Hyperiondev

TASK

Defensive Programming - Error Handling

Visit our website

Introduction

WELCOME TO THE DEFENSIVE PROGRAMMING - ERROR HANDLING 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 Python formatting to create a running program. In this lesson, 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 development/programming career!

WHAT IS DEFENSIVE PROGRAMMING?

Defensive programming is an approach to writing code where the programmer tries to anticipate problems that could affect the program and then takes steps to defend the program against these problems. MANY problems could cause a program to run unexpectedly!

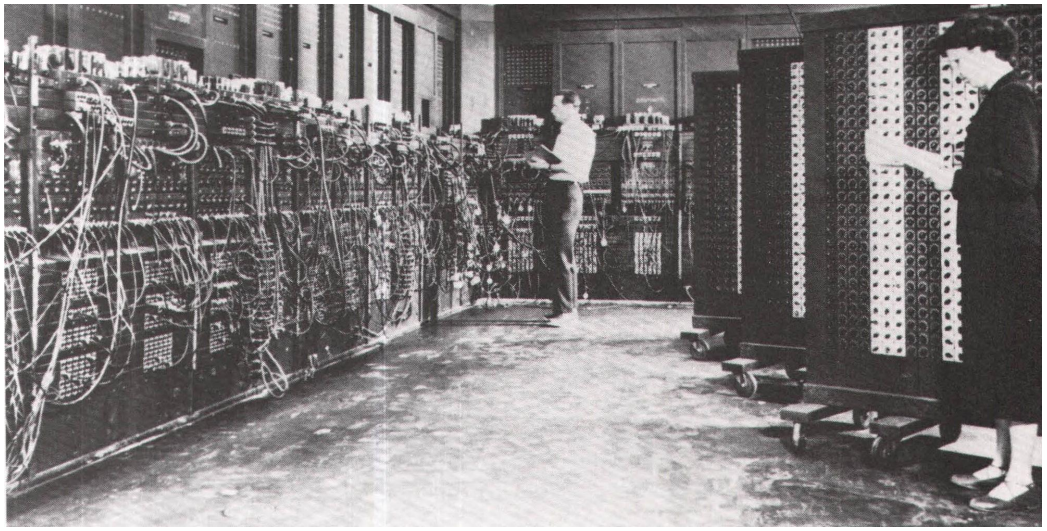
DEBUGGING

Debugging is something that you, as a software developer/programmer, will come to eat, sleep, and breathe! Debugging is when a programmer looks through their code to try and identify a problem that is causing some error.

The word "debugging" comes from the first computers, back when a computer was roughly the size of an entire room. Bugs (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 the name - debugging.

One such computer is the ENIAC. It was the first true electronic computer and was located in the University of Pennsylvania. Computers as we know them today are 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 can expand our computational power immensely. In terms of the rate of growth on computational power, Gordon Moore (co-founder of Fairchild Semiconductor and Intel (and former CEO of the latter)), predicted in the 1960s that the number of transistors in an integrated circuit would double about every two years, a prediction which has been borne out as correct over time. In the world of tech it is necessary

to actively work to keep our knowledge and expertise current! If you're interested, you can do some more research about **the ENIAC on the University of Pennsylvania's website**.



The ENIAC

*Source: **Papercut***

Software developers live by the motto: "try, try, try again!" Testing and debugging your code repeatedly is essential for developing effective and efficient code and achieving mastery as a developer/programmer.

DEALING WITH ERRORS

Everyone makes mistakes. Likely you've made some already, and that's not a bad thing! As you've probably already realised, however, just making the mistakes doesn't facilitate any learning – it's important to be able to DEBUG your code. You debug your code by identifying various types of errors, and correcting them. What type of errors might you encounter? Let's take a look.

Types of errors

There are three different categories of errors that can occur:

- **Syntax 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** occur during the execution of your program. The program will compile as normal, but the error occurs while the program is running. An error message will likely also pop up when trying to run the buggy program. Examples of what might cause a runtime error include dividing by zero, and referencing an out-of-range list item (if you're unfamiliar with lists and impatient to learn more, have a look **here**!).

- **Logical errors** occur when 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, because you need to identify where you have gone wrong in your thought process. Building **metacognitive skills** can help you gain proficiency in detecting logical errors. A useful HyperionDev resource for building metacognitive skills is **available here**.

Now that you know the three types of errors you might encounter, let's consider each in more detail.

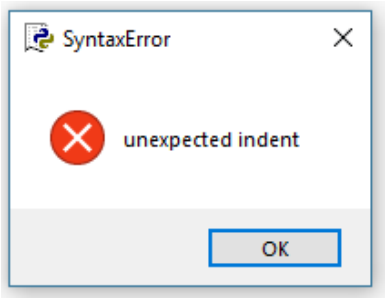
Syntax errors

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 Python 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 colon, comma, or parentheses).
- Incorrect indentation.
- Leaving an empty block (for example, an *if statement* containing no indented statements).

Below is a typical example of a compilation error message. Compilation errors are the most common type of error in Python. They can get a little complicated to fix when dealing with loops and if statements!



