CMPE 273

Enterprise Distributed Systems

Refreshers Assignment

Jaykumar Patel

SID: 012548432

A. JavaScript

1. Variables:

* Variables must have unique names known as identifiers and can assigned values with assignment (=) operator (e.g. Var num = 5;)
* Variable names can only start with a letter or $ or underscore sign (\_) and contain special characters.
* Variable names are case sensitive.
* JavaScript keywords cannot be used as identifiers.

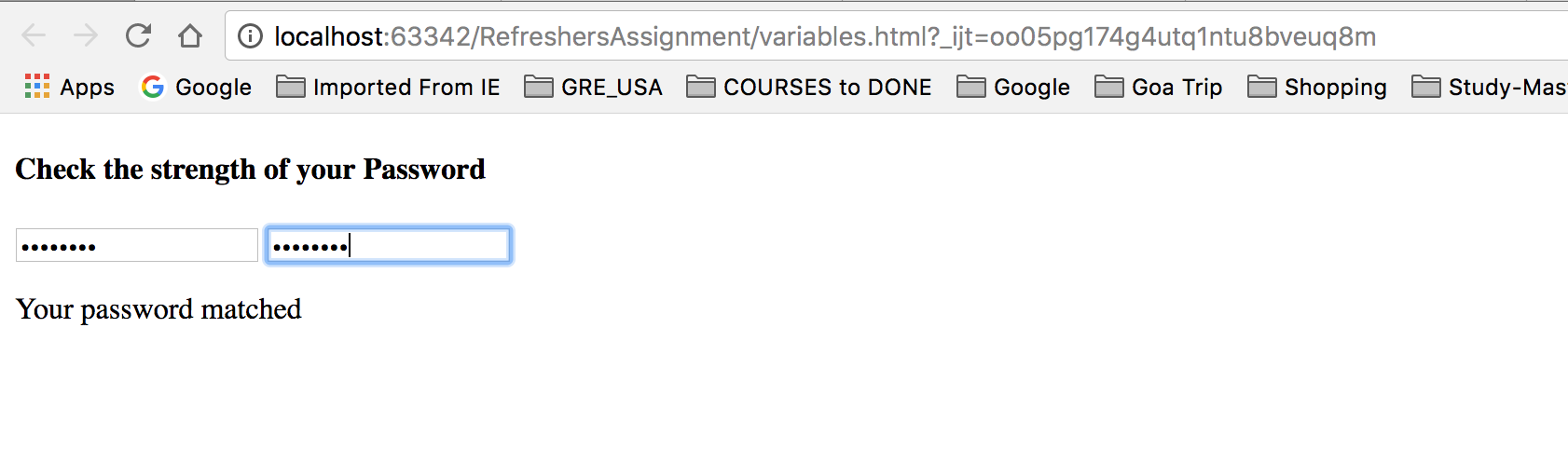
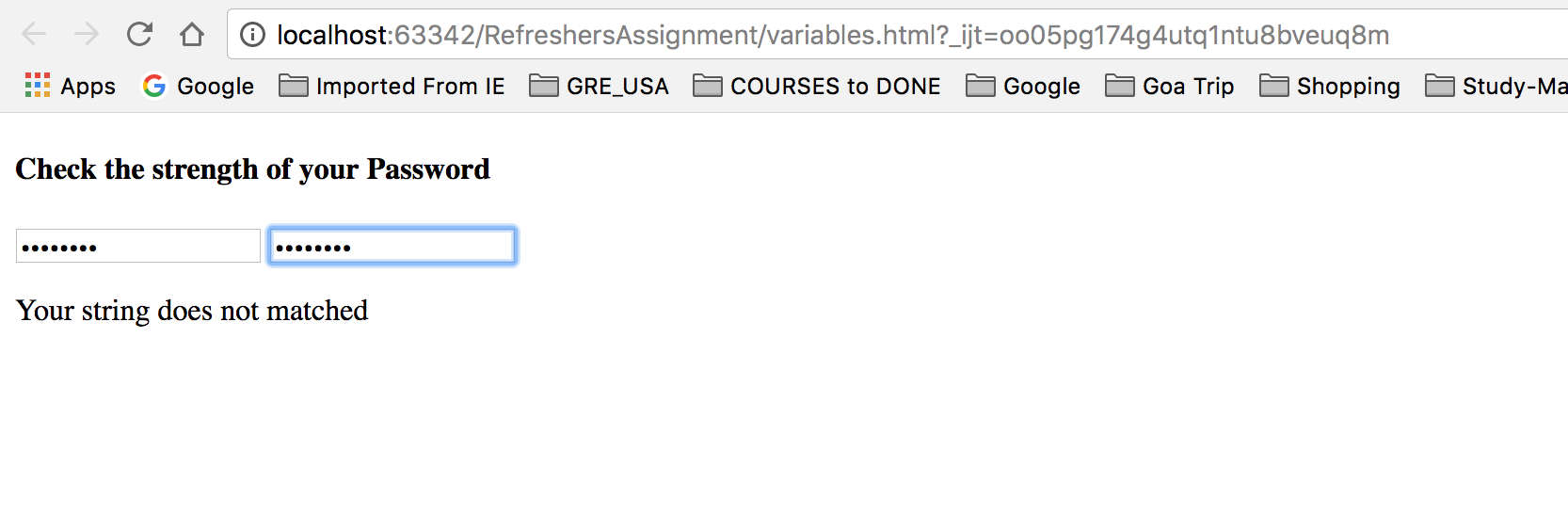
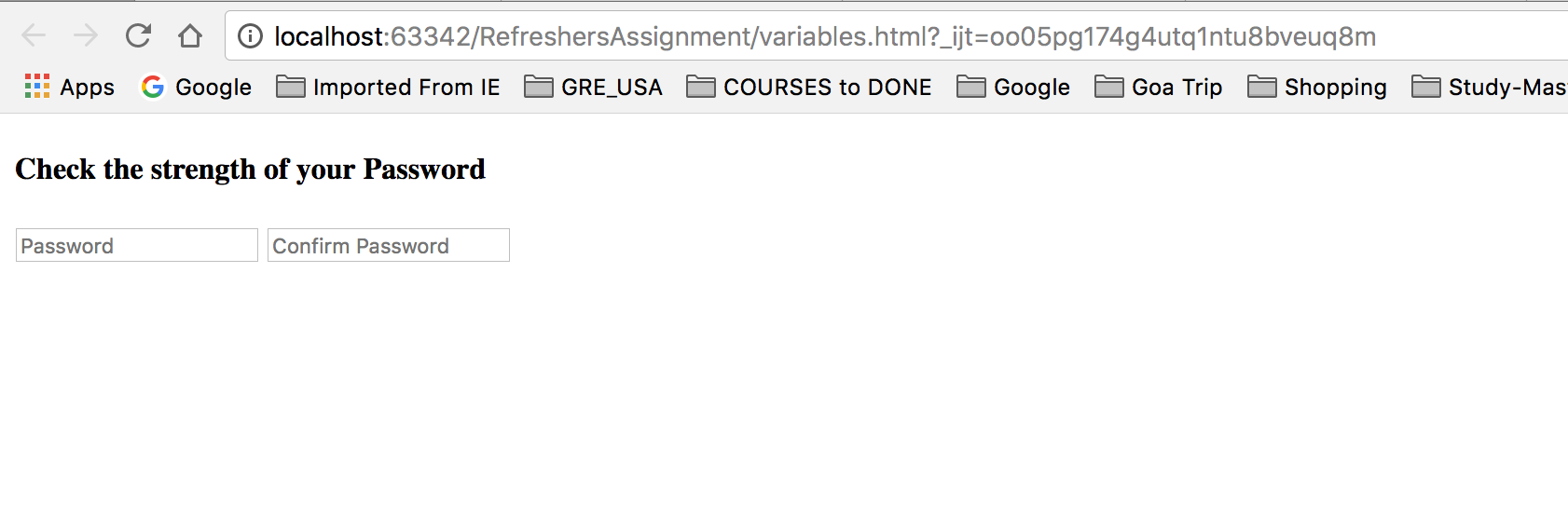
**Program Scenario**: Here, two inputs of password type will be taken from the user to match his password. When user starts typing in the second box, on every KeyUp event, function check() will be called and match the string input. If it matches, it shows, “Your password matched” or it will show “Your password does not matched”. It will also show the message “Password must contain at least eight characters!” to the user password is weak.

HTML

<!DOCTYPE **html**>  
<**html lang="en"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>Value Matching</**title**>  
 <**script src="variables.js"**></**script**>  
</**head**>  
<**body**>  
 <**h4**>Check the strength of your Password</**h4**>  
 <**input type="password" id="box1" placeholder="Password"**>  
 <**input type="password" id="box2" placeholder="Confirm Password" onkeyup="***check*()**"**>  
 *<!--input type="button" value="Check the Match" onclick="check()"-->* <**p id="input"**></**p**>  
 <**p id="length"**></**p**>  
</**body**>  
</**html**>

JavaScript

**function** *check*(){  
 **var** firstbox= ***document***.getElementById(**"box1"**).**value**;  
 **var** secondbox = ***document***.getElementById(**"box2"**).**value**;  
 **for**(**i**=0; **i**<secondbox.**length**; **i**++){  
 **if**(secondbox == firstbox){  
 ***document***.getElementById(**"input"**).innerHTML = **""**;  
 **if** (firstbox != **""** && secondbox != **""**) {  
 **if** (firstbox.**length** < 8) {  
 ***document***.getElementById(**"length"**).innerHTML = **"Password must contain at least eight characters!"**;  
 }  
 **else  
 *document***.getElementById(**"input"**).innerHTML = **"Your password matched"**;  
 }  
 }  
 **else** { ***document***.getElementById(**"input"**).innerHTML = **"Your password does not matched"**;  
 ***document***.getElementById(**"length"**).innerHTML = **""**;  
 }  
 }  
}

Output:

1. Objects:

* JavaScript objects work as the data containers for the execution.
* Objects work as Name: Value pairs in JavaScript (e.g. var fullname = {firstname : “Jay” , lastname : “Patel”})
* We can store multiple values in the object and access them with the name we used for the data storing.

Program Scenario: Here, User details will be asked. If User is from Software Engineering department then only they will give a link to Software Engineering department of SJSU. But if user is not from Software Engineering department, it will show message that user cannot access the link. Where, after entering all details and pressing button, all the details will be stored in the **Object named Person** and details will be derived with that Object only.

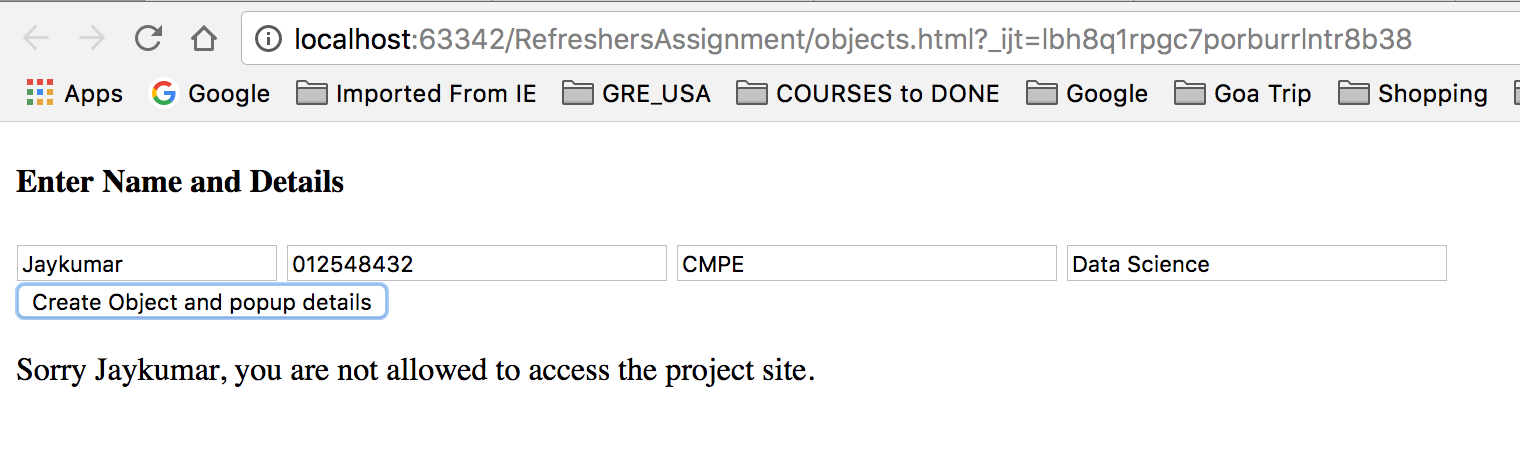
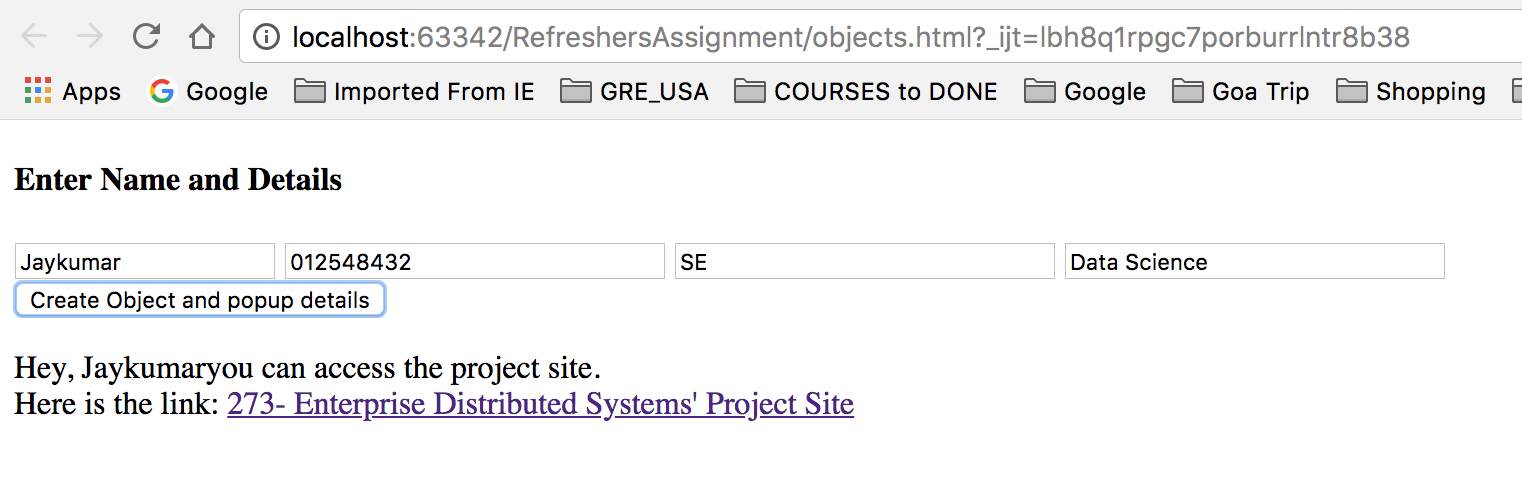
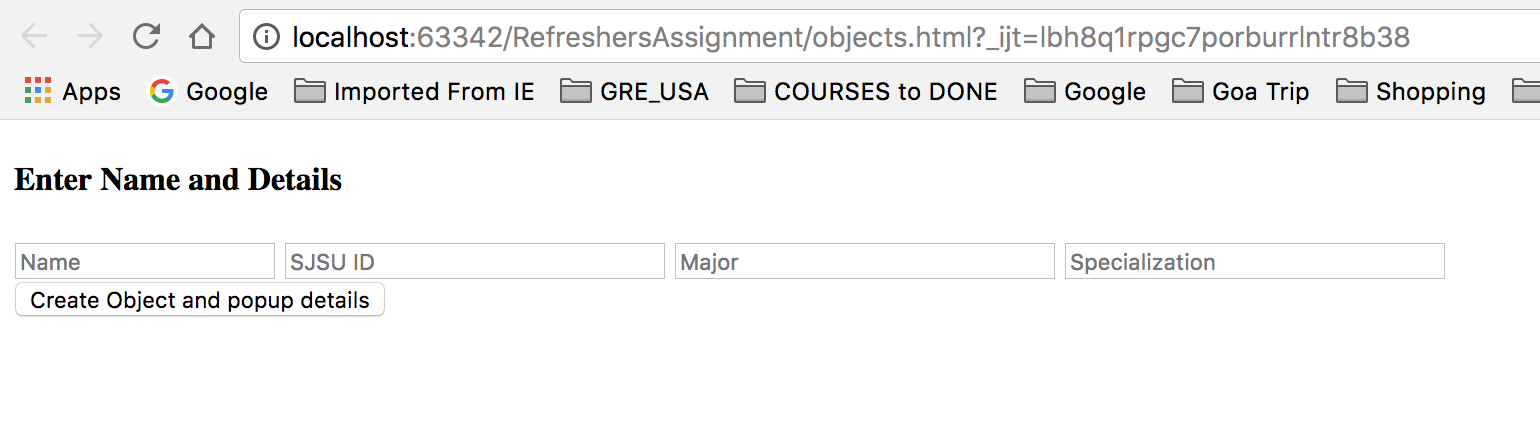
HTML

<!DOCTYPE **html**>  
<**html lang="en"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>Object Data</**title**>  
 <**script src="objects.js"**></**script**>  
</**head**>  
<**body**>  
<**h4**>Enter Name and Date of Birth to be stored in object and get popup with the same details</**h4**>  
<**input type="text" id="box1" placeholder="Name" size="20"**>  
<**input type="text" id="box2" placeholder="SJSU ID" size="30"**>  
<**input type="text" id="box3" placeholder="Major" size="30"**>  
<**input type="text" id="box4" placeholder="Specialization" size="30"**><**br**>  
<**input type="button" value="Create Object and popup details" onclick="***object*()**"**>  
<**p id="input"**></**p**>  
</**body**>  
</**html**>

JavaScript

**function** *object*(){  
 **var** title = **"273- Enterprise Distributed Systems' Project Site"**;  
 **var** website= title.link(**"http://msse.sjsu.edu/data-science"**);  
 **var** person = {  
 **name**:***document***.getElementById(**"box1"**).**value**,  
 **sid**:***document***.getElementById(**"box2"**).**value**,  
 **major**:***document***.getElementById(**"box3"**).**value**,  
 **specialization**:***document***.getElementById(**"box4"**).**value** };  
 **if**(person.**major**==**"se"** ||person.**major**==**"SE"** || person.**specialization**==**"ds"** || person.**major**==**"DS"** || person.**major**==**"Data Science"** ||person.**major**==**"Software Engineering"**){  
 ***document***.getElementById(**"input"**).innerHTML = **"Hey, "**+ person.**name** + **"you can access the project site. </br>Here is the link: "** + website;  
 }  
 **else**{  
 ***document***.getElementById(**"input"**).innerHTML = **"Sorry "**+person.**name**+**", you are not allowed to access the project site."**;  
 }  
}

Output:



1. Functions

* It is a block of code which will execute particular task as Java Functions.
* Function can be made by **function**keyword and typing name of function after it. (e.g. function calculate(){})
* Every function works only when something invokes it and execute the code written inside the curly brackets: **{}**
* Parameters can be passed into the function by passing the parameters into the function’s round brackets: **()**

Program Scenario: Here, user will input the number inside the box and he can find the square, cube, four times or five time multiplication of the number. For every function, there is different button. Every button will call specific function to perform the task and display the output.

