TASK

Towards Defensive Programming I — Error Handling

Visit our website

Introduction

WELCOME TO THE TOWARDS DEFENSIVE PROGRAMMING TASK!

You should now be quite comfortable with basic variable identification, declaration and implementation. You should also be familiar with the process of writing basic code which adheres to correct JavaScript formatting to create a running program. In this task, you'll be exposed to error handling and basic debugging to fix issues in your code, as well as the code of others — a skill that is extremely useful in a software development career!

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact an expert code reviewer to get support on any aspect of your course.

The best way to get help is to login to Discord at **https://discord.com/invite/hyperdev** where our specialist team is ready to support you.

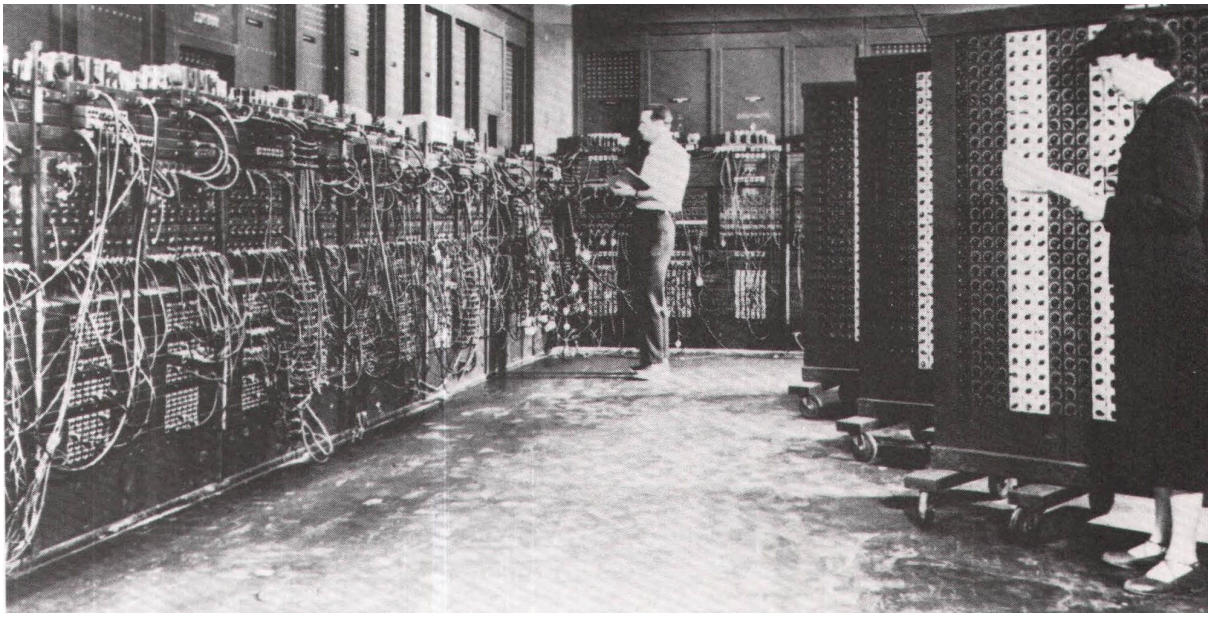
Our expert code reviewers are happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

DEBUGGING

Debugging is something that you, as a software engineer and web developer, will come to live, breathe and sleep. Debugging is when a programmer looks through their code to try and find a problem that is causing some error.

The word 'debugging' comes from the first computers — that is, when a computer was roughly the size of an entire room. It was called debugging because bugs (as in, living insects) would get into the computer and cause the device to function incorrectly. The programmers would need to go in and remove the insects, hence — debugging.

One such computer is the ENIAC. It was the first true electronic computer and was located in the University of Pennsylvania.



An image of the ENIAC.

Notice how the typical computer we know today is significantly smaller and also much more powerful than older computers like the ENIAC (despite their enormous size). It goes to show how, in just a few years, we have immensely expanded the computational power of computing devices. We owe that to the ever-changing and growing world of technology. So even though sometimes keeping up with the trends may be tiring, keep in mind that it's for the best!

