Chapter X – Testing and Debugging

Type of errors:

* System error ― there’s a problem with the system or external devices with which the program is interacting.
* Programmer error ― the program contains incorrect syntax or faulty logic; it could even be as simple as a typo.
* User error ― the user has entered data incorrectly, which the program is unable to handle.

The importance of testing and debugging:

We can use strict mode to prevent errors only we need to write ‘use strict ’; in the first line of JavaScript file, we can use strict mode in an function we can add in the first the strict mode only will work in the function.

We can use the alert() to make sure if our programm it work , the method alert make stop the programm and we make sure if our program it currently working.

The console.trace() method will log an interactive stack trace in the console. This will show the functions that were called in the lead up to an exception occurring while the code is running.

One of the most useful commands is the debugger keyword. This will create a breakpoint in your code that will pause the execution of the code and allow you to see where the program is currently up to. You can also hover over any variables to see what value they hold at that point. The program can then be restarted by clicking on the 'play' button.

Note: Remember to remove any references to the debugger command before shipping any code, otherwise the program will appear to freeze when people try to use it!

The throw statement can be applied to any JavaScript expression, causing the execution of the program to stop. For example, all the following lines of code will cause a program to halt:

Example:

throw 2;

throw 'Up';

throw { toys: 'out of pram' };

As an example, let’s write a function called squareRoot() to find the square root of a number. This can be done using the Math.sqrt() method, but it returns NaN for negative arguments. This is not strictly correct (the answer should be an imaginary number, but these are unsupported in JavaScript). Our function will throw an error if the user tries to use a negative argument:

function squareRoot(number) {

'use strict';

if (number < 0) {

throw new RangeError('You can't find the square root of negative numbers')

}

return Math.sqrt(number);

};

It is possible to handle exceptions gracefully by catching the error. Any errors can be hidden from users, but still identified. We can then deal with the error appropriately perhaps even ignore it and keep the program running.try , catch , and finally.

A Good way to test our code is do a tests before any actual code .

Practice:

<https://github.com/michell179708/web-fronted/blob/main/week05/practice.js>