Code:**

package com.cmpe273.assignment1.javademos;

public interface Animal {

public String makeNoise();

public String callKid();

}

package com.cmpe273.assignment1.javademos;

public class Cat implements Animal{

public String makeNoise() {

return "Meow Meow!!!";

}

public String callKid(){

return "hey kitten!!!";

}

}

package com.cmpe273.assignment1.javademos;

public class Dog implements Animal {

public String makeNoise() {

return "Whoo whoo!!!";

}

public String callKid(){

return "hey puppy!!!";

}

}

Junit Tests:

package com.cmpe273.assignment1.javademos;

import org.junit.Before;

import org.junit.Test;

import junit.framework.Assert;

public class InterfaceTest {

public static Animal dog, cat;

@Before

public void testInit(){

dog = new Dog();

cat = new Cat();

}

@Test

public void testCatNoise(){

Assert.assertEquals("Meow Meow!!!", cat.makeNoise());

}

@Test

public void testDogNoise(){

Assert.assertEquals("Whoo whoo!!!", dog.makeNoise());

}

@Test

public void testCatCallKid(){

Assert.assertEquals("hey kitten!!!", cat.callKid());

}

@Test

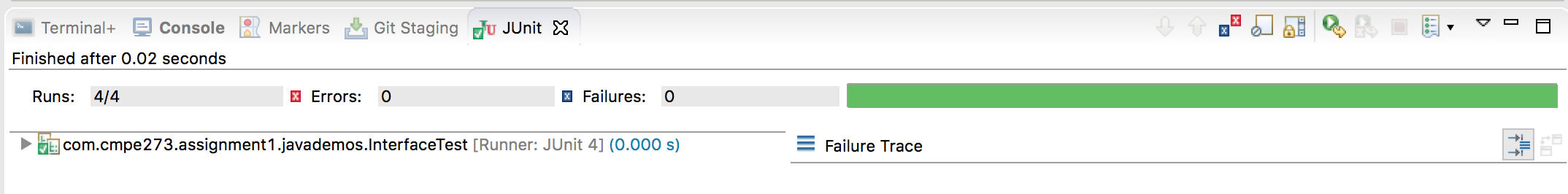
public void testDogCallKid(){

Assert.assertEquals("hey puppy!!!", dog.callKid());

}

}

**Output:**

****

**Collections**

**Introduction:** Collection as the name suggests, represents a group of similar objects. Prior to Java 2, there were some classes like Properties, Dictionary etc were to manipulate the groups of objects. But there was need of unified system, which is precisely provided by Collection framework in later versions of Java.

**Question:** Use a collection type and demonstrate, test update, delete functions, etc.

**Code:**

package com.cmpe273.assignment1.javademos;

import java.util.Hashtable;

public class CollectionDemo {

public Hashtable<Integer, String> ht;

public CollectionDemo(){

ht=new Hashtable<Integer, String>();

}

public int initHashMap(){

ht.put(0, "Zero");

ht.put(1, "One");

return ht.size();

}

public String updateRecord(int i, String s){

ht.put(i, s);

return ht.get(i);

}

public int deleteRecord(int i){

ht.remove(i);

return ht.size();

}

}

Junit tests:

package com.cmpe273.assignment1.javademos;

import org.junit.Before;

import org.junit.Test;

import junit.framework.Assert;

public class CollectionTest {

CollectionDemo cd = new CollectionDemo();

@Before

public void testInitMap(){

cd.initHashMap();

}

@Test

public void testUpdateMap(){

Assert.assertEquals("Updated", cd.updateRecord(0, "Updated"));

}

@Test

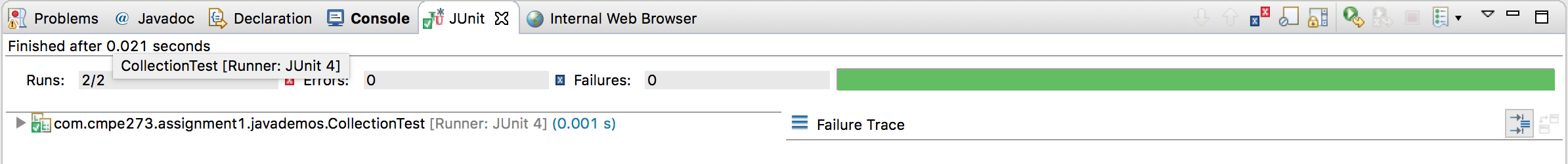
public void testDeleteMapRecord(){

Assert.assertEquals(1, cd.deleteRecord(0));

}

}

**output:**



Generics

**Introduction:** Using Java generics allow us to write generic methods or classes. There is also a compile time safety to catch invalid or incompatible types. Apart from compile-time checking, there is also type-safety and type casting is not required.

