Molly Weaver

February 27, 2021

Foundations of Programming: Python

Assignment 07 – Error Handling

<https://github.com/molweave532/IntroToProg-Python-Mod07>

Error Handling in Python

# Introduction

For homework 07 we were asked to explain the concept of error handling. I offer up a video and a web site for background information, then go through some basic coding as an example.

## Background Information

I am going to briefly introduce the concept of error handling in Python. The first thing I always do when trying to learn something new is to go online and look for videos of someone explaining the concept. I am a very visual learner. Sometimes it’s easier to learn by watching someone else do something than to read about it. You can always go back and read to get a more granular view. Here is a great starter video on error handling –

Python Tutorial: Using Try/Except Blocks for Error Handling – by Corey Schafer

<https://www.youtube.com/watch?v=NIWwJbo-9_8> (external link)

It’s only about ten minutes long but it covers all the basics of the try-except block. If you’d like more detail, here is a good place to go.

<https://realpython.com/python-exceptions/> (external link)

### The Try-Except Block

A try-except block is used to catch errors that will cause a program to stop executing, often related to faulty user input. The point is to catch the error and allow the user to fix the input rather than allowing the program to crash. The code is very simple (**figure 1**).

Text

Description automatically generated with medium confidence

***Figure 1.*** A basic try-except block.

Suppose the program is expecting a number as input and the user enters a string instead (**figure 2**).



***Figure 2.*** Expecting an integer.

Without a try-except block Python will give the following error message (**figure 3**).

Text

Description automatically generated

***Figure 3.*** Generic Python error message.

This may be readable to someone who is familiar with Python, but to a casual user it looks very dire and confusing. Also, it will crash the program. As you can see in the last line, this is a ValueError – Python was expecting a different type of value to be entered by the user, namely an integer. If we take our input statement and put it after a try statement, we can then create a custom error message (**figure 4.**)

Graphical user interface, text

Description automatically generated

***Figure 4.*** Catching the error and showing it to the user.

The line “except Exception as error:” will catch the error in the variable “error” and then print it to the screen without ending the program. It’s also slightly less intimidating to the casual user (**figure 5**).

A picture containing text, orange

Description automatically generated

***Figure 5.*** Less intimidating error message

We can also create a custom error message by leaving out “as error:” (**figure 6**).

Graphical user interface

Description automatically generated with medium confidence

***Figure 6.*** Creating a custom error message.

Running the code will still produce an error message if the user enters unexpected input (**figure 7**), but now we see our custom error message.

A picture containing chart

Description automatically generated

***Figure 7.*** Displaying the custom error message to the user.

That is definitely more readable, but it still isn’t very clear to the user what happened. It will also print the same error message no matter what type of error occurred. The term “Exception” catches all error messages and executes the code after the except statement. We can create more specific error messages by having multiple except statements after our try statement (**figure 8**).

A screenshot of a computer

Description automatically generated with medium confidence

***Figure 8.*** Multiple except statements.

You want to put the most generic except statement last in your try block. Only the first applicable except statement is executed. If we put a generic error as the first test, it would only execute the generic exception because it is designed to catch all errors and you would never get to the more specific tests. In this case, our first except statement will catch most user input errors (**figure 9**), but the second one is there just in case something unexpected happens.

A picture containing chart

Description automatically generated

***Figure 9.*** Custom and user-friendly error message.

#### The Else Clause

You can also add an else clause after the except statements (**figure 10**).

Graphical user interface, text, application

Description automatically generated

***Figure 10.*** A try-except block with an else clause.

An else clause will only execute if no errors occur in response to the try statement (**figure 11**).

A picture containing text

Description automatically generated

***Figure 11.*** Acceptable user input.

In this case, the user entered the expected value type, an integer, so it triggered the else clause. But what if the user entered a floating-point number (**figure 12**)?

A picture containing text, orange

Description automatically generated

***Figure 12.*** Acceptable input that will still throw an error message.

4.5 is a number between 1 and 10, but we are only looking for integers. It looks like this is also a ValueError because that is the except clause that was triggered. We’d better change our code to let the user know we are looking specifically for integers (**figure 13**).

Text

Description automatically generated

***Figure 13.*** Using more specific language.

It’s important to be specific in both your requests for user input and your error messages!

##### Raising a Custom Error

Now what if a user enters 45, or -12? Those are both integers, but they are outside the range we are looking for. Python allows you to create custom error messages to help define your user input more precisely (**figure 14**).

Graphical user interface, text, application

Description automatically generated

***Figure 14.*** Raising an exception to valid data.

In the above code, an exception is raised if the user enters an integer that is not part of the defined range. It’s not necessarily an error to Python, but it’s still not input that is desired. I chose to put the code in my else clause, so we already know the input is an integer. We are just testing to see if it is in the expected range (**figure 15**).

A picture containing text, orange, close

Description automatically generated

***Figure 15.*** A custom error message if the user enters undefined data.

###### The Finally Clause

There is one more statement that can be included in a try-else block, the finally statement. Unlike the else clause, finally will be executed whether or not an error occurs. It can be used to complete a necessary task before ending the program due to an error, or it can be used to go back and give the user a chance to re-enter their input. Here we will just print a message to the user (**figure 16**).

Graphical user interface, text

Description automatically generated

***Figure 16.*** The finally clause.

It will execute if the user enters faulty data (**figure 17**).

A white background with black text

Description automatically generated with low confidence

***Figure 17.*** The finally clause and faulty data.

It will also execute if the user enters acceptable data (**figure 18**).

A picture containing text

Description automatically generated

***Figure 18.*** The finally clause and acceptable data.

Conclusion

Error messages can occur in Python when the program encounters something unexpected. In this tutorial, I showed how to use error handling to define user data to conform to a desired data type and range without crashing the program. I introduced the try-except block to test that the data is of the desired type. If it isn’t, an error message is displayed. If it is, I used the else clause to display a message that the data entered is acceptable. I also showed how to create custom exceptions if the data entered isn’t in the desired range but doesn’t automatically trigger an error in Python. I ended with the finally clause to execute statements regardless of the type of data entered.