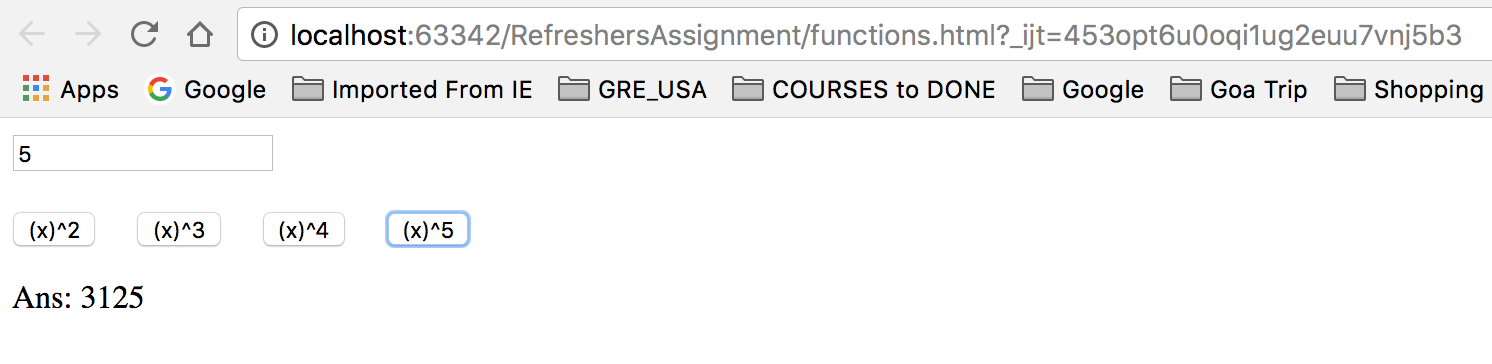
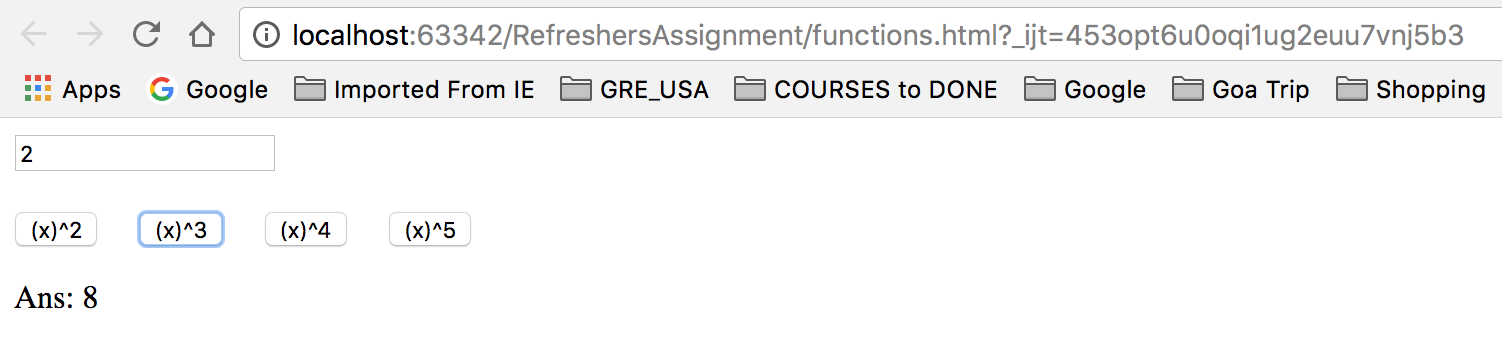
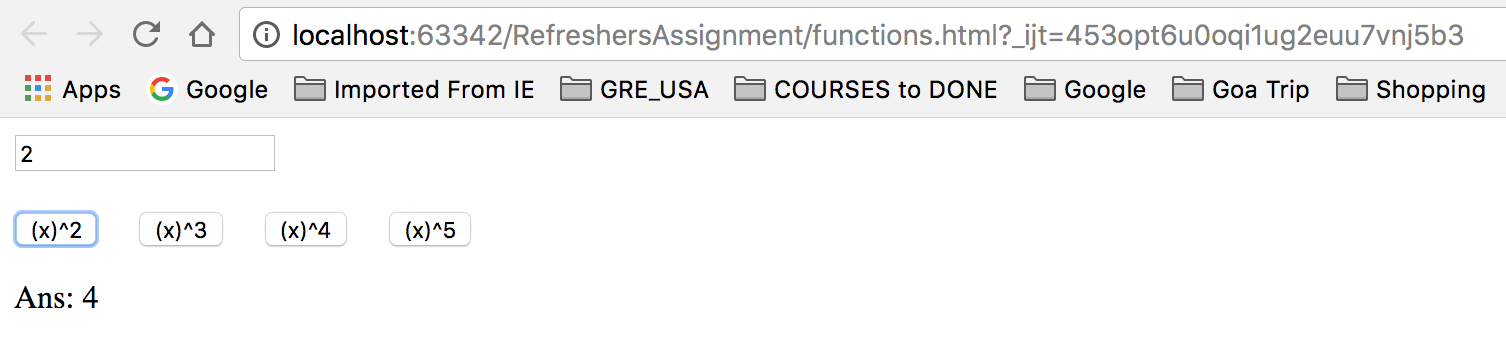
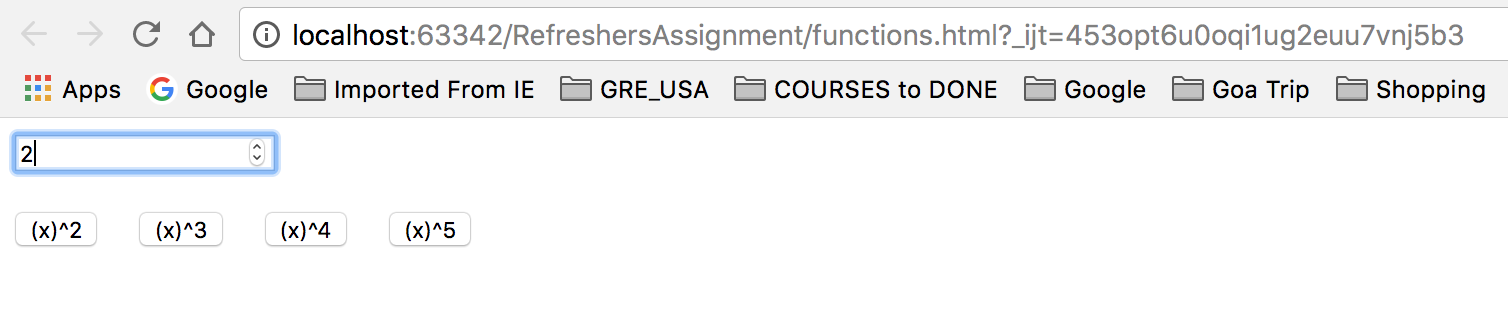
HTML

<!DOCTYPE **html**>  
<**html lang="en"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>Powers of the numbers</**title**>  
 <**script src="functions.js"**></**script**>  
</**head**>  
<**body**>  
<**input type="number" id="box1" placeholder="Enter any number" size="10"**><**br**><**br**>  
<**input type="button" value="(x)^2" onclick="***square*()**"**>**&emsp;**<**input type="button" value="(x)^3" onclick="***cube*()**"**>**&emsp;**<**input type="button" value="(x)^4" onclick="***restto4*()**"**>**&emsp;**<**input type="button" value="(x)^5" onclick="***restto5*()**"**>**&emsp;**<**p id="input"**></**p**>  
</**body**>  
</**html**>

JavaScript

**function** *square*(){  
 **var** x = ***document***.getElementById(**"box1"**).**value**;  
 ***document***.getElementById(**"input"**).innerHTML = **"Ans: "**+ x\*x;  
 **return** x\*x;  
}  
**function** *cube*(){  
 **var** x = ***document***.getElementById(**"box1"**).**value**;  
 ***document***.getElementById(**"input"**).innerHTML = **"Ans: "**+ x\*x\*x;  
 **return** x\*x\*x;  
}  
**function** *restto4*(){  
 **var** x = *square*() \* *square*();  
 ***document***.getElementById(**"input"**).innerHTML = **"Ans: "**+ x;  
}  
**function** *restto5*(){  
 **var** x = *square*() \* *cube*();  
 ***document***.getElementById(**"input"**).innerHTML = **"Ans: "**+ x;  
}

Output



1. Events

* JavaScript reacts on events which will execute on HTML elements.
* Different HTML elements have different event scenario which will perform the event if you have mentioned something for that particular event.

**Program Scenario:** Here, on user input three events will work. When key is down, screen color will turn to red. When key is pressed, screen color will be blue and when key is up, screen color will be black. Also when mouse pointer comes over the photo, it will make the photo zoom and when pointer goes out, photo will go back to the normal size. There are two buttons which will change the background color to grey and normal after pressing with onclick event.

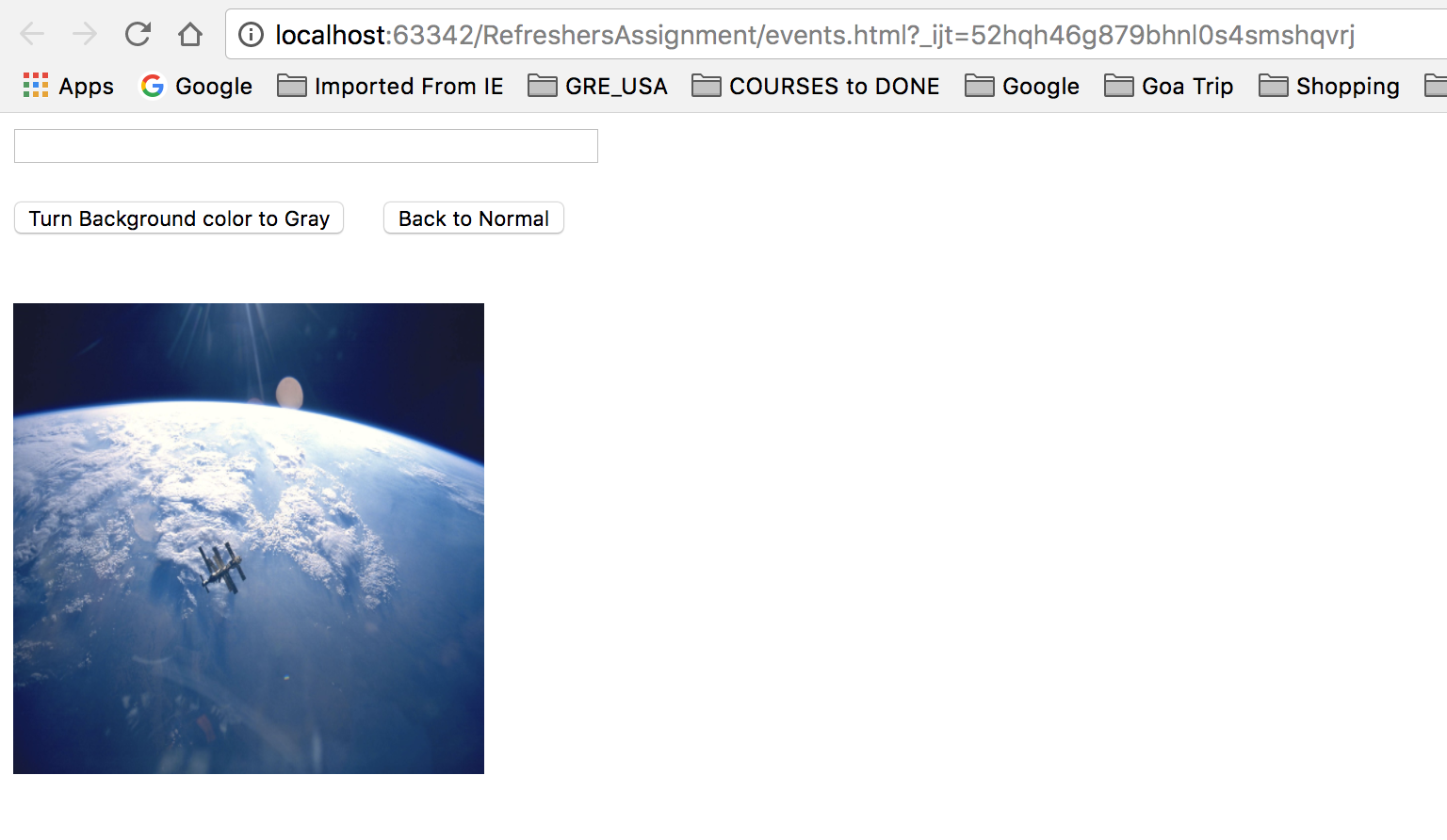
HTML

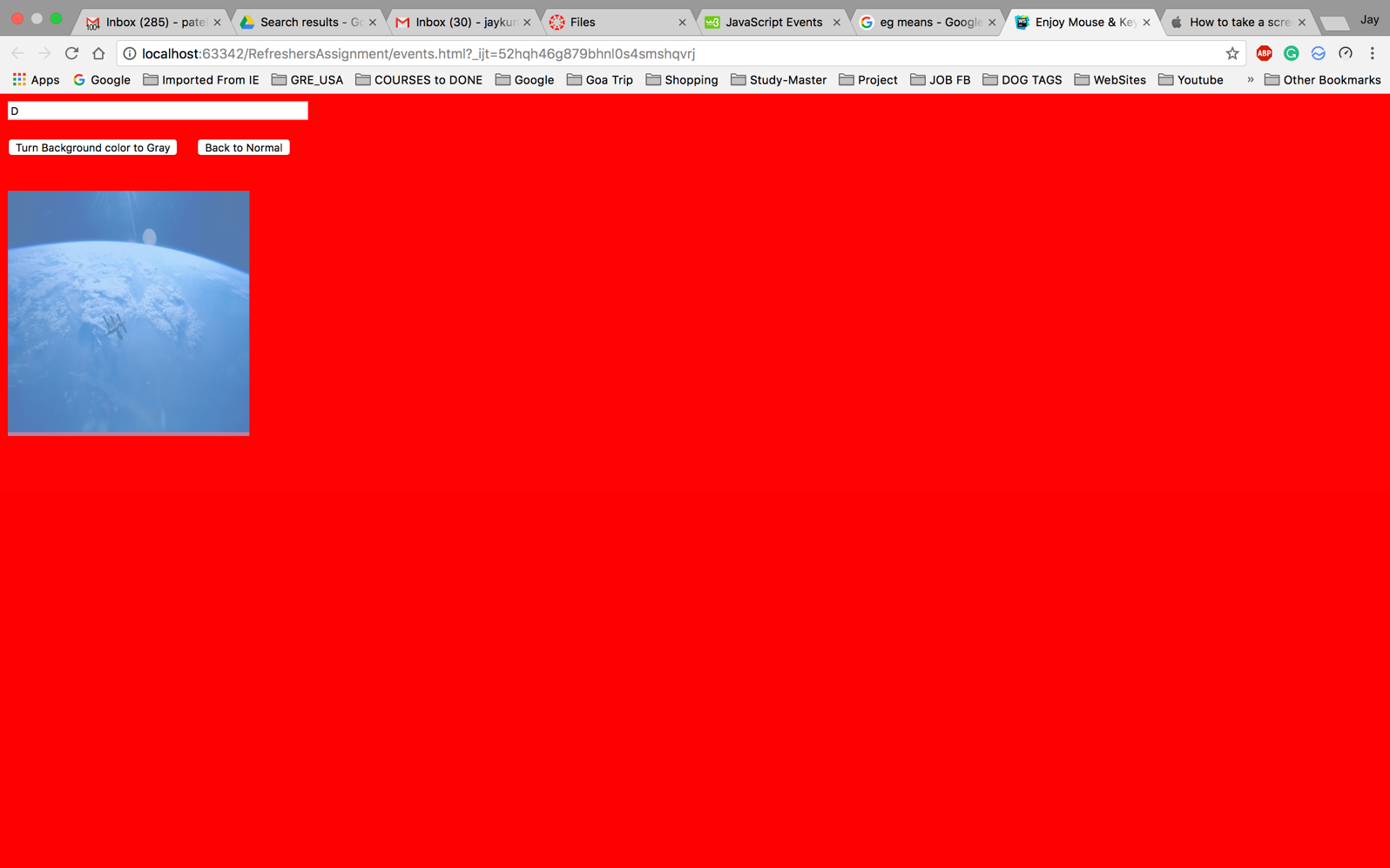
<!DOCTYPE **html**>  
<**html lang="en"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>Enjoy Mouse & Keyboard Events</**title**>  
 <**script src="events.js"**></**script**>  
</**head**>  
<**body**>  
<**input type="text" onkeypress="***kbp*()**" onkeydown="***kbd*()**" onkeyup="***kbu*()**" size="50"**><**br**><**br**>  
<**input type="button" value="Turn Background color to Gray" onclick="***gray*()**"**>**&emsp;**<**input type="button" value="Back to Normal" onclick="***white*()**"**>**&emsp;**<**br**><**br**><**br**>  
<**img onmouseout="***normal*(**this**)**" onmouseover="***zoom*(**this**)**" src="https://images3.alphacoders.com/235/thumb-1920-23517.jpg" alt="Google"**>  
</**body**>  
</**html**>

