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 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 will 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!

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!

A note from the

HyperionDev Team

Along with the numeric data types we learnt about earlier, 'complex numbers' are an additional type of number in Python. Of the three kinds of numbers below, you are familiar with integers and floats as we studied them earlier, but complex numbers may be new for you.

- 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")`

DEALING WITH ERRORS

Everyone makes mistakes, including programmers. However, when there are mistakes in code that you've written (or someone else has written), 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:** Compiler errors are due to inaccuracies in code, where the compiler throws you an error to point out something which will not compile and therefore cannot be run. This is due to incorrect syntax or semantics or other such errors (wrong spelling, incorrect indentation, missing parentheses, doing operations that aren't 'allowed' such as adding a string with an int, etc.)

- **Runtime errors:** Runtime errors are detected when the program executes. This means your program compiles because the compiler found no inaccuracies in the code, but when the program 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.

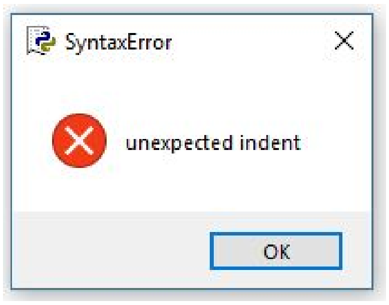
COMPILATION ERRORS

Consider the code below.

```
Print("Hello World!")
```

The statement above is an example of a syntax error. As you can see the print keyword has a capital 'P', but Python is case sensitive so `Print` and `print` are two different things. Copy the example command above and try running it — what error message do you get?

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.



SyntaxError

unexpected indent

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.

See another example below.

```
word = "Python"
character = word[6]
```

The statement above is an example of a runtime error. As you can see, the second statement is referencing an index that is out of bounds. Try running it - what error message do you get?

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 every once in a while, 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.

For example:

```
area_trapezoid = 3 + 5 * 6 / 2
print(area_trapezoid)
```

The code above is an example of something that would throw a logical error. This code aims to calculate the area of a trapezoid with parallel sides of length 3 and 5, and a height of 6. The area of this trapezoid should be 24, but when you run this program, you'll see that this is not the case. The reason is the missing brackets around 3 + 5. The correct code should be:

```
area_trapezoid = (3 + 5) * 6 / 2
```

Hopefully you now see why logical errors are the most difficult to locate and fix. They don't crash the program, but they produce an incorrect result due to a mistake in the program's logic. You won't get an error message here, because no syntax or runtime error has occurred, yet your output will be inaccurate.

A note from our coding mentor

Nkosi

*I'm sure you have heard the term **debugging** before. No? It's not an extermination service - or, 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 HyperionDev, studied at the University of Pennsylvania where ENIAC is still on display (see ENIAC in the image).

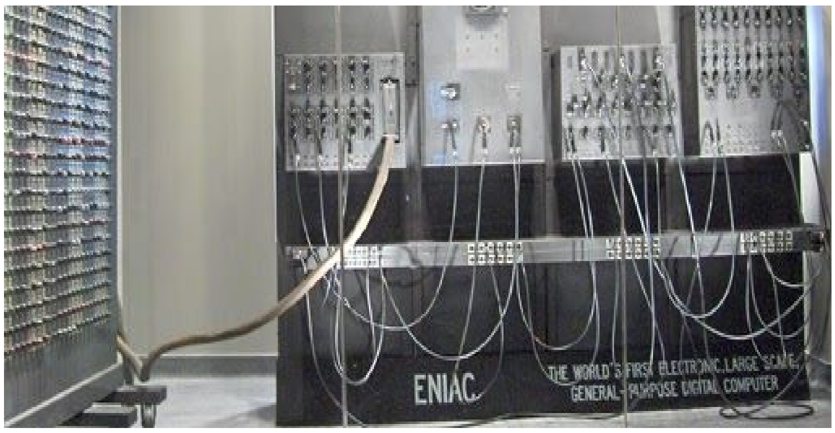

ENIAC
THE WORLD'S FIRST ELECTRONIC LARGE SCALE GENERAL-PURPOSE DIGITAL COMPUTER

Image source: ***https://news.mlh.io/hacking-history-visit-eniac-penn-09-14-2014***

Software developers live by the motto: "try, 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.

- Make sure to run **example.py** to see some examples of error and how they get displayed in output. Feel free to write and run your own example code before attempting the Task to become more comfortable with the topic.

Compulsory Task 1

Follow these steps:

- Open **errors1.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.
- Whenever you 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, be sure to add a comment in the line you fixed it and indicate which of the three types of errors it was.
- Now do the same for **errors2.py**. Fix the errors, add comments to the code to indicate which errors you fixed, and finally run the code to make sure it produces the desired result.

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!).

Compulsory Task 3

Follow these steps:

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

Completed the task(s)?

Ask your mentor to review your work!

Review work

Things to look out for:

1. Make sure that you have installed and setup all programs correctly. You have setup **Dropbox** correctly if you are reading this, but **Python or Notepad++** may not be installed correctly.
2. If you are not using Windows, please ask your mentor for alternative instructions.

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.