When a syntax error occurs, the line in which the error is found will often be highlighted (although exactly how depends on your development environment (IDE)). The cursor may also automatically be placed at the point the error occurred to make the error easy to identify. However, watch out - this can be misleading! For example, if you leave out a symbol, such as a closing bracket or quotation mark, the error message could refer to a place later on in the code, which is not the actual source of the problem.

Nonetheless, when you get a syntax error, go to the line indicated by the error message and correct the error if possible. If it is not in that line, work backwards through the code until you identify the source of the problem and then correct it. Then try to compile your code again. Debugging is an essential part of being a programmer, and although sometimes frustrating, it can be quite fun to problem-solve the causes of your errors!

Runtime errors

Using your knowledge of strings, take a look at the code below and see if you can identify the runtime error:

```
greeting = "Hello!"
print(greeting[6])
```

This would be the error you get, but why?

```
Exception has occurred: IndexError
string index out of range
```

Look carefully at the description of the error. It must have something to do with the string index. String characters are indexed from zero, and so the final index for "!" would be five, not six. That means that nothing exists for `greeting[6]`, so we get a runtime error. It's important to read error messages carefully and think in a deductive way to solve them. 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, logical errors still occur, and more so if you dive into coding solutions to problems without planning the high-level approach using pseudo code. This pseudo code planning, so often skipped by students eager to start coding, is a vital component of logical planning for problem solving. It takes time to sit down and design an algorithm, but it makes later debugging much easier.

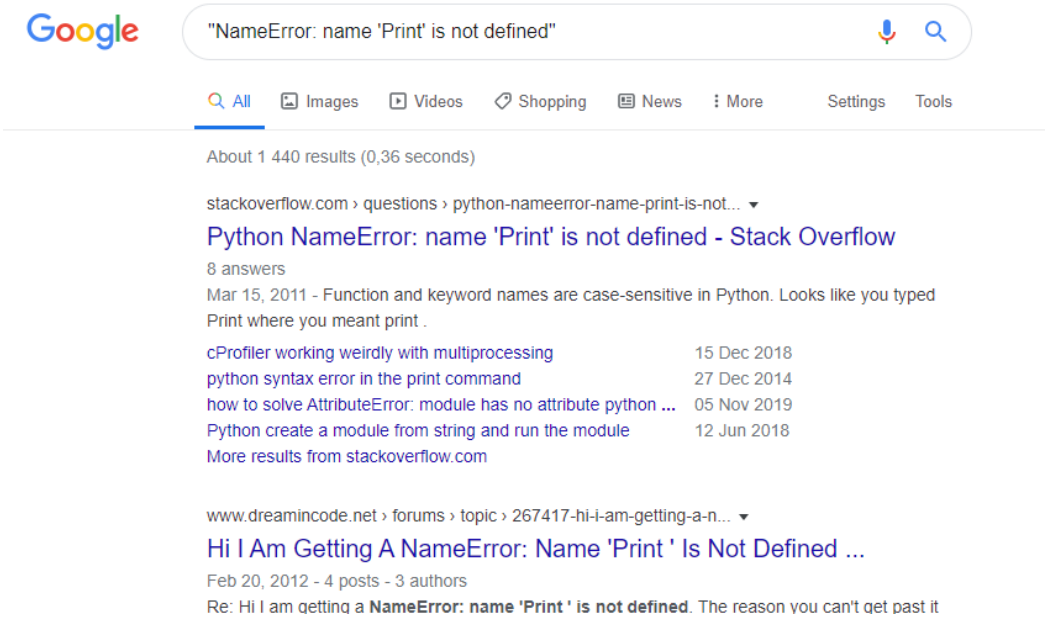
TIPS FOR DEBUGGING

Debugging becomes easier with practice and experience. You have probably seen plenty of errors similar to the one shown in the image below in the Python shell.

```
Print("Hello World!")
NameError: name 'Print' 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.



LLMs like OpenAI's ChatGPT or Google's Bard can also be really helpful in debugging tricky errors!

In addition, your IDE may have some other built-in features to help you identify and correct subtle errors. Find out if it has a debugger and how to turn it on if it is not on by default (**here are the instructions for VS Code**).

Many debuggers will allow you to do things like "Go", "Step", etc. This part of debugger functionality is especially important if you use breakpoints in your code. A breakpoint is a point in your code where you tell it to pause the execution of the code so that you can check what is going on. For example, if you don't understand why an *if statement* isn't doing what you think it should, you could create a breakpoint at the *if statement*. You can follow this **video** to learn how to set breakpoints in VSCode.

Instructions

- First, read and run the **example files** provided. Feel free to write and run your own example code before doing the Practical Task to become more comfortable with the concepts covered in this task.

Practical Task 1

For both the **errors.py** and **errors2.py** task files in your folder, open the files and follow these steps:

- 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 with a brief explanation of why that is.
- Save the corrected file.

Practical Task 2

Follow these steps:

- Create a new Python file called **logic.py**.
- Write a program that displays a logical error (be as creative as possible!).

Challenge

Use this opportunity to extend yourself by completing an optional challenge activity.

Follow these steps:

- Create a new Python file called **optional_challenge.py**.

- 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

Hyperion 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.