You can do some more research about the ENIAC if you want on the University of Pennsylvania's website: **http:// www.seas.upenn.edu/about-seas/eniac/**.

Software developers live by the motto: “try, try again!” Testing and debugging your code repeatedly is essential for developing effective and efficient code.

DEALING WITH ERRORS

Everyone makes mistakes. It is likely that you've made some already. It's important to be able to DEBUG your code. You debug your code by removing errors from it.

Types of errors

There are three main errors that can occur in JavaScript:

- **Syntax errors**: These errors are due to incorrect syntax used in the program (e.g. wrong spelling, incorrect indentation, missing parentheses, etc.). The compiler can detect such errors. If syntax errors exist when the program is compiled, the compilation of the program fails and the program is terminated. Syntax errors are also known as **compilation errors**. They are the most common type of error.

- **Runtime errors**: These errors occur during the execution of your program. Your program will compile, but the error occurs while the program is running. An error message will also pop up when trying to run it. Examples of what might cause a runtime error include dividing by zero or referencing an out of range list item. This type of error is known as a runtime error because the error is only picked up by the computer at runtime.

- **Logical errors:** Your program’s syntax is correct, and it runs and compiles, but the output is not what you are expecting. This means that the logic you applied to the problem contains an error. Logical errors can be the most difficult errors to spot and resolve. Planning your program carefully with pseudo-code significantly reduces logical errors.

SYNTAX ERRORS

Syntax errors are comparable to making spelling or grammar mistakes when writing in English (or any other language), except that a computer requires perfect syntax to run correctly.

Common JavaScript syntax errors include:

- Leaving out a keyword or putting it in the wrong place

- Misspelling a keyword (for example a function or variable name)
- Leaving out an important symbol (such as a semicolon, curly bracket or parentheses)

Below is a typical example of a compilation error message. Syntax errors are the most common type of error in JavaScript. They can get a little complicated to fix when dealing with loops and if-statements. For example:

```
console.log("Hello);
```

This will throw the following:

```
error: SyntaxError: Unterminated string constant (1:12)
```

When a syntax error occurs, the line in which the error is found will also be underlined in red.

Go to the line indicated by the error message and correct the error, then try to compile your code again. If you cannot see an error on that line, see if you have made any syntax errors on previous lines that could be the source of the problem.

Debugging is not always fun but it is an essential part of being a programmer!

RUNTIME ERRORS

In JavaScript, there are three types of runtime errors:

- **TypeError:** These errors are thrown when there is a value that is not of a proper data type. For example:

```
console.logg("Hello");
```

This will throw the following:

```
error: Uncaught TypeError: console.logg is not a function
```

- **ReferenceError:** This error is thrown when we are trying to refer to something that doesn't exist, for example, if we are trying to use a variable we haven't created yet. For example:

```
console.log(randomVariable);
```

This will throw the following error message:

```
error: Uncaught ReferenceError: randomVariable is not defined
```

- **RangeError:** This type of error is thrown when a value is not allowed in a particular range. For example:

```
console.log('string'.repeat(-1))
```

This will throw the following error message:

```
error: Uncaught RangeError: Invalid count value
```

It's important to read error messages carefully and think in a deductive way to solve runtime errors. It may at times be useful to copy and paste the error message into Google to figure out how to fix the problem.

LOGICAL ERRORS

You are likely to encounter logical errors, especially when dealing with loops and control statements. Even after years of programming, it takes time to sit down and design an algorithm. Finding out why your program isn't working takes time and effort but becomes easier with practice and experience.

