Hyperion Development

Task: Defensive programming - Error Handling

www.hyperiondev.com

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 the correct Python formatting to create a running program. This task is aimed at furthering your knowledge of types of variables to create more functional programs. You'll also be exposed to error handling and basic debugging in order to fix issues in your code, as well as the code of others - a skill which is extremely useful in a software development career!

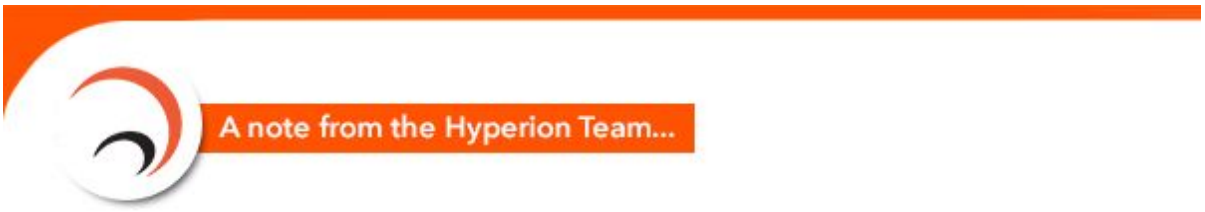
Connect with your mentor

CONNECT

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/support** 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!



There are three distinct numeric types in Python 3: integers, floating point numbers and complex numbers.

- *Integers: Integers are positive or negative whole numbers. In Python 3 the size of an integer is unlimited. E.g of integer:* `num = int("-12")`
- *Floats: Floating point numbers are generally numbers with a decimal point. Floats may also be in scientific notation, with E or e indicating the power of 10. E.g. of floats:*
  - `x = float("15.20")`
  - `y = float("-32.54e100")`
- *Complex: Complex numbers have a real and imaginary part, which are each a floating point number. E.g. of complex number:* `c = complex("45.j")`

-The Hyperion Team

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 3 different categories of errors that can occur:

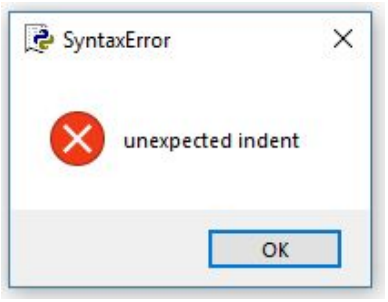
- **Compilation errors:** When you try to compile your code the program will immediately crash. This is due to syntax errors (wrong spelling, incorrect indentation, missing parenthesis etc).

- **Runtime errors:** Your program compiles but when it is actually run, an error occurs and the program stops! An error message will also pop up when trying to run it.

- **Logical errors:** Your program runs and compiles but the output isn't what you're expecting. This means that the logic you applied to the problem contains an error.

Compilation Errors

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 compilation error occurs, the line in which the error is found will also be highlighted in red and the cursor will automatically be put there so that error is easily found.

Go to the line indicated by the error message and correct the error, then try to compile your code again. Debugging isn't always fun but it is an important part of being a programmer!

Runtime Errors

Say we had this line in our program: `num = int("18.2")`
Your code would compile properly, but when running the code you'd get the error:

```
ValueError: invalid literal for int() with base 10: '-12.3'
```

Look carefully at the description of the error. It must have something to do with the format of the number you assigned to the variable 'num' when trying to parse a String into an Integer. You can't cast 18.2 to an Integer because integers don't have decimal points. 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.

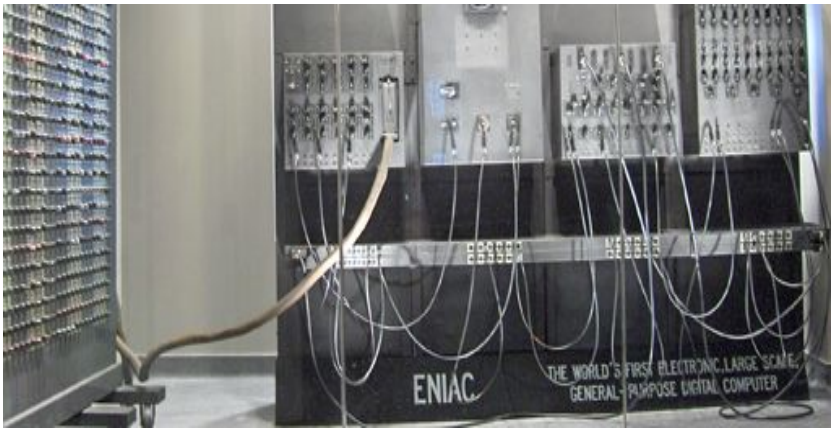
Logical Errors

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 Masood…

*Hey again, I'm sure you have heard the term **debugging** before? No? It's not an extermination service, well it is kind of, let me explain.*

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.

- **Masood Gool**

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.
- Save the corrected file.
- Now run the program and notice the output - fix the error and add comments to the code to indicate which of the three types of errors it was.
- Each time you fix an error, add a comment in the line you fixed it and indicate which of the three types of errors it was.
- Do the same for 'exampleErrors.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 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.

Give 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.