TASK

Defensive Programming — Error Handling

Visit our website

Introduction

Welcome to the 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 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 your mentor to get support on any aspect of your course.

The best way to get help is to login to **www.hyperiondev.com/portal** to start a chat with your mentor. You can also schedule a call or get support via email.

Your mentor is 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!

Dealing with Errors

Everyone makes mistakes. Likely 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 different categories of errors that can occur:

- **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**: Runtime 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.

- **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.

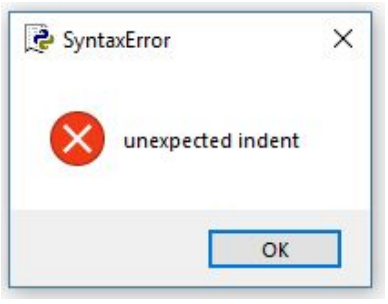
Syntax Errors

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 also be highlighted in red. The cursor may also automatically be put there so that error is easily found. However, 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.

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

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. If you recall from Task 5, we could indices from 0, and so the final

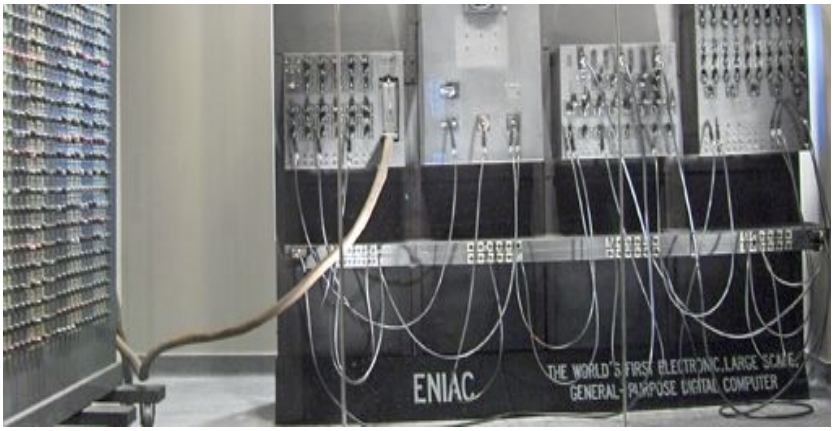
index for “!” would be 5. That means that nothing exists for index 6, so we get a runtime error. It's important to read error messages carefully and think in a deductive way to solve runtime errors. It may at times be useful to copy and paste the error message into Google to figure out how to fix the 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.

A note from our coding mentor
Joseph

The term ‘Debugging’ comes from when bugs (yes, literal bugs - insects) caused problems in computers. This happened when computers were as big as rooms! One of the first computers was known as ENIAC. This computer was located at the University of Pennsylvania. Riaz Moola, the Founder of Hyperion Development, studied at the University of Pennsylvania where ENIAC is still on display (see ENIAC in the image).



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

Instructions

First, read **example.py**. Open it using IDLE.

- **example.py** should help you understand some simple Python. Every task will have example code to help you get started. Make sure you read all of **example.py** and try your best to understand.

- You may run **example.py** to see the output. Feel free to write and run your own example code before doing the Task to become more comfortable with Python.

Compulsory Task 1

Follow these steps:

- Open **errors.py** 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 **example_errors.py**

Compulsory 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!).

Optional Bonus Task

Follow these steps:

- Create a new Python file in this folder called **optional_task.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 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.