# MIS2402 – Assignment 04

## JavaScript Conditional Statements, Input Validation and Business Rules

### Scenario:

In this assignment we are going to write a little “helper” application that represents logic to be used at an (imaginary) TSA airport security checkpoint. Imagine that the TSA agents just can’t quite remember what all the rules are, and that they need an application to help them know what to do next. Either that, or perhaps this application could be used to help train new TSA agents!

You will use JavaScript if statements can be used to do two things: validate user input and implement business rules.

### Instructions:

1. Connect to the class server. Use the username and password your instructor gave you. The class server’s hostname is: **misdemo.temple.edu**
2. After your successful login, click the “New terminal console” icon in Bitvise.
3. At the prompt, type:

cd wwwroot

1. At the prompt, type in this wget command:  
     
   wget **https**://misdemo.temple.edu/assignments/assignment04.zip
2. Unzip the file with an “unzip” command. This will create a folder called assignment04 and file named airporttsa.html.

unzip assignment04.zip

1. Now, enter these statements at the Linux prompt:

rm assignment04.zip

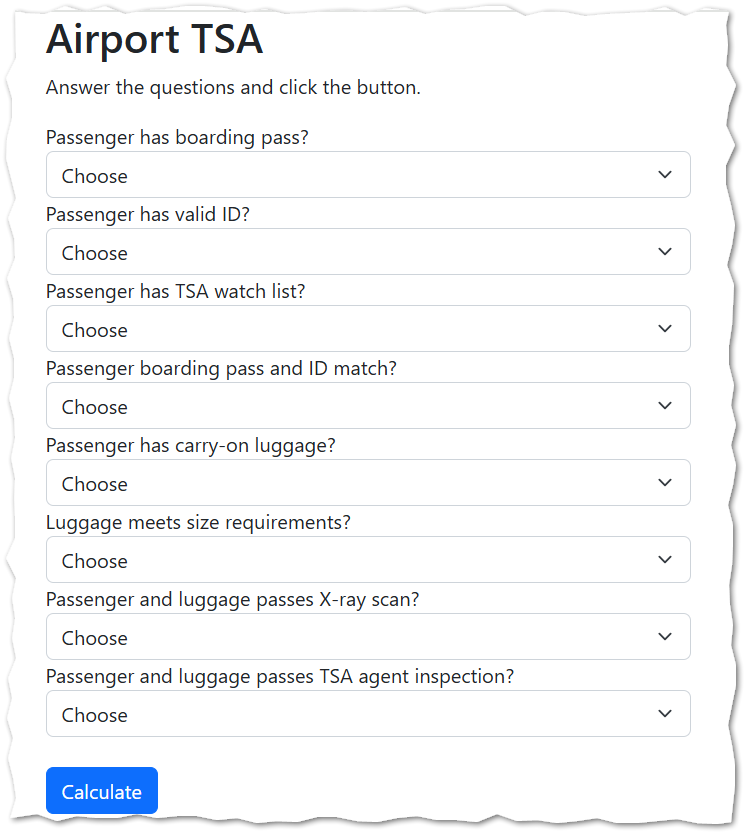
cd assignment04

1. Now, in Chrome, type this URL into your browser:

https://misdemo.temple.edu/**tuz54321x**/assignment04/airporttsa.html

You should replace tuz54321x with your own account.

1. You will be prompted to provide your misdemo username and password. After you do that. You should see a web page that looks like this:



1. Now, back in Bitvise, open the airporttsa.html file with nano.

nano airporttsa.html

1. Look inside the <script> tag in our html file. We will edit the functions found there.
2. **(Start with the error trapping / input validation)** Complete the validateForm function so that, if any one of the input variables is an empty string, the function will return “N”. Otherwise, the function should return “Y”.

1. Even though you have not completed the second function, you can test the validateForm function using the web developer console in Chrome. Like this:

validateForm("Y","Y","Y","Y","Y","Y","Y","Y")
'Y'
validateForm("N","N","N","Y","Y","Y","Y","Y")
'Y'
validateForm("","","","Y","Y","Y","Y","Y")
'N'
validateForm("","N","Y","N","N","Y","Y","Y")
'N'
validateForm("","Y","Y","Y","Y","Y","Y","Y")
'N'

Remember, you are not validating the passenger here… you are validating the HTML form. The form is considered valid here if all the questions have been answered.  
  