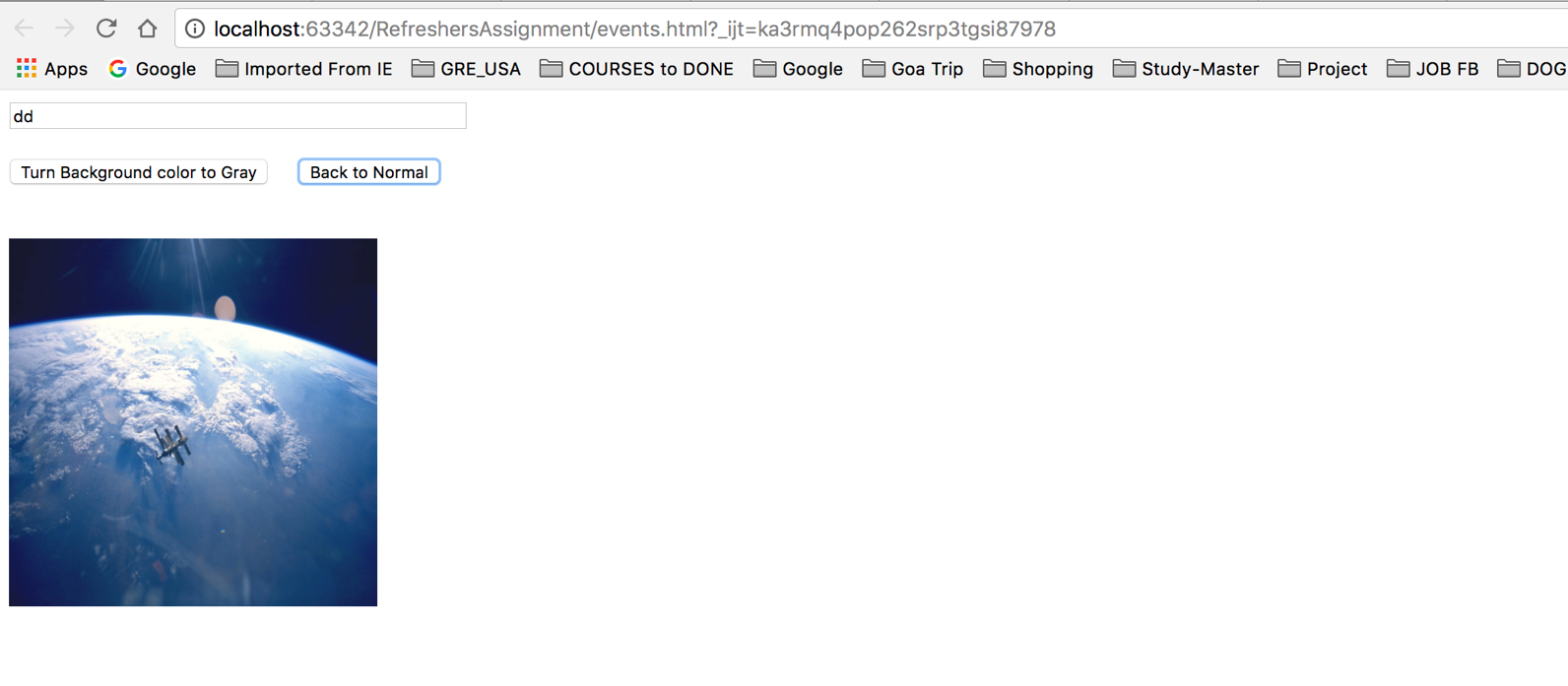
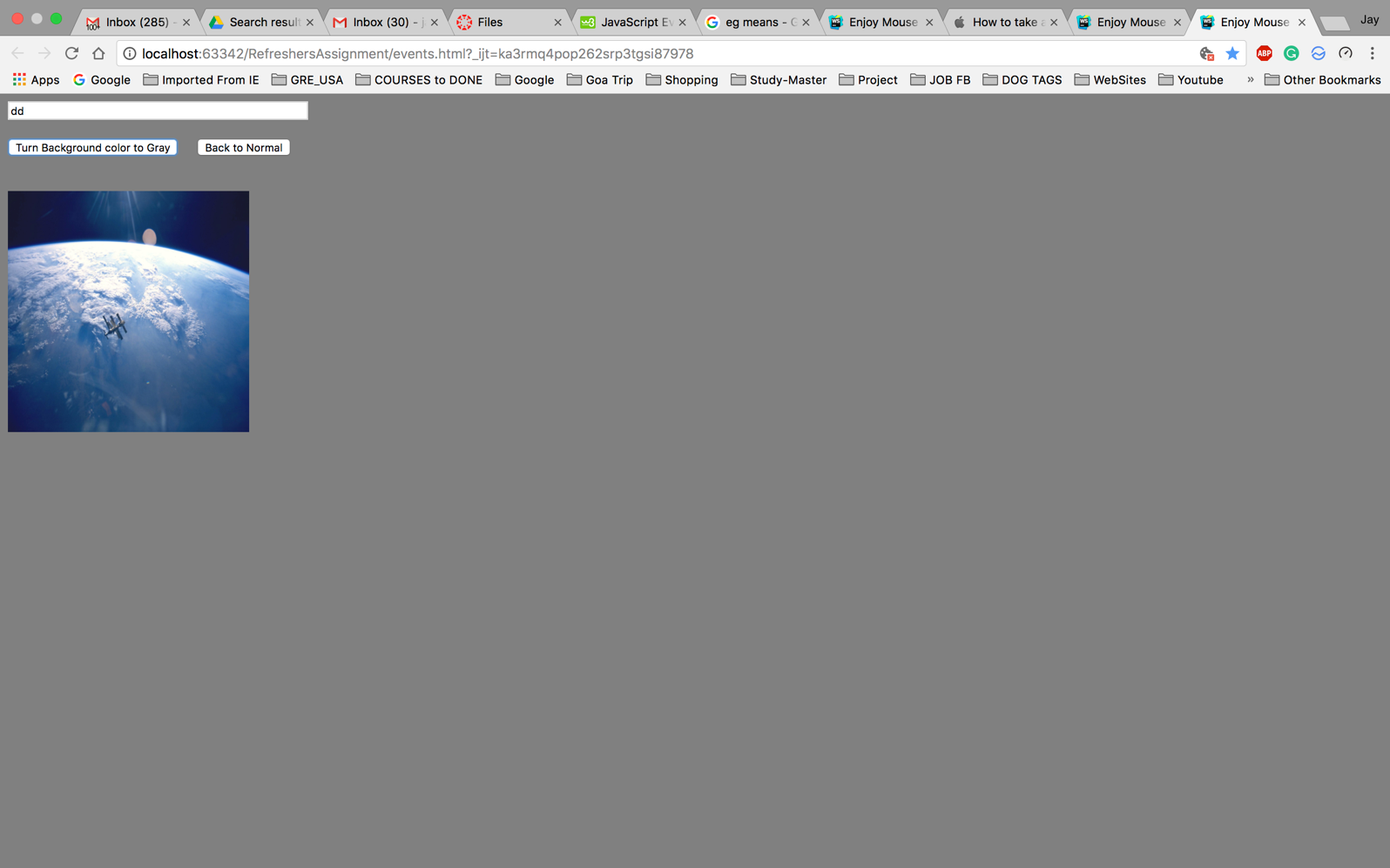
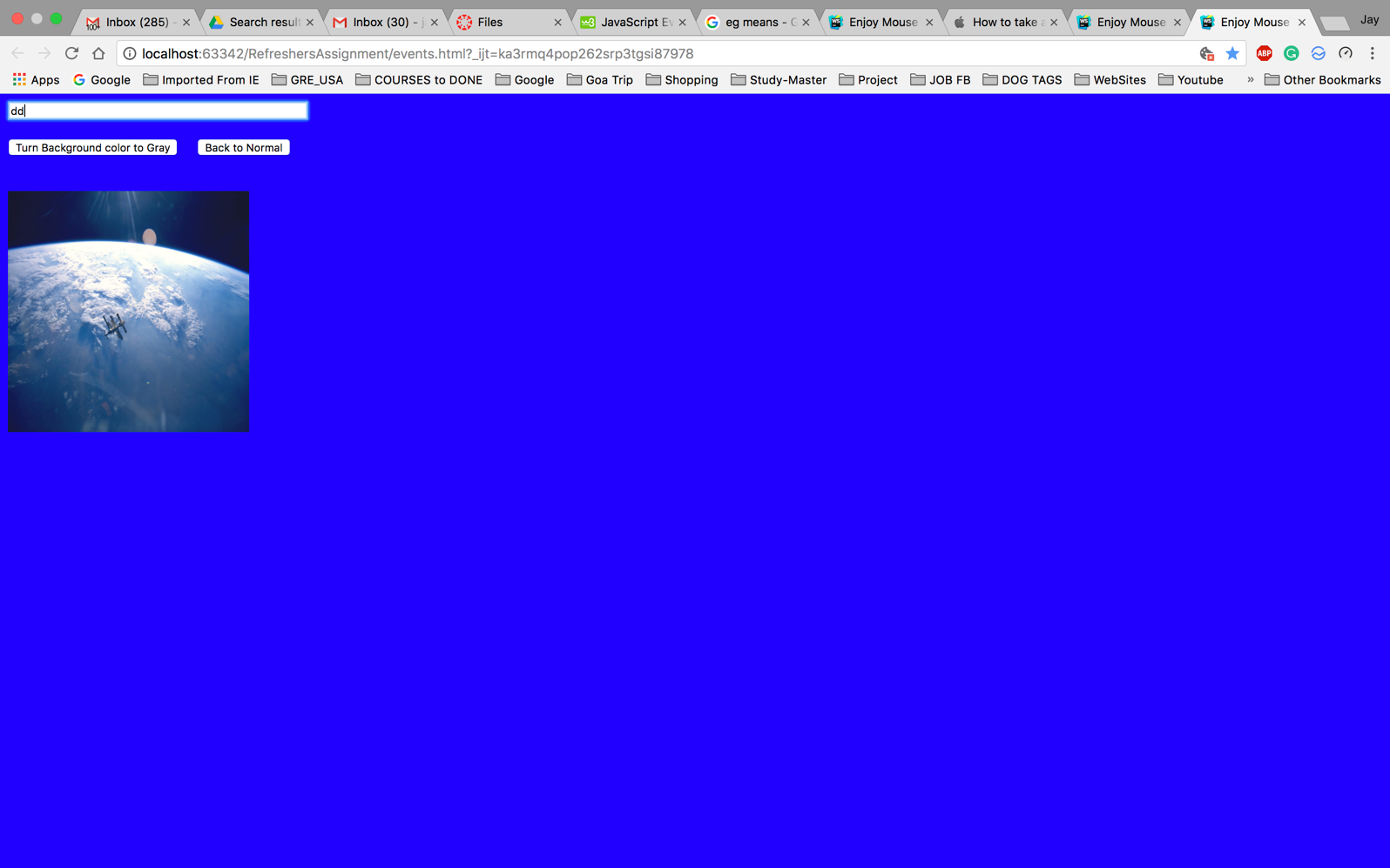
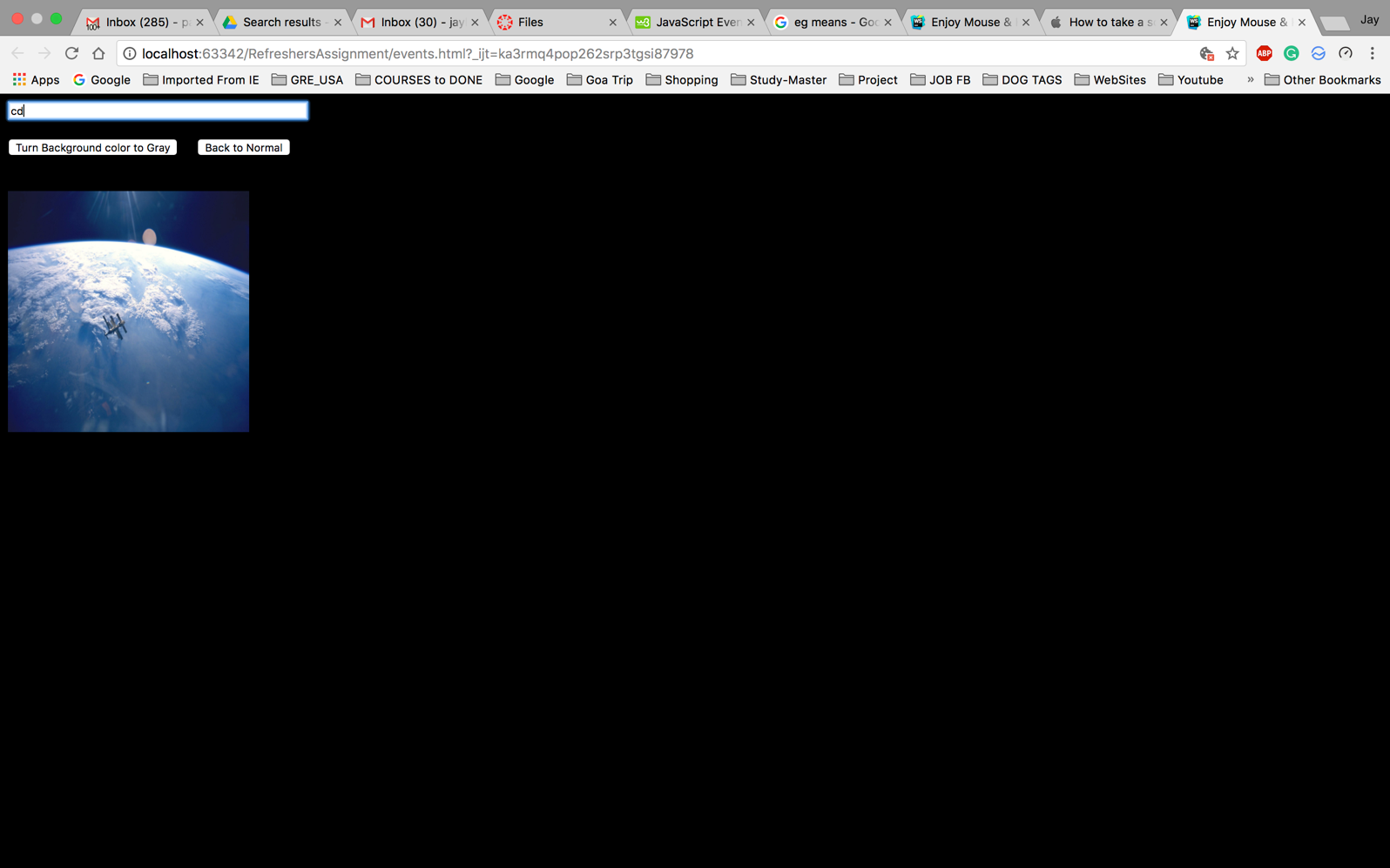
JavaScript

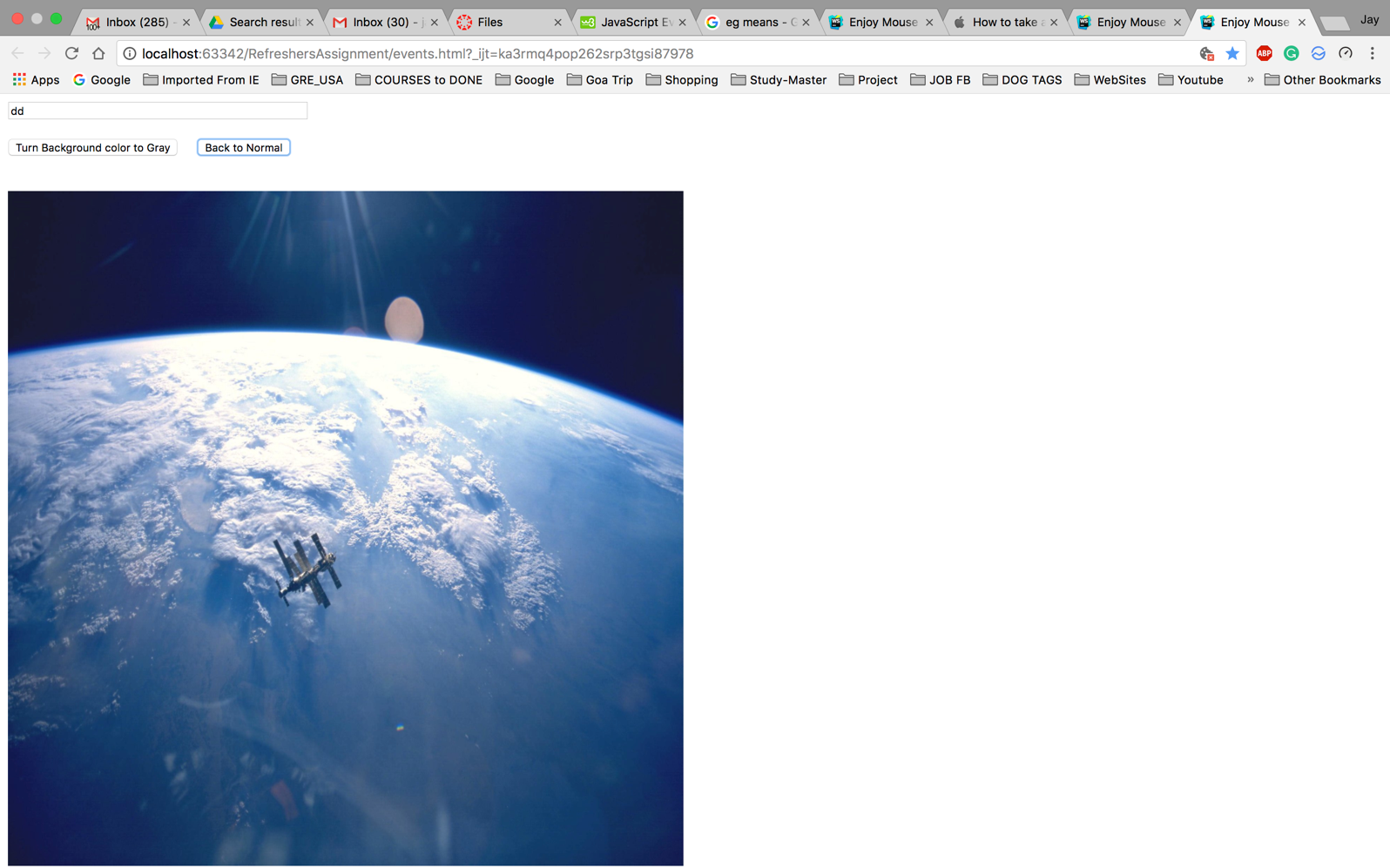
**function** *gray*(){  
 ***document***.body.style.backgroundColor = **"Gray"**;  
}  
**function** *white*(){  
 ***document***.body.style.backgroundColor = **"White"**;  
}  
**function** *zoom*(x){  
 x.style.height = **"700px"**;  
 x.style.width = **"700px"**;  
}  
**function** *normal*(x){  
 x.style.height = **"250px"**;  
 x.style.width = **"250px"**;  
}  
**function** *kbp*(){  
 ***document***.body.style.backgroundColor = **"Blue"**;  
}  
**function** *kbd*(){  
 ***document***.body.style.backgroundColor = **"Red"**;  
}  
**function** *kbu*(){  
 ***document***.body.style.backgroundColor = **"Black"**;  
}

Output:









1. Arrays

* It can store multiple values with one variable name.
* You can refer every items by referencing with index number.

