JavaScript / Testing & Debugging Exercise

To do first: Download **DebuggingExercise.zip** from the course page.
- ✓ There you find the files *DebuggingExercise.html* and *DebuggingExercise.js*

1. Open **DebuggingExercise.html** in a web browser.

2. The program should classify a given age (minor / adult). A valid age is an integer between 0 and 122. The absolute minimum set of ***test cases*** includes six *equivalence classes* as below[1].

   Test the program with the following values:

Value	Expected output	Equivalence class
25	*You're an adult.*	Age of majority
12	*You're a minor.*	Age of minority
123	*Please enter an integer between 0 and 122.*	Too high value of age
-1	*Please enter an integer between 0 and 122.*	Too low value of age
XYZ	*Please enter an integer between 0 and 122.*	The value is of invalid type
	Please enter an integer between 0 and 122.	Value of age is not entered

3. The program does not seem to crash, but it shows some incorrect output. Let's use the **debugger** to track the error(s).

   3.1 Open the debugger.

   3.2 In the debugger, set a breakpoint at **line 7** in the JavaScript code.

   3.3 Run the program and enter **12** as age.

   3.4 Execute the code line by line and follow the values of the variables.

   3.5 Is the inputted age correct?

   If it is not correct, then see the code where an age is inputted. There you find an error.
   - ✓ Make the required correction to the JavaScript code in your editor. Remember to save the code after the modification.
   - ✓ **NB!** Do ***not*** make any other corrections yet.

4. Let's see if the program shows the expected output now.

   4.1 Run the program and enter **12** as age.

   4.2 Execute the code line by line and follow the values of the variables.

   4.3 Is the inputted age correct?
   - ✓ If it is correct, then continue executing the code line by line ...

   4.4 Does the program execution go to the desired part in the conditional statement?
   - ✓ If it doesn't, then study the conditional statement in the code and make the required correction. Do not make any other corrections yet.

5. Let's see if the program shows the expected output now.

   5.1 Disable the breakpoint to speed up testing.

   5.2 Test the program again with the six values specified in point 2.

[1] Note: We keep this minimal to save your time. In the real life, we should also include boundary values in our test cases. That is, **0** as the first age of minority, **17** as the last age of minority, **18** as the first age of majority, and **122** as the last age of majority. This type of testing can be automated with the help of testing tools.

6. The program does not pass all of the test cases yet. Let's use the debugger again to see what happens in the program.
   - 6.1 In the debugger, set a breakpoint at **line 7** in the JavaScript code.
   - 6.2 Run the program. **Leave the age textbox empty** and click on OK.
   - 6.3 Execute the code line by line and follow the values of the variables.
   - 6.4 What is the value of age now?
     - ✓ If there is a problem, then make the required correction to the JavaScript code.
     - ✓ See the last page in the theory handout "Conditional Statements". There you see how to improve the JavaScript code.
   - 6.5 Repeat the previous steps until the program shows the expected output.

7. Let's see if the program finally shows the expected output on all of the test cases.
   - 7.1 Disable the breakpoint to speed up testing.
   - 7.2 Test the program again with the six values specified in point 2.

8. If the program shows the expected output, then your error tracking / debugging process has succeeded. Congratulations!

   If the program does not pass all test cases yet, then you have missed something and you should continue the process until you have got rid of the errors.

SUMMARY

You can use the debugger
- ✓ To track errors
- ✓ To study the logic of a given program

To systematically verify if a simple program seems to work correctly enough,
- ✓ You determine different types of situations (like age is too low, age is too high, age of minority, age of majority etc.) called ***equivalence classes*** and specify at least one ***test case*** (input & expected result) per each equivalence class.
- ✓ You run tests to ***find errors***. In general, testing cannot prove that the program does not contain any errors. The objective of testing is to find errors.
- ✓ When you have found an error, you can use a ***debugger*** to locate the source of the error in your code.
- ✓ The process ends when you run the whole set of your test cases and the program passes all of them.
- ✓ This type of repetitive testing can be automated with the help of a testing tool that controls the execution of tests and the comparison of actual outcomes with expected outcomes.

Hint: *Before the final exam, it is useful to recap the given idea of this type of testing...*