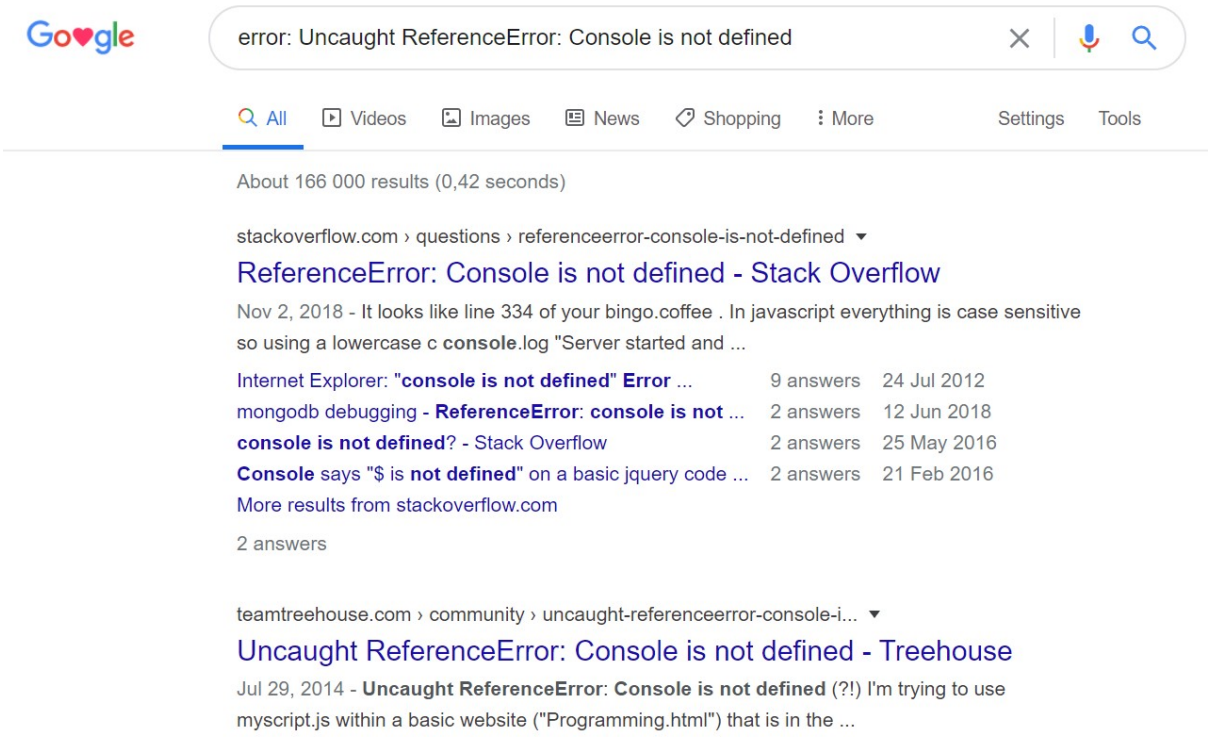
TIPS FOR DEBUGGING

You have probably seen plenty of errors similar to the one shown in the image below.

```
Console.log("Hello");
error: Uncaught ReferenceError: Console is not defined
```

Many of the errors are quite easy to identify and correct. However, often you may find yourself thinking, “What does that mean?”. One of the easiest, and often most effective, ways of dealing with error messages that you don’t understand, is to simply copy and paste the error into Google. Many forums and websites are

available that can help you to identify and correct errors easily. **Stack Overflow** is an excellent resource that software developers around the world use to get help.



Instructions

Open **example.js** and the **Error Practice** folder in Visual Studio Code and read through the comments before attempting these tasks.

Getting to grips with JavaScript takes practice. You will make mistakes in this task. This is completely to be expected as you learn the keywords and syntax rules of this programming language. It is vital that you learn to debug your code. To help with this remember that you can:

- Use either the JavaScript console or Visual Studio Code (or another editor of your choice) to execute and debug JavaScript in the next few tasks.
- Remember that if you really get stuck, you can contact an expert code reviewer for help.

Compulsory Task 1

Follow these steps:

- Open **errors1.js** in your task folder.
- Attempt to run the program. You will encounter various errors.
- Fix the errors and then run the program.
- Each time you fix an error, add a comment in the line where you fixed it and indicate which of the three types of errors it was.
- Save the corrected file.
- Do the same for **errors2.js**

Compulsory Task 2

Follow these steps:

- Create a new JavaScript file called **creatingErrors.js**.
- Write a program with **two compilation errors**, a **runtime error** and a **logical error**.
- Next to each error, add a comment that explains what type of error it is and why it occurs.

Rate us

Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved, or think we've done a good job?

Click here to share your thoughts anonymously.