**Program Scenario:** Here, we take the input from the user to insert numbers. After pressing Enter number, it adds that number to the array. If you write particular number and click remove this number, it will delete that number from the array. And when you specify the index, starts from 0, it will delete the element from the particular index.

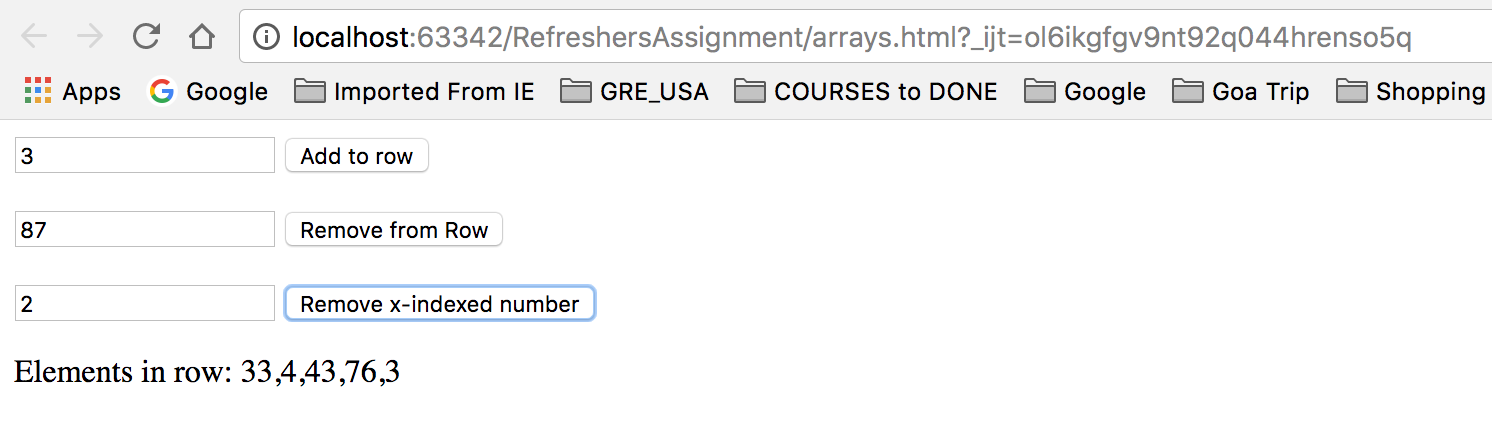
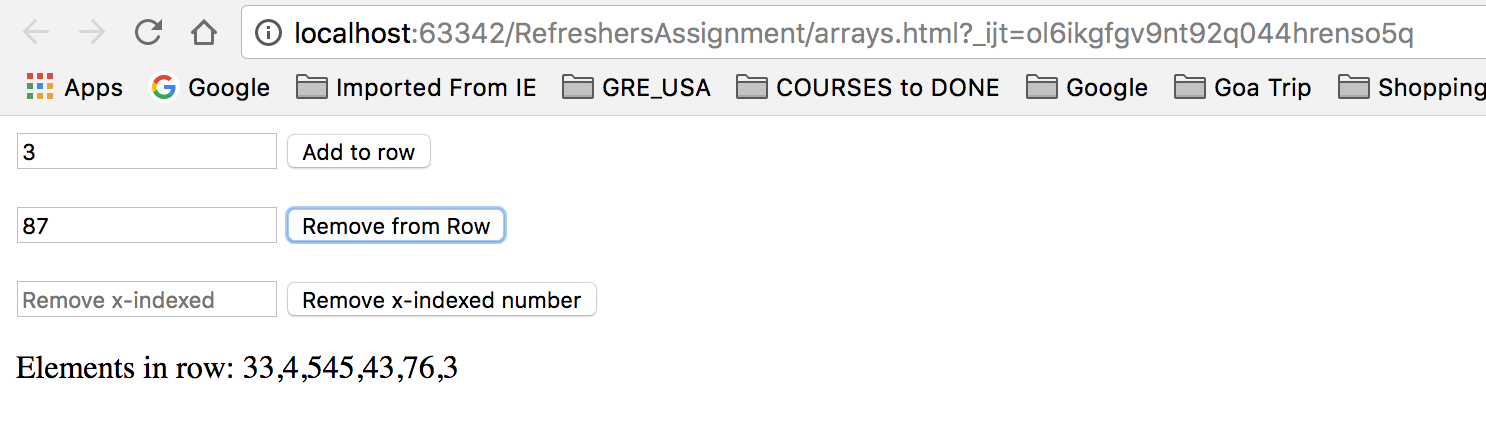
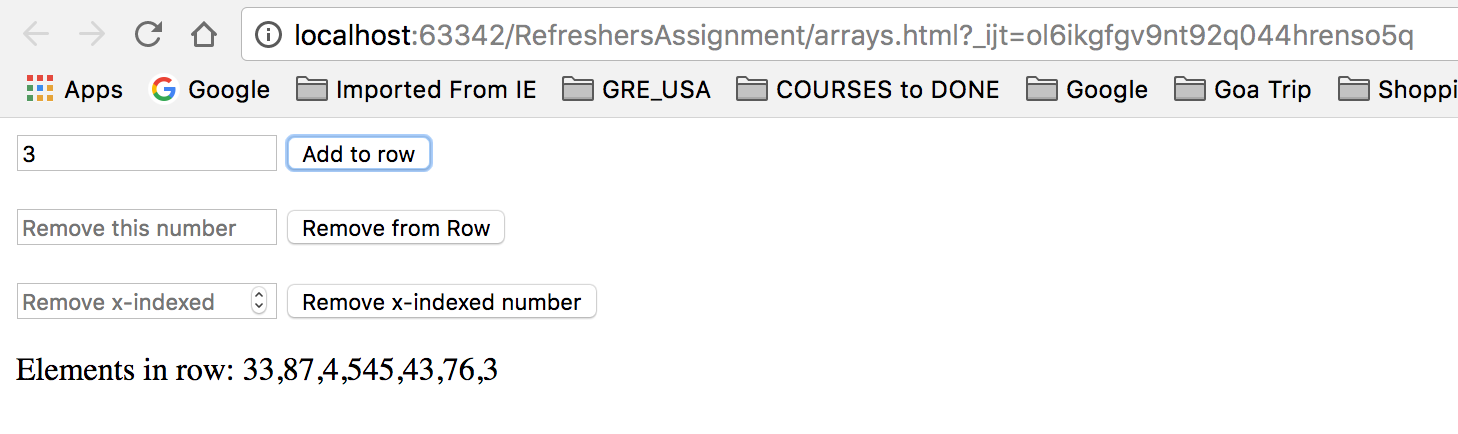
HTML

<!DOCTYPE **html**>  
<**html lang="en"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>Create and manage the row</**title**>  
 <**script src="arrays.js"**></**script**>  
</**head**>  
<**body**>  
<**input type="number" id="box1" placeholder="Enter Number"**>  
<**input type="button" value="Add to row" onclick="***addtoarray*()**"**><**br**><**br**>  
<**input type="number" id="box2" placeholder="Remove this number"**>  
<**input type="button" value="Remove from Row" onclick="***remove*()**"**><**br**><**br**>  
<**input type="number" id="box3" placeholder="Remove x-indexed"**>  
<**input type="button" value="Remove x-indexed number" onclick="***removex*()**"**>  
<**p id="input"**></**p**>  
<**p id="input1"**></**p**>  
</**body**>  
</**html**>

JavaScript

**var *row***= **new *Array***();  
**function** *addtoarray*(){  
 **var** add=***document***.getElementById(**"box1"**).**value**;  
 ***row***.push(add);  
 ***document***.getElementById(**"input"**).innerHTML = **"Elements in row: "** + ***row***;  
}  
**function** *remove*(){  
 **var** remove=***document***.getElementById(**"box2"**).**value**;  
 ***row*** = ***row***.filter(**item** => **item** != remove);  
 ***document***.getElementById(**"input"**).innerHTML = **"Elements in row: "** + ***row***;  
}  
**function** *removex*(){  
 **var** index=***document***.getElementById(**"box3"**).**value**;  
 **if** (index != -1) {  
 ***row***.splice(index, 1);  
 }  
 ***document***.getElementById(**"input"**).innerHTML = **"Elements in row: "** + ***row***;  
}

Output



1. Inheritance

* It contains the value which can inherit another variables which can pass with and retrieve the values with the object name.
* It can be used for the reusability.
* We can inherit the properties with objects, constructor or extendind the values.

**Program Scenario:** Here, when you enter the car company and model. It inherit the values what you can access in the JavaScript this object and inherit for the car details. When you edit the choice and press the button for the edit, it inherit the new values under the same object and display on screen.

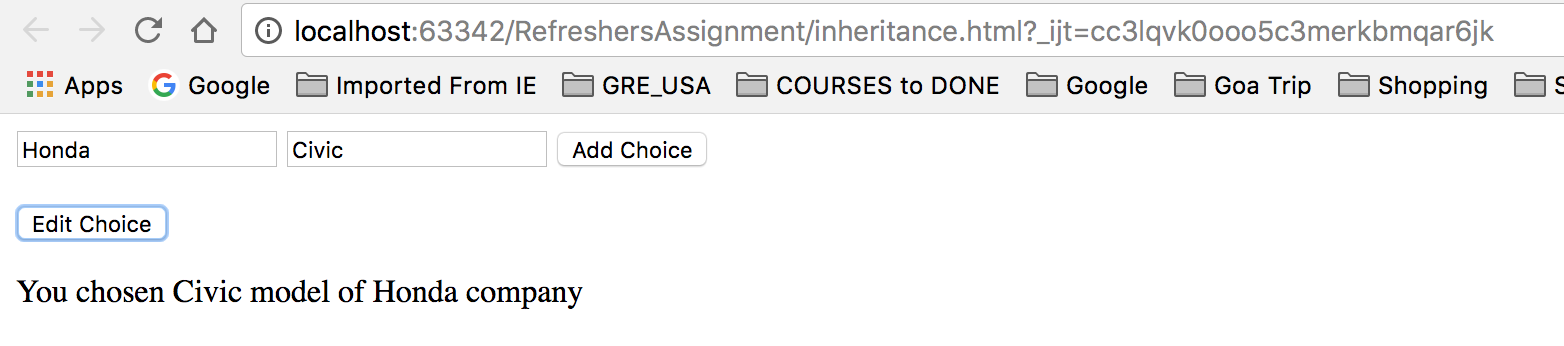
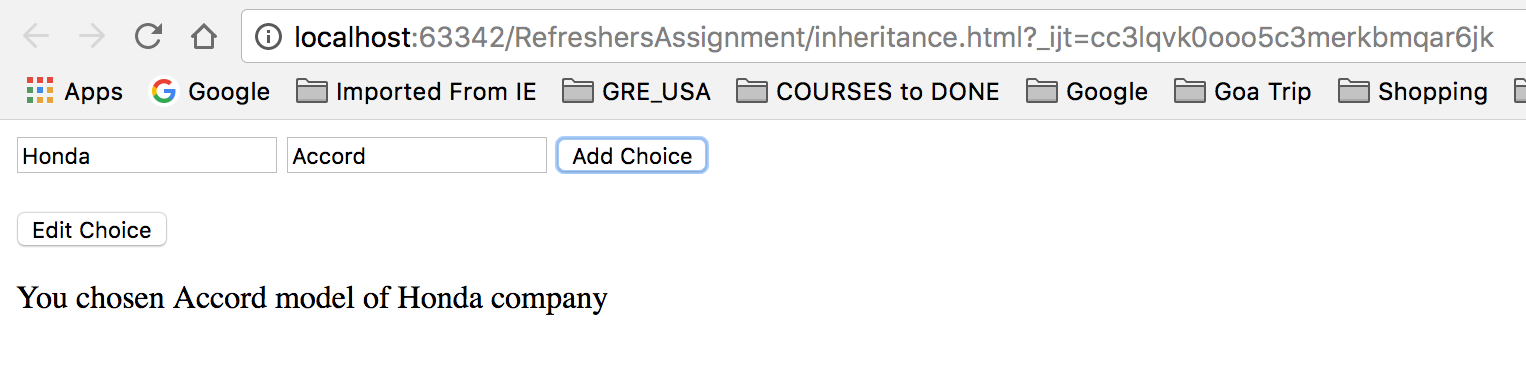
HTML

<!DOCTYPE **html**>  
<**html lang="en"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>Inheritance</**title**>  
 <**script src="inheritance.js"**></**script**>  
</**head**>  
<**body**>  
<**input type="text" id="box1" placeholder="Car Company"**>  
<**input type="text" id="box2" placeholder="Car Model"**>  
<**input type="button" value="Add Choice" onclick="***addchoice*()**"**><**br**><**br**>  
<**input type="button" value="Edit Choice" onclick="***editchoice*()**"**>  
<**p id="input"**></**p**>  
</**body**>  
</**html**>

JavaScript

**function** *addchoice*(){  
 **this**.**company** = ***document***.getElementById(**"box1"**).**value**;  
 **this**.**model**= ***document***.getElementById(**"box2"**).**value**;  
 ***document***.getElementById(**"input"**).innerHTML = **"You chosen "**+ **this**.**model**+ **" model of "**+ **this**.**company**+ **" company"**;  
}  
**function** *editchoice*(){  
 **var** details = **new** *addchoice*();  
 details.**company**= ***document***.getElementById(**"box1"**).**value**;  
 details.**model**= ***document***.getElementById(**"box2"**).**value**;  
 ***document***.getElementById(**"input"**).innerHTML = **"You chosen "**+ details.**model**+ **" model of "**+ details.**company**+ **" company"**;  
}

Output



1. Conditions:

* It defined the particular task you want to perform with checking the rules you define under condition method you implemented.
* You can execute conditions with **if, else if or switch** statement same as Java.

**Program Scenario:** Here, we check the length and width you provide and check whether it is square or rectangle and display it with counting the area or square or rectangle.