The question to allow / not allow the passenger past the security checkpoint is the purpose of the next function.

1. **(Implement the business rules )** Complete the getMessage function so that the business rules are implemented. The three possible messages are:

“Yes - admit”

“No - reentry allowed”

“No - detain”

The “No - detain" option is used to help ensure the safety of passengers and prevent crime.

|  |
| --- |
| The business rules for our imaginary TSA Security checkpoint are represented by the workflow diagram at the end of this document. |

1. Save your work (Control-O). Test your work. Do your responses follow the business rules documented in the diagram?
2. Determine the URL for your work. Go find the corresponding assignment on canvas and put the URL to your work here.

## How will this assignment be graded?

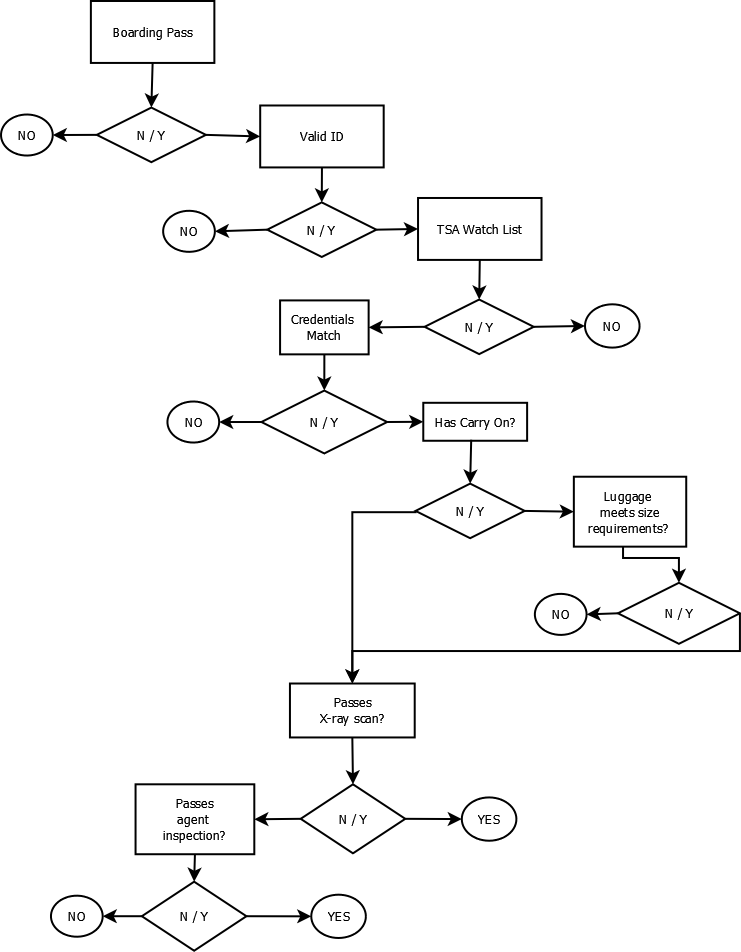
100 – You provided a good, working URL, everything runs, and your answers are correct.

80 – Your code has one or two small problems (such as a miscalculation or a formatting problem)

50 – Your code has multiple problems, but it does not crash and you made an effort.

0 – Your code is missing, your code completely crashes, you failed to provide a good URL, or you solved the problem using language features not covered in class.

## Workflow Diagram / Business Rules



DETAIN

DETAIN

While workflow diagrams are time-consuming to produce, and of limited value in day-to-day work, they can come in handy sometimes. Notice the efforts made here to make this illustration accurate, consistent, and easy-to-read.

* The diagram is read from top to bottom.
* The diamond shape consistently represents a decision. The “Yes” decisions consistently flow to the right (the direction we read in), and the “No” decisions flow to the left.
* The oval shape consistently represents the end of the workflow: the YES or NO outcome.