**Question:** Demonstrate use of generics through an example.

**Code:**

package com.cmpe273.assignment1.javademos;

public class GenericDemo<E> {

public E e;

public GenericDemo(E e){

        this.e = e;

}

@SuppressWarnings("hiding")

public <E> String printArray(){

return e.toString();

}

}

Junit tests:

package com.cmpe273.assignment1.javademos;

import org.junit.Before;

import org.junit.Test;

import junit.framework.Assert;

public class GenericTest{

public GenericDemo<?> gd1, gd2;

@Before

public void initStringGeneric(){

gd1 = new GenericDemo<String>("genDemo");

gd2 = new GenericDemo<Integer>(5);

}

@Test

public void testPrintGeneric(){

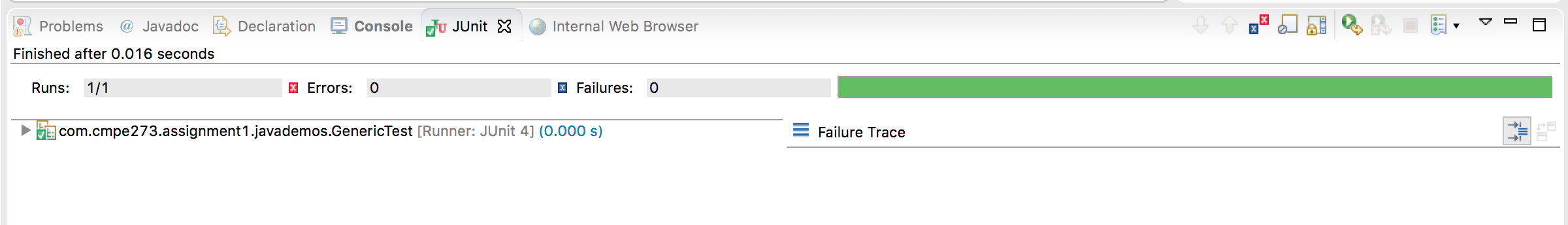
Assert.assertEquals("genDemo", gd1.printArray());

Assert.assertEquals("5", gd2.printArray());

}

}

**Output:**



Multi-threading

**Introduction:** Java is a multithreaded language which means we can write multi-threaded applications using Java.

In regular programs, only one thread that is main thread runs the program. In multi-threaded programs one can create multiple threads to run a program. Multi-tasking when multiple processes share a common resource such as processor. In multi-threaded apps, you further divide operations from a specific process into multiple threads for the faster and parallel processing.

**Question:** Write a simple incrementer / decrementer program and demonstrate how it can be triggered using multiple threads. Write junit tests that run the program through multiple threads and pass the test.

**Code:**

package com.cmpe273.assignment1.javademos;

public class MultithreadingDemo {

    public int count=100;

    public int decrementCount()

    {

        count--;

        return count;

    }

}

Junit tests:

package com.cmpe273.assignment1.javademos;

import org.junit.Test;

import org.junit.runner.RunWith;

import com.anarsoft.vmlens.concurrent.junit.ConcurrentTestRunner;

import junit.framework.Assert;

@RunWith(ConcurrentTestRunner.class)

public class CounterTest {

private MultithreadingDemo demo = new MultithreadingDemo();

@Test

public void testMultithreading()

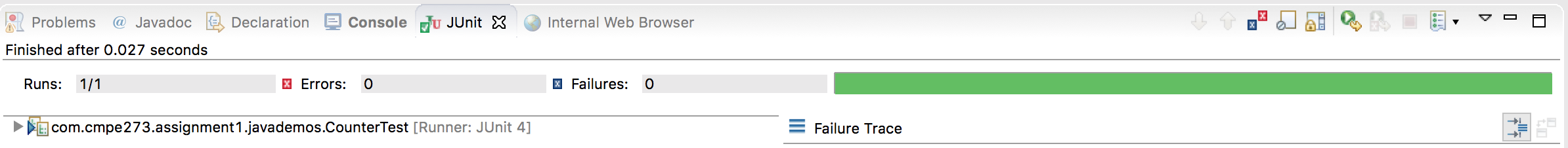
{

Assert.assertEquals(96, 3, demo.decrementCount());

}

}

**Output:**

****