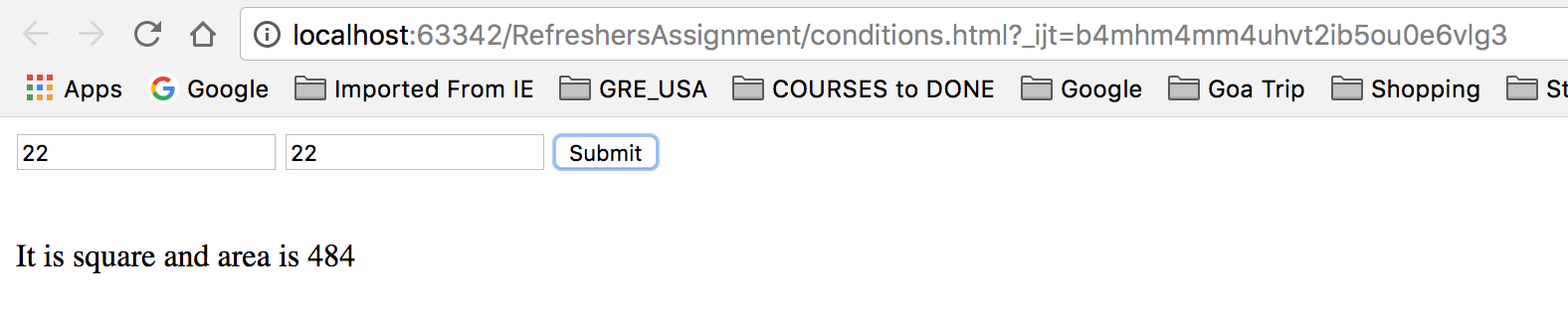
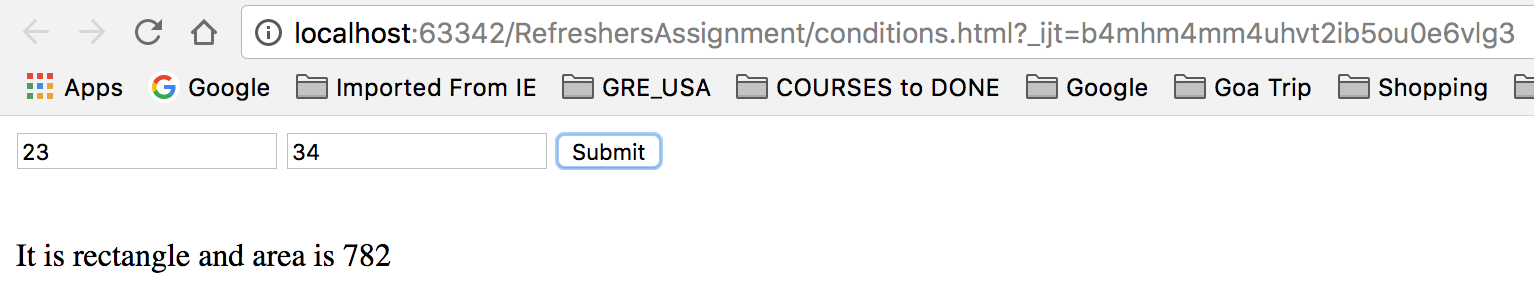
HTML

<!DOCTYPE **html**>  
<**html lang="en"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>Condition Check</**title**>  
 <**script src="conditions.js"**></**script**>  
</**head**>  
<**body**>  
<**input type="number" id="box1" placeholder="Enter Length"**>  
<**input type="number" id="box2" placeholder="Enter Width"**>  
<**input type="button" value="Submit" onclick="***calculate*()**"**><**br**><**br**>  
<**p id="input"**></**p**>  
</**body**>  
</**html**>

JavaScript

**function** *calculate*(){  
 **if**(***document***.getElementById(**"box1"**).**value** == ***document***.getElementById(**"box2"**).**value**){  
 ***document***.getElementById(**"input"**).innerHTML = **"It is square and area is "**+ ***document***.getElementById(**"box1"**).**value** \* ***document***.getElementById(**"box2"**).**value**;  
 }  
 **else  
 *document***.getElementById(**"input"**).innerHTML = **"It is rectangle and area is "**+ ***document***.getElementById(**"box1"**).**value** \* ***document***.getElementById(**"box2"**).**value**;  
}

Output



1. Regular Expression:

* It is the search pattern formed by sequence of characters.
* It searches data in out data we want to check where RegExp can be of one character to multiple characters and can have complicated design which can be used to perform different tasks like searching in text, matching in text and replacing texts.

Program Scenario: Here, in the first box you enter random texts of any length and then if you want to check whether your name is there in that text input or not, you enter your name in the second textbox and press that button. If your name matches the constant text inside the value entered in first box, it will display your name and if it does not find your name, it will display nothing.

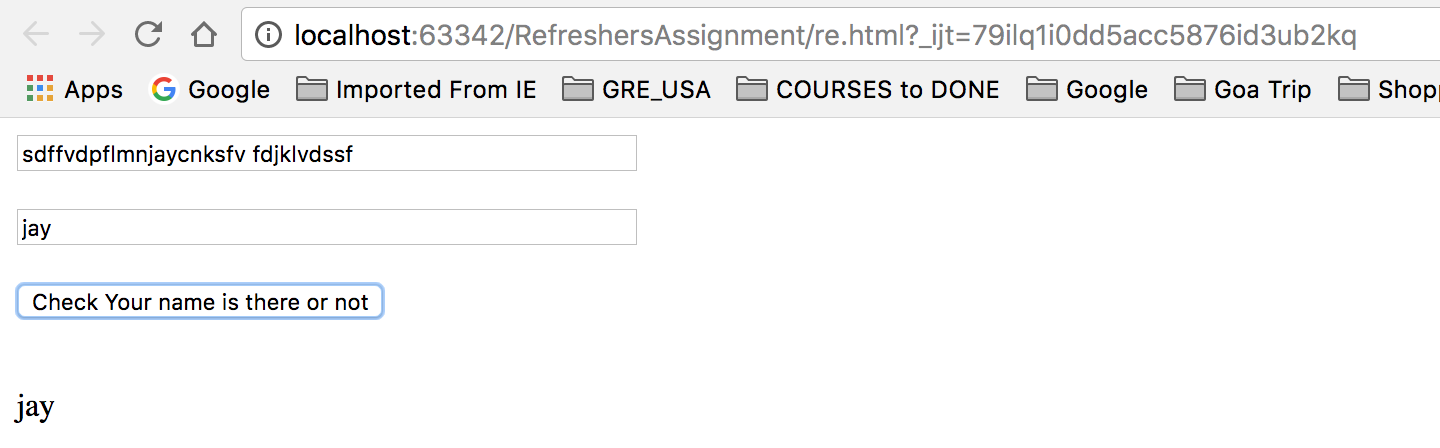
HTML

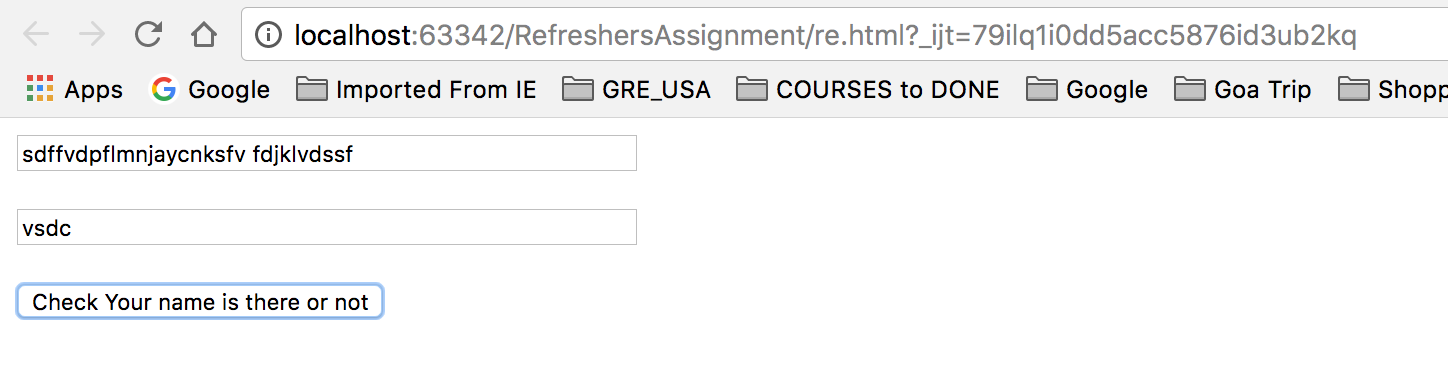
<!DOCTYPE **html**>  
<**html lang="en"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>Find Your Name</**title**>  
 <**script src="re.js"**></**script**>  
</**head**>  
<**body**>  
<**input type="text" size="50" id="box1" placeholder="Enter any random text"**><**br**><**br**>  
<**input type="text" size="50" id="box2" placeholder="Enter your name"**><**br**><**br**>  
<**input type="button" value="Check Your name is there or not" onclick="***re*()**"**><**br**><**br**>  
<**p id="input"**></**p**>  
</**body**>  
</**html**>

JavaScript

**function** *re*(){  
 **debugger  
 var** name= **new *RegExp***(***document***.getElementById(**"box2"**).**value**,**"gi"**);  
 **var** text= ***document***.getElementById(**"box1"**).**value**;  
 **var** matching = text.match(name);  
 ***document***.getElementById(**"input"**).innerHTML = matching;}

Output





1. Strict Mode:

* Purpose of strict mode is to check whether the code defined under strict mode condition written as per the JavaScript standards or not.
* It creates the error if the code is not as per the JavaScript standards and will not let the code get executed.

**Program Scenario:** Here, we enter two values in the box and press the button for adding them. One button is for non-strict mode and another is for strict mode. If you press the non-strict mode, it executes the JavaScript function without strict mode expression and display the value. And if you press the button for strict mode value, it executes strict mode and will not perform add function even if code written into the function is the same and create an error in the console.

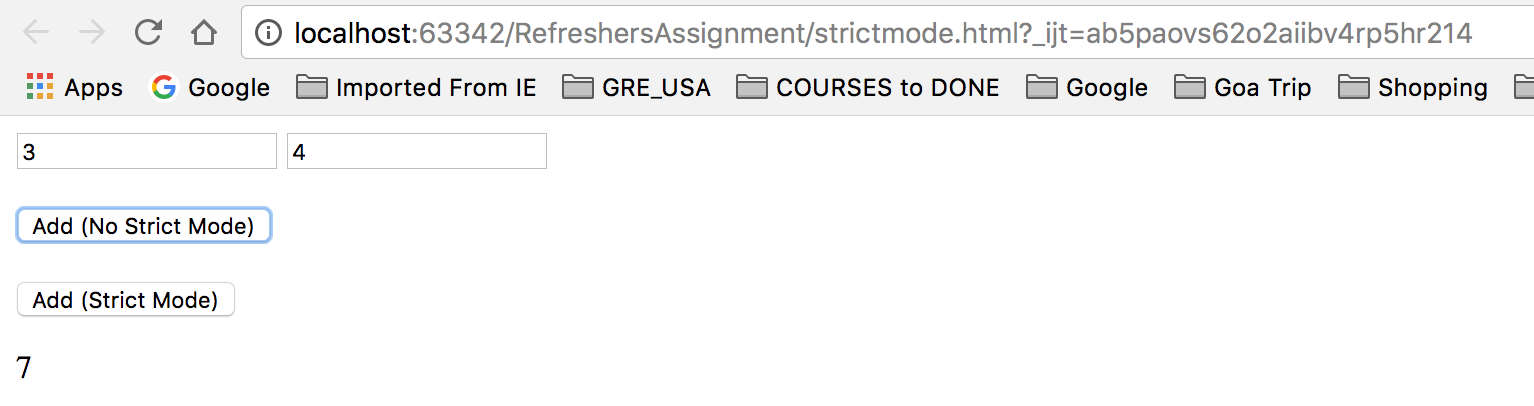
HTML

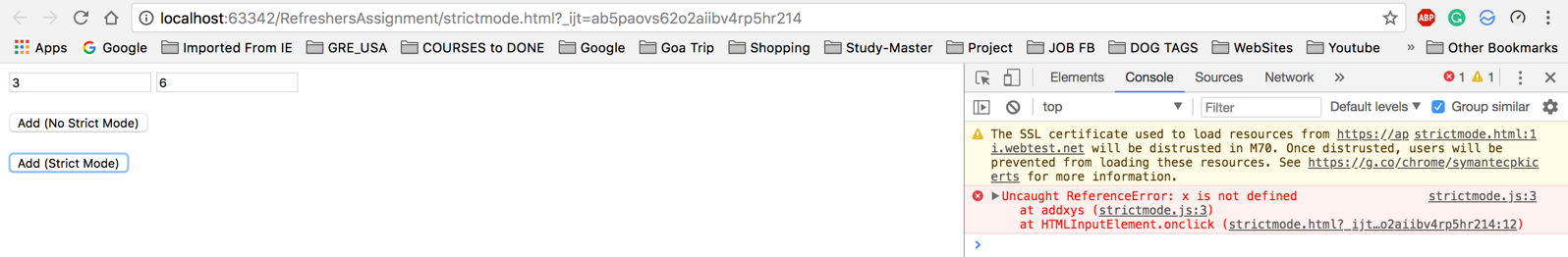
<!DOCTYPE **html**>  
<**html lang="en"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>Strict Mode Data Deletion</**title**>  
 <**script src="strictmode.js"**></**script**>  
</**head**>  
<**body**>  
 <**input type="int" id="box1" placeholder="Enter Value for X: "**>  
 <**input type="int" id="box2" placeholder="Enter Value for Y: "**><**br**><**br**>  
 <**input type="button" value="Add (No Strict Mode)" onclick="***addxy*()**"**><**br**><**br**>  
 <**input type="button" value="Add (Strict Mode)" onclick="***addxys*()**"**>  
 <**p id="input"**></**p**>  
</**body**>  
</**html**>

JavaScript

**function** *addxys*(){  
 **"use strict"**;  
 x = *parseInt*(***document***.getElementById(**"box1"**).**value**) + *parseInt*(***document***.getElementById(**"box2"**).**value**);  
 ***document***.getElementById(**"input"**).innerHTML = x;  
}  
**function** *addxy*(){  
 **x** = *parseInt*(***document***.getElementById(**"box1"**).**value**) + *parseInt*(***document***.getElementById(**"box2"**).**value**);  
 ***document***.getElementById(**"input"**).innerHTML = **x**;  
}

Output





10. Errors:

* JavaScript raises errors something is wrong with the scripting or some method is not defined.
* It works with the **Try{}, Catch(){}, throw and finally{}.**
* When code inside the try block raise the error, it will get catched by the catch and task will be perform. With throw, we can throw the exception is catch it explicitly.
* **Finally**  block will get executed everytime.

Program Scenario: Here, we enter two values and press the add button. In JavaScript file, when it calls the add function, error will be raised as add function is not mentioned in JavaScript.

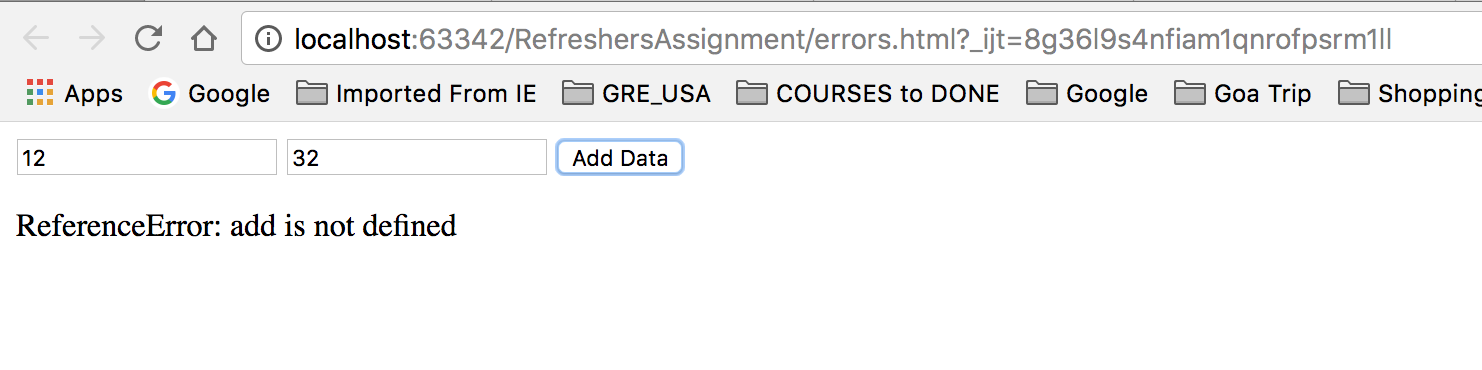
HTML

<!DOCTYPE **html**>  
<**html lang="en"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>Find if something is wrong</**title**>  
 <**script src="errors.js"**></**script**>  
</**head**>  
<**body**>  
 <**input type="number" id="box1" placeholder="Enter first number"**>  
 <**input type="number" id="box2" placeholder="Enter second number"**>  
 <**input type="button" value="Add Data" onclick="***retrivedata*()**"**>  
 <**p id="inputerror"**></**p**>  
</**body**>  
</**html**>

JavaScript

**function** *retrivedata*(){  
 **try**{  
 **var** x = ***document***.getElementById(**"box1"**).**value**;  
 **var** y = ***document***.getElementById(**"box2"**).**value**;  
 **if**(x == **""** || y == **""**)  
 **throw "Values cannot be NULL"**;  
 **else** {  
 **var** x = *parseInt*(***document***.getElementById(**"box1"**).**value**) + *parseInt*(***document***.getElementById(**"box2"**).**value**);  
 add(x);  
 }  
 }  
 **catch**(err) {  
 ***document***.getElementById(**"inputerror"**).innerHTML = err;  
 }  
}

Output



1. Type Conversion:

* Different datatypes like number, string, Boolean, object and function.
* We can type cast datatypes into each other by storing the values and it is called, Type conversion to another datatype.
* We can find the data type of the variable with typeof() which returns the string and have data type of the variable.
* JavaScript also works for auto type conversion when you try to print one data type to another format. (e.g. String(25+10))

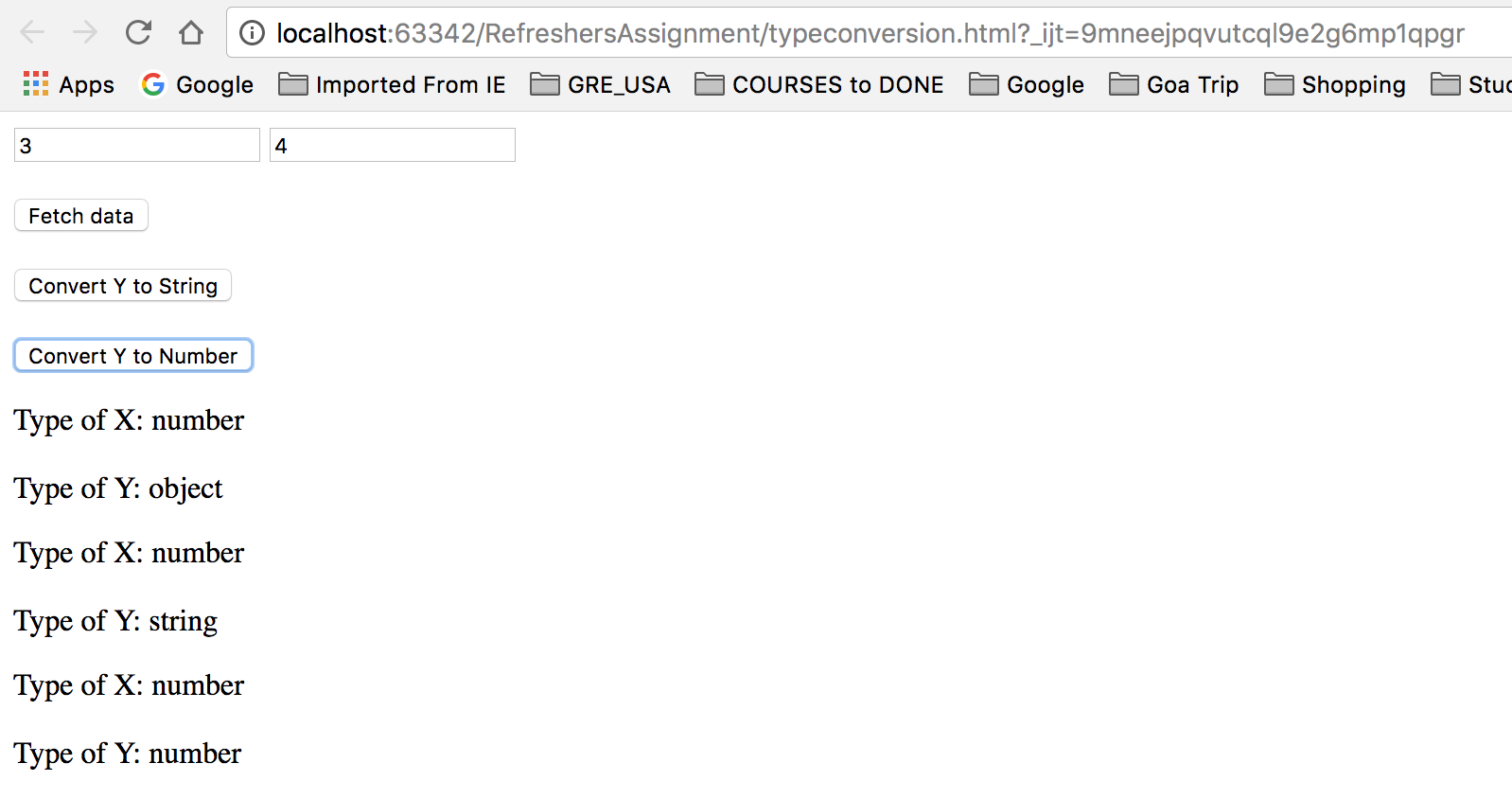
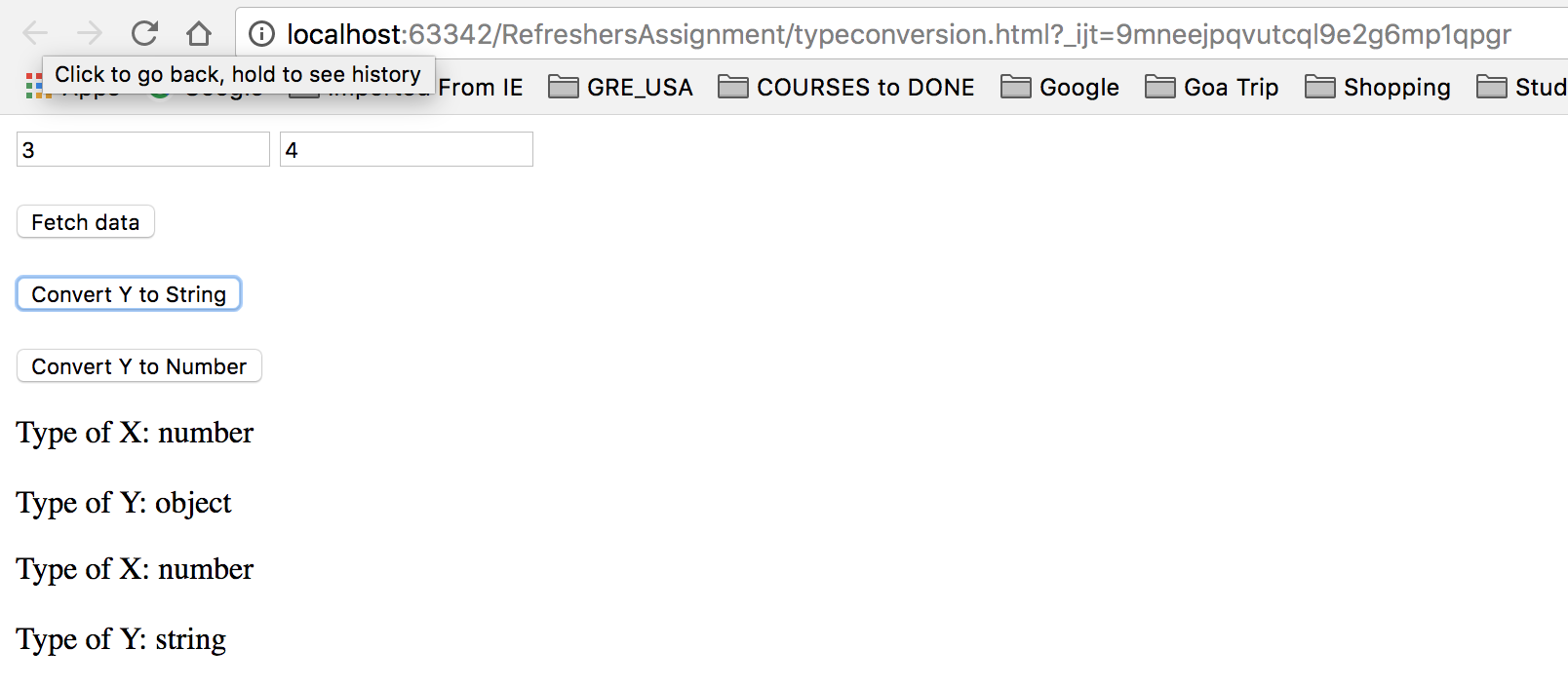
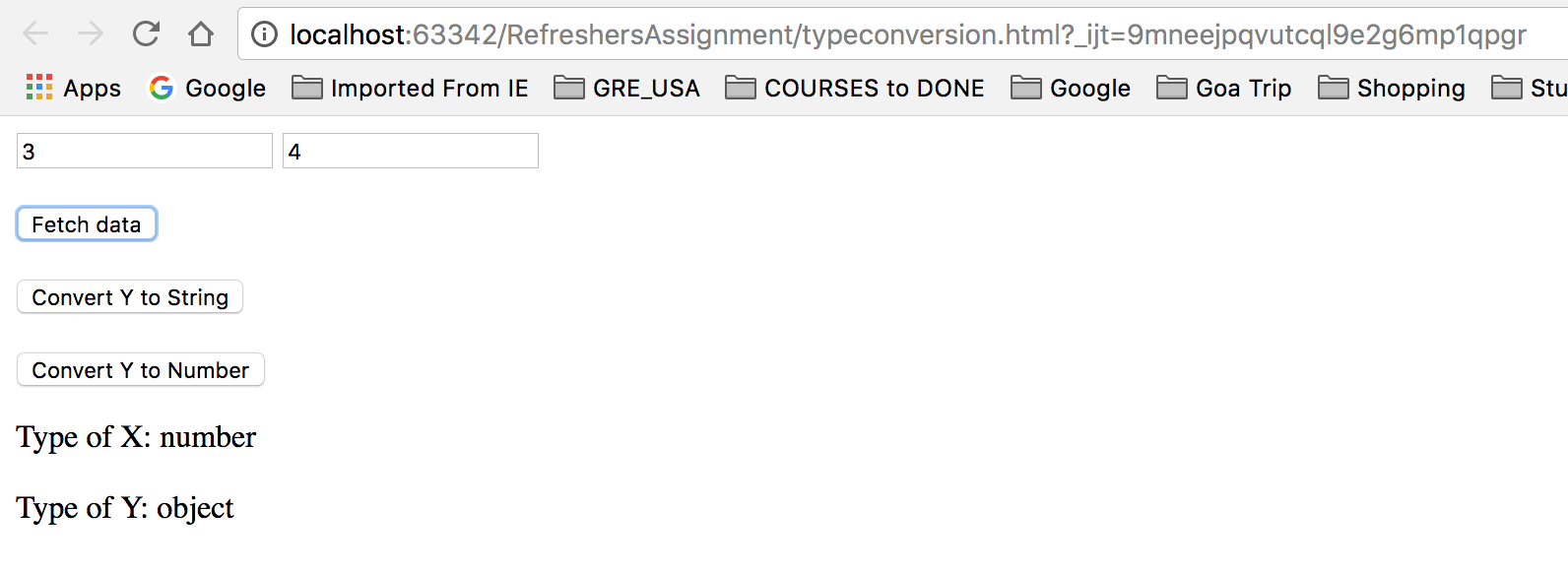
Program Scenario: Here, when you enter the data first and press fetch data button, It executes the function fetchdetails and gives the information of the data. Where y is of object type. When you press convert y to string button, in convertstring function, it will convert the type of y to string from object and call the same fetchdetails function and retrieve the type of y as string. Again when you press convertnumber button, it will convert the type of y to number and display that type of y is number.

HTML

<!DOCTYPE **html**>  
<**html lang="en"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>Convert DataType</**title**>  
 <**script src="typeconversion.js"**></**script**>  
</**head**>  
<**body**>  
 <**input type="number" id="box1" placeholder="Enter Number"**>  
 <**input type="text" id="box1" placeholder="Second Number"**><**br**><**br**>  
 <**input type="button" value="Fetch data" onclick="***fetchdetails*()**"**><**br**><**br**>  
 <**input type="button" value="Convert Y to String" onclick="***convertstring*()**"**><**br**><**br**>  
 <**input type="button" value="Convert Y to Number" onclick="***convertnumber*()**"**>  
 <**p id="input"**></**p**>  
 <**p id="input1"**></**p**>  
 <**p id="input2"**></**p**>  
</**body**>  
</**html**>

JavaScript

**function** *fetchdetails*(){  
 **var** x = *parseInt*(***document***.getElementById(**"box1"**).**value**);  
 **var** y = ***document***.getElementById(**"box2"**);  
 ***document***.getElementById(**"input"**).innerHTML = **"Type of X: "** + **typeof** x + **"<br><br>"** + **"Type of Y: "** + **typeof** y;  
}  
**function** *convertnumber*(){  
 **var** x = *parseInt*(***document***.getElementById(**"box1"**).**value**);  
 **var** y = *parseInt*(***document***.getElementById(**"box2"**));  
 ***document***.getElementById(**"input2"**).innerHTML = **"Type of X: "** + **typeof** x + **"<br><br>"** + **"Type of Y: "** + **typeof** y;  
}  
**function** *convertstring*(){  
 **var** x = *parseInt*(***document***.getElementById(**"box1"**).**value**);  
 **var** y = *toString*(***document***.getElementById(**"box2"**));  
 ***document***.getElementById(**"input1"**).innerHTML = **"Type of X: "** + **typeof** x + **"<br><br>"** + **"Type of Y: "** + **typeof** y;  
}

Output

1. JSON:

* JSON Stands for **J**ava**S**cript **O**bject **N**otation.
* It is language independent and self-describing which is easy to understand for the user.
* JSON format is same as JavaScript format which is easily convertible.
* JSON data is **name value** pairs

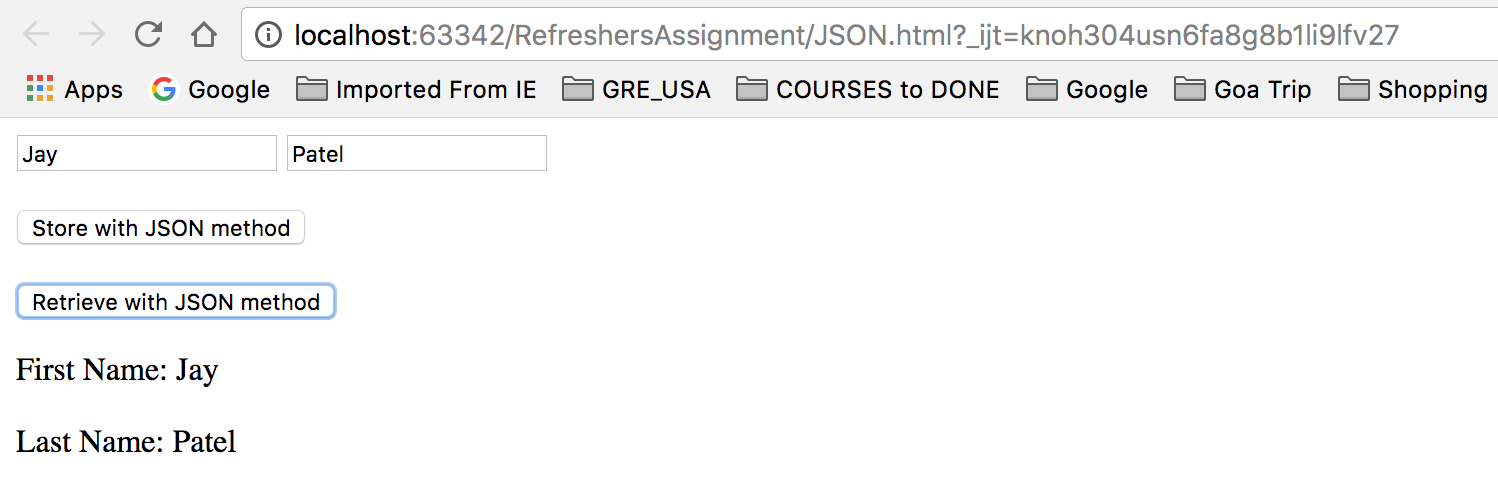
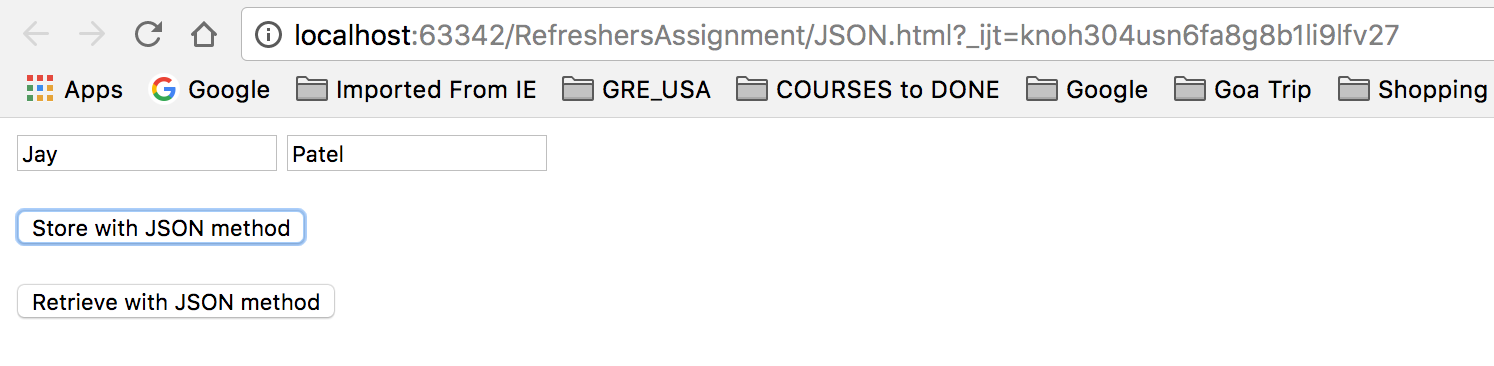
Program Scenario: Here, we take input and storing the data with JSON object. When we press the Retrieve with JSON method button, it fetch the details from the JSON method and displays on the screen.

HTML

<!DOCTYPE **html**>  
<**html lang="en"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>JSON Data storing and retriving</**title**>  
 <**script src="JSON.js"**></**script**>  
</**head**>  
<**body**>  
 <**input type="text" id="box1" placeholder="First Name"**>  
 <**input type="text" id="box2" placeholder="Last Name"**><**br**><**br**>  
 <**input type="button" onclick="***storeJSON*()**" value="Store with JSON method"**><**br**><**br**>  
 <**input type="button" onclick="***retriveJSON*()**" value="Retrieve with JSON method"**>  
 <**p id="input"**></**p**>  
</**body**>  
</**html**>

JavaScript

**function** *storeJSON*(){  
 **myJSON** = { **"fname"**:***document***.getElementById(**"box1"**).**value**, **"lname"**:***document***.getElementById(**"box2"**).**value** };  
 **myJSONdata** = ***JSON***.stringify(**myJSON**);  
 ***localStorage***.setItem(**"JSONdata"**, **myJSONdata**);  
}  
**function** *retriveJSON*(){  
 **data** = ***localStorage***.getItem(**"JSONdata"**);  
 **JSONobj** = ***JSON***.parse(**data**);  
 ***document***.getElementById(**"input"**).innerHTML = **"First Name: "** + **JSONobj**.**fname** + **"<br><br>Last Name: "** + **JSONobj**.**lname**;  
}

Output

1. HTML5
2. Local Storage:

* Local storage is used to retrieve user’s data even if he navigates from one page to another
* Earlier local storage was used through cookies
* Web storage is per origin like protocol.
* All pages, from one origin, can store and access the same data.

**Program Scenario**: A text box is given, and an input is taken from the user. The page is then redirected, and the value of the text box is retained.

HTML File 1

<!DOCTYPE **html**>  
<**html lang="en"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>Local Storage</**title**>  
 <**script src="localstorage.js"**></**script**>  
</**head**>  
<**body**>  
 <**input type="text" id="box1" placeholder="Retrive on next page"**>  
 <**input type="button" onclick="***retrive*()**" value="Retrive on Next Page"**>  
</**body**>  
</**html**>

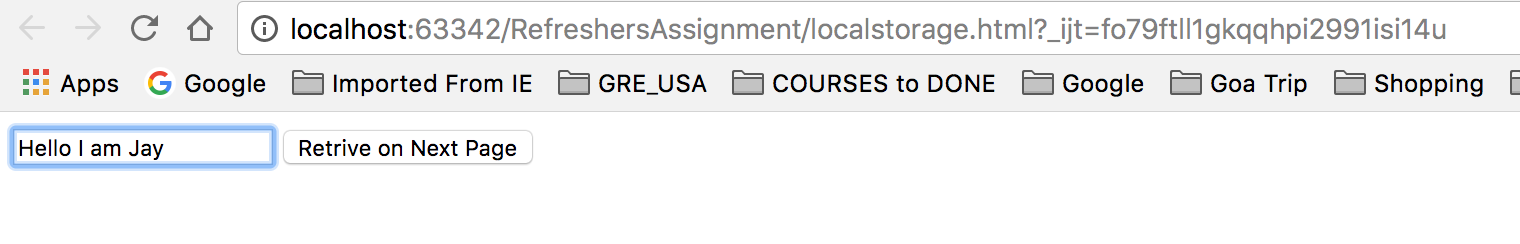
HTML File 2

<!DOCTYPE **html**>  
<**html lang="en"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>Retrived Data</**title**>  
 <**script src="localstorage.js"**></**script**>  
</**head**>  
<**body onload="***retrivevalue*()**"**>  
<**p id="input"**></**p**>  
</**body**>  
</**html**>

JavaScript

**function** *retrive*(){  
 ***localStorage***.setItem(**"retrive"**, ***document***.getElementById(**"box1"**).**value**);  
 ***window***.location.assign(**"localstorage2.html"**);  
}  
**function** *retrivevalue*(){  
 ***document***.getElementById(**"input"**).innerHTML=***localStorage***.getItem(**"retrive"**);  
}

Output:



1. Media (Audio and Video):

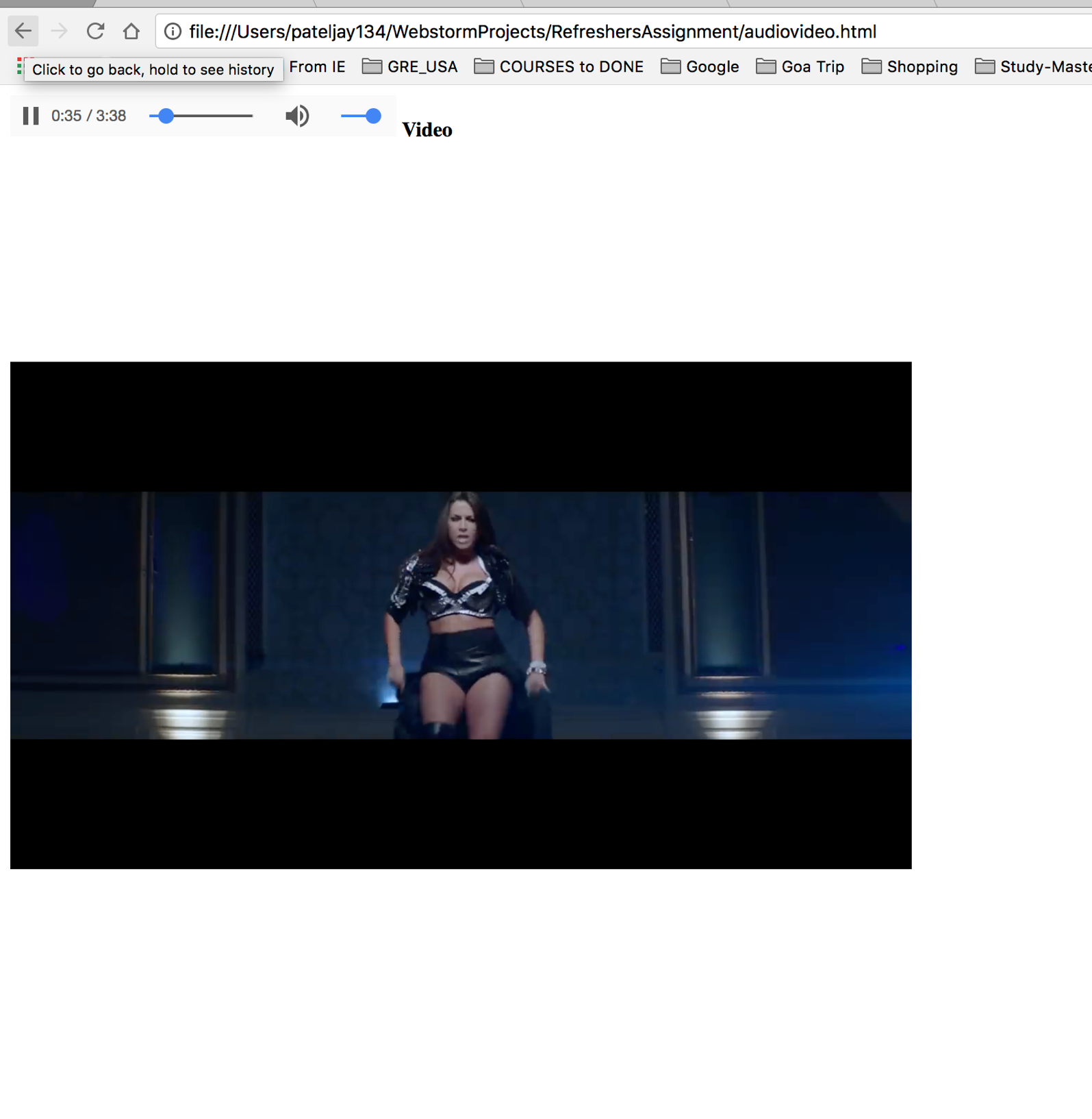
* Video and audio tags are used to insert audios and videos in HTML page
* Before HTML 5, these were supposed to be played on plugins like Flash Player
* Video Tags help to make the HTML page more attractive

**Program Scenario**: Different Audio and Video tags are inserted in HTML page to show which are either downloaded or fetched from third party.

HTML File 1

<!DOCTYPE **html**>  
<**html lang="en"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>Audio and Video Files</**title**>  
</**head**>  
<**body**>  
<**audio controls**>  
 <**source src="/Users/pateljay134/Downloads/audio.mp3" type="audio/mp3"**>  
</**audio**>  
<**b**>Video</**b**><**br**><**br**>  
<**video width="700" height="700" controls**>  
 <**source src="/Users/pateljay134/Downloads/video.mp4" type="video/mp4"**>  
 Your browser does not support the video tag.  
</**video**>  
</**body**>  
</**html**>

Output:



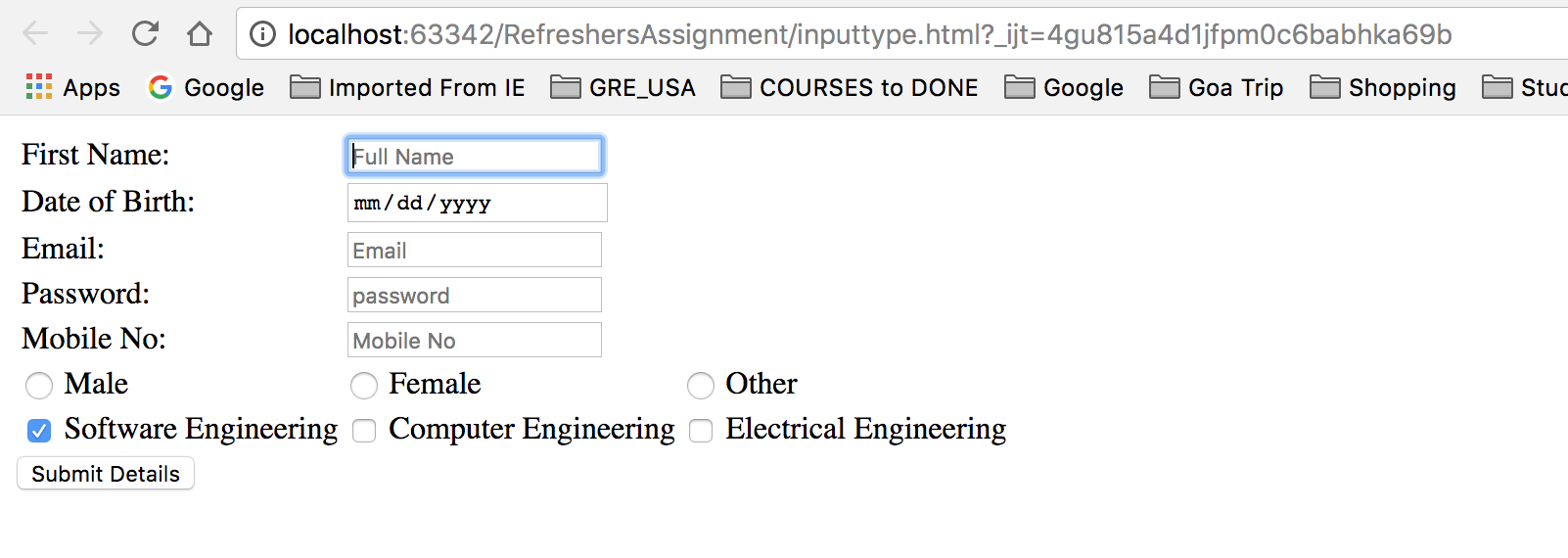
1. Input Type:

* Input tags are used to take inputs from a user and give it to the associated page
* There are different types of input like text, number, range etc
* Input tags are used in forms to fetch details given by a users
* **Program Scenario**: Created a table and have given multiple input types and tested different properties.

HTML File 1

<!DOCTYPE **html**>  
<**html lang="en"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>Input Type Examples</**title**>  
 <**script src="inputtype.js"**></**script**>  
</**head**>  
<**body**>  
 <**table**>  
 <**tr**>  
 <**td**>First Name: </**td**>  
 <**td**><**input type="text" id="type1" placeholder="Full Name"**></**td**>  
 </**tr**>  
 <**tr**><**td**>Date of Birth:</**td**>  
 <**td**><**input type="date" id="type2"**></**td**>  
 </**tr**>  
 <**tr**>  
 <**td**>Email: </**td**>  
 <**td**><**input type="email" id="type3" placeholder="Email"**></**td**>  
 </**tr**>  
 <**tr**>  
 <**td**>Password: </**td**>  
 <**td**><**input type="password" id="type4" placeholder="password"**></**td**>  
 </**tr**>  
 <**tr**>  
 <**td**>Mobile No: </**td**>  
 <**td**><**input type="tel" max="10" id="type5" placeholder="Mobile No"**></**td**>  
 </**tr**>  
 <**tr**><**td**><**input type="radio" name="gender" value="male"**> Male</**td**>  
 <**td**><**input type="radio" name="gender" value="female"**> Female</**td**>  
 <**td**><**input type="radio" name="gender" value="other"**> Other</**td**>  
 </**tr**>  
 <**tr**>  
 <**td**><**input type="checkbox" name="se" value="se" checked**> Software Engineering</**td**>  
 <**td**><**input type="checkbox" name="ce" value="ce"**> Computer Engineering</**td**>  
 <**td**><**input type="checkbox" name="ee" value="ee"**> Electrical Engineering</**td**>  
 </**tr**>  
 </**table**>  
 <**input type="button" value="Submit Details" onclick="***displaydata*()**"**>  
 <**p id="input"**></**p**>  
</**body**>  
</**html**>

Output:



1. Geolocation:

* Geolocation is used to get user’s current position
* It works with longitudes, latitudes and radians
* It can also be used to find distance between two points
* **Program Scenario**: Get current location

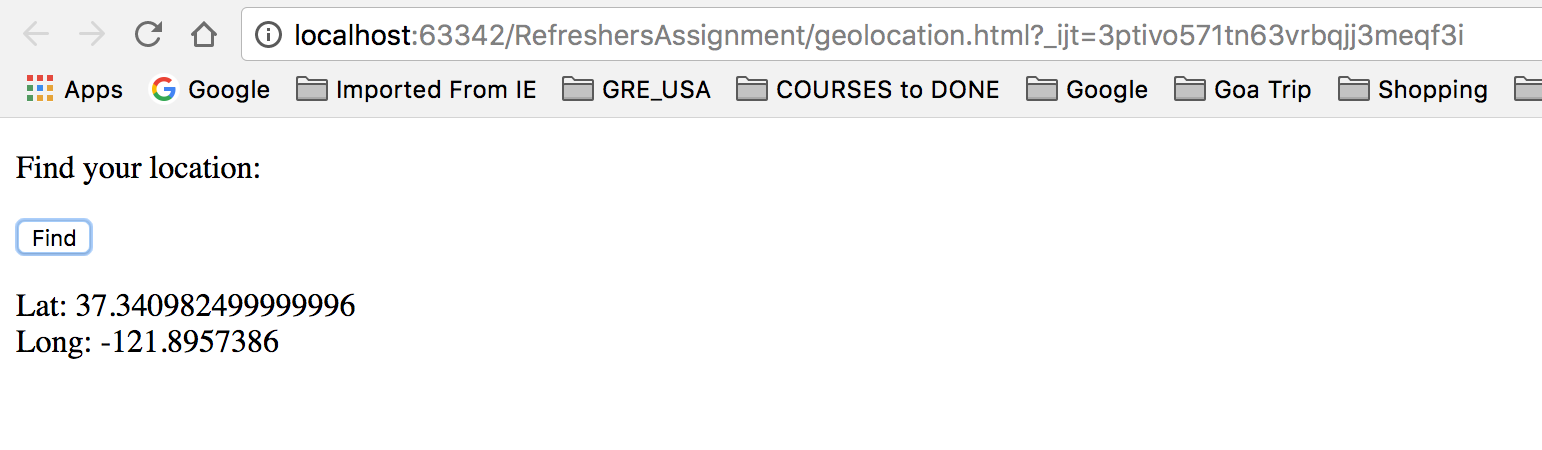
HTML

<!DOCTYPE **html**>  
<**html**>  
<**head**>  
 <**script src="geolocation.js"**></**script**>  
</**head**>  
<**body**>  
  
<**p**>Find your location:</**p**>  
  
<**button onclick="***findLocation*()**"**>Find</**button**>  
  
<**p id="input"**></**p**>  
  
  
</**body**>  
</**html**>

JavaScript

**function** *findLocation*() {  
 **if** (***navigator***.geolocation) {  
 ***navigator***.geolocation.getCurrentPosition(*showPosition*);  
 } **else** {  
 ***document***.getElementById(**"input"**).innerHTML = **"Geolocation is not supported on this browser"**;  
 }  
}  
**function** *showPosition*(position) {  
 ***document***.getElementById(**"input"**).innerHTML = **"Lat: "** + position.coords.latitude +  
 **"<br>Long: "** + position.coords.longitude;  
}

Output:



1. JAVA
2. Queues:

* Queues are special type of data structures which inserts the element at the beginning of queue and pops.
* Java already has function to add, delete function to add data directly to the queue.
* It works as first in first out method.
* Data inserted first will be fetched first from the queue.

**Program Scenario**: Here, we create queue and ask for the new data to insert into the queue. It prioritize the data and gives output the first data with more priority. Here, priority means ahead alphabetical order.

Java Code:

import java.util.\*;

public class QueueFile{

static Queue create(Queue ticketline) {

ticketline.add("Jay");

ticketline.add("Murtaza");

ticketline.add("Deepak");

ticketline.add("Vajid");

ticketline.add("Deepak");

ticketline.add("Mustafa");

ticketline.add("Madhukar");

return ticketline;

}

static Queue add(Queue ticketline, String name) {

ticketline.add(name);

return ticketline;

}

static Object prioritynameinqueue(Queue ticketline) {

return ticketline.element();

};

public static void main(String args[]) {

Queue<String> ticketline = new PriorityQueue<String>();

ticketline = create(ticketline);

System.out.println("Queue: " + ticketline);

System.out.println("Enter any name you want to add to the queue: ");

ticketline = add(ticketline, new Scanner(System.in).nextLine());

System.out.println("\nNew Queue: " + ticketline);

System.out.println("First prioritized person in queue: "+ prioritynameinqueue(ticketline));

}

}

**JUnit Test Code:**

import static org.junit.Assert.\*;

import java.util.PriorityQueue;

import java.util.Queue;

import java.util.Scanner;

import org.junit.Test;

public class QueueFileTest {

@Test

public void testMain() {

Queue<String> ticketline = new PriorityQueue<String>();

ticketline = QueueFile.create(ticketline);

System.out.println("Queue: " + ticketline);

ticketline = QueueFile.add(ticketline, "Vajid");

System.out.println("\nNew Queue: " + ticketline);

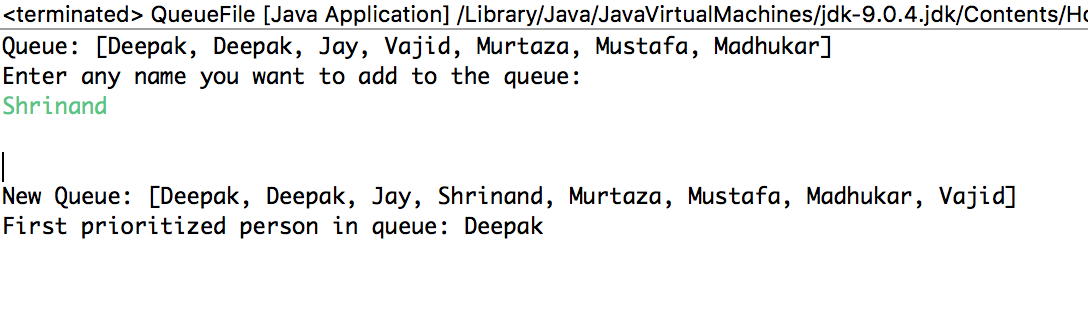
System.out.println("First prioritized person in queue: "+ QueueFile.prioritynameinqueue(ticketline));

String output= (String) QueueFile.prioritynameinqueue(ticketline);

assertEquals(true, output.equals("Deepak"));

}

}

Output:

1. Stack:

* Stack follows last in first out technique.
* An element is inserted at the top at the stack and element is popped out from the beginning of the stack.

**Program Scenario**: We insert the data into the stack and ask to search data if it is in the stack or not. And gives the output about the data is there or not. It also gives output the first element in the stack.

Java Code:

import java.util.\*;

class Stacks{

static Stack add(Stack total) {

total.push(22);

total.push(12);

total.push(34);

total.push(65);

total.push(2);

return total;

}

static void delete(Stack total) {

for (int i = 0; i<total.size(); i++)

total.pop();

System.out.println("Your team is deleted.");

}

static int peek(Stack total){

Integer element = (Integer)total.peek();

return element;

}

static boolean search(Stack stack){

System.out.println("Enter the element you want to search:");

Scanner s = new Scanner(System.in);

int search = s.nextInt();

Integer stackpos = (Integer) stack.search(search);

if(stackpos == -1) {

System.out.println("Not Found");

return false;

}

else

return true;

}

public static void main (String[] args){

Stack total = new Stack();

total = add(total);

int peek = peek(total);

boolean searched = search(total);

if(searched==true)

System.out.println("Found");

}

}

JUnit Test Code:

import static org.junit.Assert.assertEquals;

import java.util.Stack;

import org.junit.Test;

public class StackTest {

@Test

public void test() {

Stack total = new Stack();

total = Stacks.add(total);

int peek = Stacks.peek(total);

boolean searched = Stacks.search(total, 10);

assertEquals(false, searched);

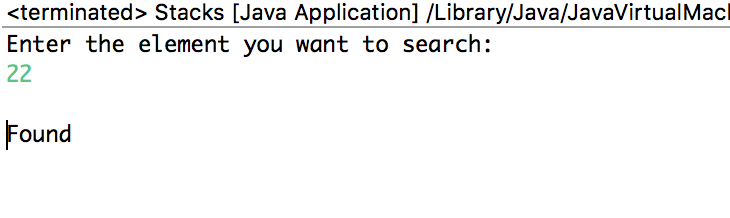
searched = Stacks.search(total, 22);

assertEquals(true, searched);

}

}

Output:



1. Arrays:

* It can store multiple values with one variable name.
* You can refer every items by referencing with index number.

**Program Scenario**: Here, we initialize one array and gives output after reversing the elements of the array.

Java Code:

import java.util.\*;

class ArrayFile{

static int[] returnArray(int[] enrollment) {

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

System.out.println(enrollment[i]);

System.out.println("Reversed Array: ");

for(int i=0;i<enrollment.length/2;i++) {

int temp = enrollment[i];

enrollment[i]=enrollment[enrollment.length-i-1];

enrollment[enrollment.length-i-1]=temp;

}

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

System.out.println(enrollment[i]);

return enrollment;

}

public static void main(String args[]) {

int[] enrollment = {10,22,45,8};

System.out.println("Enrolled Numbers: ");

returnArray(enrollment);

System.out.println("Your class is ready to begin now");

}

}

**JUnit Test Code:**

import static org.junit.Assert.\*;

import org.junit.Test;

public class ArrayFileTest {

@Test

public void testReturnArray() {

int[] enrollment = { 10, 22, 45, 8 };

System.out.println("Enrolled Numbers: ");

int[] finalArray = ArrayFile.returnArray(enrollment);

//Checking 1st Element of Array after reversing

assertEquals(8, finalArray[0]);

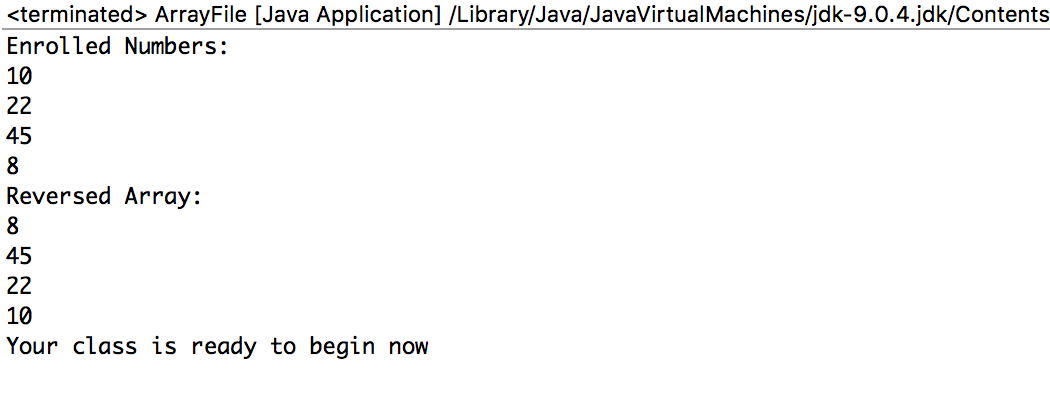
//Checking 1st Element of Array after reversing

assertEquals(45, finalArray[1]);

}

}

Output:



1. Interfaces:

* Interface are used to overcome the problem of multiple inheritance in java.
* Interface can only have abstract methods which can only be declared, not defined.

**Program Scenario**: Here, we are inserting the data for two pairs of points and calculating the values of difference between the pair of points. Then, we compare which pair of points have more distance, less distance or equal distance.

Java Code:

public class Interface implements Relation {

public static void main(String args[]) {

System.out.print("We are comparing the distance between the two pair of points\n\n");

Interface i1 = new Interface(0,0,1,1);

Interface i2 = new Interface(1,2,8,5);

if(i1.isGreater(i1, i2)==true)

System.out.println("First point of points have more distance than second pair\n");

if(i1.isLess(i1, i2)==true)

System.out.println("Second point of points have more distance than first pair\n");

if(i1.isEqual(i1, i2)==true)

System.out.println("Distance between the two pairs of points are equal\n");

}

private double x1;

private double x2;

private double y1;

private double y2;

public Interface(double x1, double x2, double y1, double y2){

this.x1 = x1;

this.x2 = x2;

this.y1 = y1;

this.y2 = y2;

}

public double getLength(){

double length = Math.sqrt((x2-x1)\*(x2-x1) + (y2-y1)\* (y2-y1));

return length;

}

public boolean isGreater( Object a, Object b){

double aLen = ((Interface)a).getLength();

double bLen = ((Interface)b).getLength();

return (aLen > bLen);

}

public boolean isLess( Object a, Object b){

double aLen = ((Interface)a).getLength();

double bLen = ((Interface)b).getLength();

return (aLen < bLen);

}

public boolean isEqual( Object a, Object b){

double aLen = ((Interface)a).getLength();

double bLen = ((Interface)b).getLength();

return (aLen == bLen);

}

}

**Relation:**

public interface Relation {

public boolean isGreater( Object a, Object b);

public boolean isLess( Object a, Object b);

public boolean isEqual( Object a, Object b);

}

**JUnit Test Code:**

import static org.junit.Assert.\*;

import org.junit.Test;

public class InterfaceTest {

@Test

public void testMain() {

Interface i1 = new Interface(0, 0, 1, 1);

Interface i2 = new Interface(1, 2, 8, 5);

if (i1.isGreater(i1, i2) == true)

System.out

.println("First point of points have more distance than second pair\n");

if (i1.isLess(i1, i2) == true)

System.out

.println("Second point of points have more distance than first pair\n");

if (i1.isEqual(i1, i2) == true)

System.out

.println("Distance between the two pairs of points are equal\n");

assertEquals(false, i1.isGreater(i1, i2));

assertEquals(true, i1.isLess(i1, i2));

assertEquals(false, i1.isEqual(i1, i2));

}

}

Output:



1. Collections:

* Collection is used to encapsulate multiple objects together.
* Collection was introduced because we had to give specific of an array however in collection, we can increase the size dynamically.

**Program Scenario**: Here, we are creating the hash table with different double values storing with different names. Then we call to update the hash table by giving input to add extra balance to some of the hash table name.

Java Code:

import java.util.\*;

class Collection

{

static Hashtable createhashtable(Hashtable balance) {

String str;

balance.put("Jay", new Double(3434.34));

balance.put("Murtaza", new Double(-19.08));

balance.put("Deepak", new Double(1378));

balance.put("Dhruval", new Double(123.22));

balance.put("Madhukar", new Double(99.22));

return balance;

}

static Hashtable updatehashtable(Hashtable balance,double amount) {

Set set=balance.keySet();

Iterator itr=set.iterator();

while(itr.hasNext())

{

String str=(String) itr.next();

System.out.println(str +" : " +balance.get(str));

}

String name = new Scanner(System.in).nextLine();

double bal=((Double)balance.get(name)).doubleValue();

balance.put(name, bal+amount);

System.out.println(name+"'s new balance:"+balance.get(name));

return balance;

}

public static void main(String args[])

{

Hashtable balance= new Hashtable();

createhashtable(balance);

balance = updatehashtable(balance,5500);

Set set=balance.keySet();

Iterator itr=set.iterator();

while(itr.hasNext())

{

String str=(String) itr.next();

System.out.println(str +" : " +balance.get(str));

}

}

}

**JUnit Test Code:**

import static org.junit.Assert.\*;

import java.util.Hashtable;

import java.util.Iterator;

import java.util.Set;

import org.junit.Test;

public class CollectionTest {

@Test

public void testUpdatehashtable() {

Hashtable balance = new Hashtable();

Collection.createhashtable(balance);

Hashtable finalBalance = new Hashtable();

finalBalance = Collection.updatehashtable(balance, 5500);

Set set = balance.keySet();

Iterator itr = set.iterator();

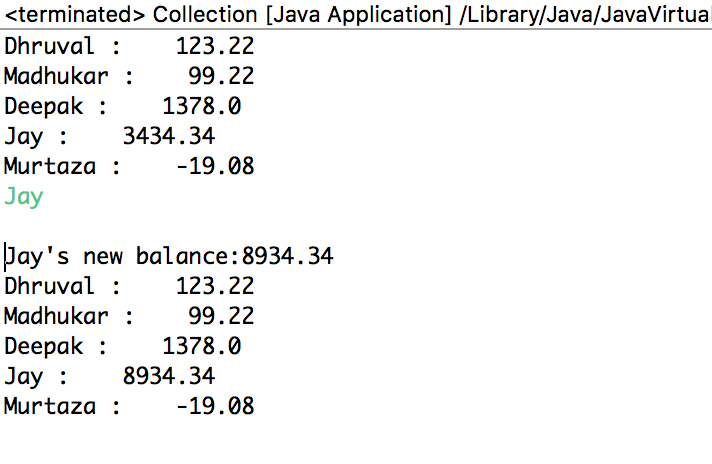
String str = (String) itr.next();

assertEquals(123.22, (double)balance.get(str));

}

}

Output:



1. Generics:

* Generics are used to contain different types of datatypes object together
* Generics also provides compile time safety that helps developer to catch invalid datatypes

**Program Scenario**: Here we are creating two arrays with integer and double values and passing to store data. Then we call for any value of array data match with the other data and return the true false value.

Java Code:

import java.util.\*;

public class Generic{

static < T > void printArray( T[] inputArray ){

for ( T element : inputArray ){

System.out.printf( "%s ",element);

}

}

static < T > boolean checkmatch( T[] inputArray ){

boolean match=false;

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

for(int j=i+1;j<inputArray.length;j++) {

if(inputArray[i]==inputArray[j])

match=true;

}

}

return match;

}

public static void main(String args[]){

Integer[] intArray = { 5,3,6,2,5 };

Double[] doubleArray = { 2.2,6.6,1.1};

System.out.println( "Array intArray contains:" );

printArray(intArray);

boolean match=false;

match = checkmatch(intArray );

if(match==true)

System.out.println( "\nYou have two common values" );

System.out.println( "\nArray doubleArray contains:" );

printArray( doubleArray );

}

}

**JUnit Test Code:**

import static org.junit.Assert.assertEquals;

import static org.junit.jupiter.api.Assertions.\*;

import org.junit.jupiter.api.Test;

class GenericTest {

@Test

void testCheckmatch() {

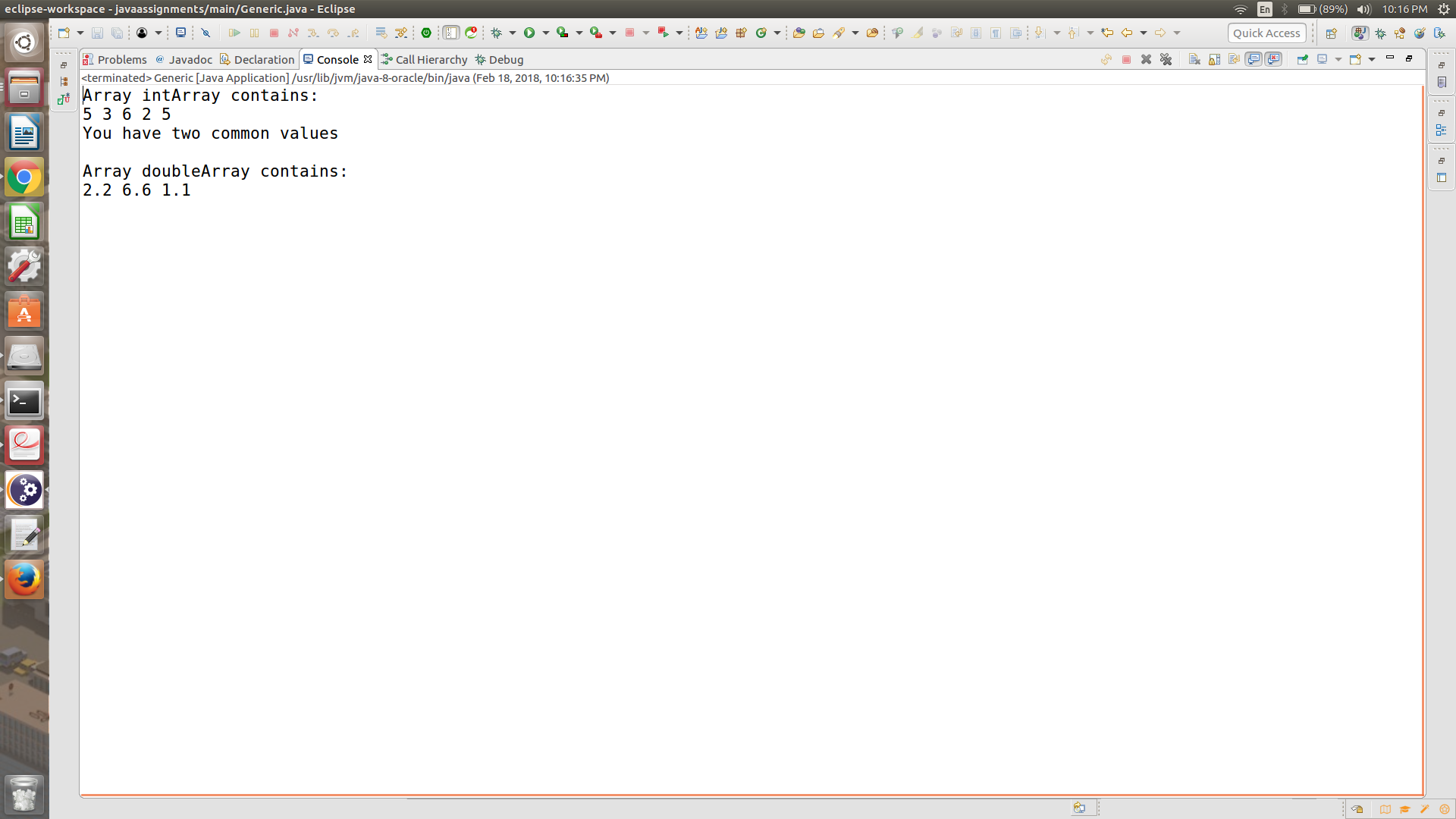
Integer[] intArray = { 5,3,6,2,5 };

boolean match=Generic.checkmatch(intArray);

assertEquals(true,match);

}

}

Output:

1. MultiThreading:

* Multithreading helps to execute multiple threads or programs simultaneously
* It operates on the basis of priority i.e. process with lower priority will get executed first.

**Program Scenario**: Here, we are creating three threading and run it. In each threading we create one array and print it. If one thread is interrupted by another thread, the output will not execute in line.

Java Code:

class ThreadDemo implements Runnable{

public void run(){

int i=0;

System.out.println("thread is running...");

int[] array = new int[10];

array = start(array);

print(array);

}

static int[] start(int[] array) {

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

array[i]=i+1;

}

System.out.println("thread may get interrupted during execution");

return array;

}

static void print(int[] array) {

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

System.out.println(array[i]);

}

}

public static void main(String args[]){

ThreadDemo td=new ThreadDemo();

Thread thread1 =new Thread(td);

Thread thread2 =new Thread(td);

thread1.start();

thread2.start();

Thread thread3 =new Thread(td);

thread3.start();

}

}

**JUnit Test Code:**

import static org.junit.Assert.\*;

import org.junit.Test;

public class ThreadDemoTest {

@Test

public void testStart() {

int[] array = new int[10];

array = ThreadDemo.start(array);

assertEquals(10, array[9]);

assertEquals(9, array[8]);

assertEquals(8, array[7]);

}

}

Output:

