George Kim

February 21, 2019

Foundations of Programming: Python

Assignment 07

GitHub Link: <https://github.com/gjkim44/IntroToProg-Python>

Blog page link: <https://myfirstpythonblog.blogspot.com/2019/02/introduction-in-assignment-7-we-are.html>

Try, Except and Pickling

# Introduction

In Assignment 7, we are asked to do something a bit different compared to the previous assignments. The links to understand *Try, Except* and *Pickling* will be provided by us, the students. Professor Root, in our past assignments, provided us an overview with additional links to further our education and understanding on the topics we were covering. In this paper, I will be providing the additional links and the explanation of how Try and Except work in Python along with Pickling. I will be going over the steps of how I accomplished my task and will be providing screen shots and comments for explanation.

**Understanding Try, Except and Pickling**

First we have to understand the topics we are covering. What is *Try and Except*? What is *Pickling*? I will split this up into two parts to identify each topic. Let’s start with the first topic of *Try an Except*. There are some information links for your review, if you wish too, that I have inlcuded for : Try/Except and Pickling.

**Try and Except**

Links:

<https://realpython.com/python-exceptions/>

<https://www.programiz.com/python-programming/exception-handling>

<https://www.programiz.com/python-programming/user-defined-exception>

videos:

Corey Schafer

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

Mike Dane

<https://www.youtube.com/watch?v=KdMAj8Et4xk>

**Pickling**

Links:

<https://www.geeksforgeeks.org/pickle-python-object-serialization/>

<https://www.datacamp.com/community/tutorials/pickle-python-tutorial>

<https://www.thepythoncorner.com/2016/12/object-serialization-in-python/>

<https://pythonprogramming.net/python-pickle-module-save-objects-serialization/>

Video:

Mark Jay

<https://www.youtube.com/watch?v=Pl4Hp8qwwes>

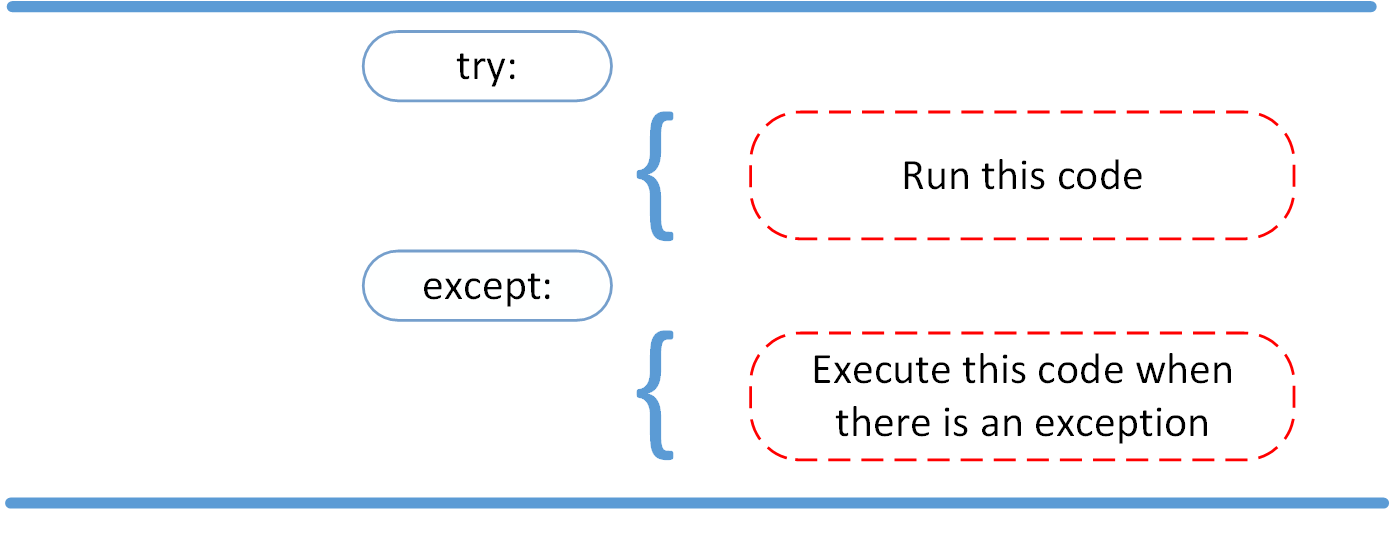
Master Code Online

<https://www.youtube.com/watch?v=s30yFHmhIe0>

**Try and Except**

As stated by the website www.realpython.com:

The try and except block in Python is used to catch and handle exceptions. Python executes code following the try statement as a “normal” part of the program. The code that follows the except statement is the program’s response to any exceptions in the preceding tryclause.



(realpython.com, <https://realpython.com/python-exceptions/>, 2019) (External Site)

Looking at the above diagram it is simple on how it works. You TRY a code out, EXCEPT if there is an issue, this code/error will display. What you are trying to do is catch errors ahead of time before your code is finished. You are trying to account for user errors or loading errors, file errors, etc. Here is a little better example below.

**try**:

**print** "Hello World"

**except**:

**print** "This is an error message!"

(www.pythonforbeginners.com, <https://www.pythonforbeginners.com/error-handling/python-try-and-except>, 2019)(External Site)

The error message displayed is to the user regarding the code being used. There are a wide variety of message that can be displayed, depending on the error. Here are a few examples:

Exception Errors

Some of the common exception errors are:

**IOError**

If the file cannot be opened.

**ImportError**

If python cannot find the module

**ValueError**

Raised when a built-in operation or function receives an argument that has the

right type but an inappropriate value

**KeyboardInterrupt**

Raised when the user hits the interrupt key (normally Control-C or Delete)

**EOFError**

Raised when one of the built-in functions (input() or raw\_input()) hits an

end-of-file condition (EOF) without reading any data

(www.pythonforbeginners.com, <https://www.pythonforbeginners.com/error-handling/python-try-and-except>, 2019)(External Site)

Here are some examples of what can be shown as a message to the user, for each Error above:

**except** IOError:

**print**('An error occured trying to read the file.')

**except** ValueError:

**print**('Non-numeric data found in the file.')

**except** ImportError:

**print** "NO module found"

**except** EOFError:

**print**('Why did you do an EOF on me?')

**except** KeyboardInterrupt:

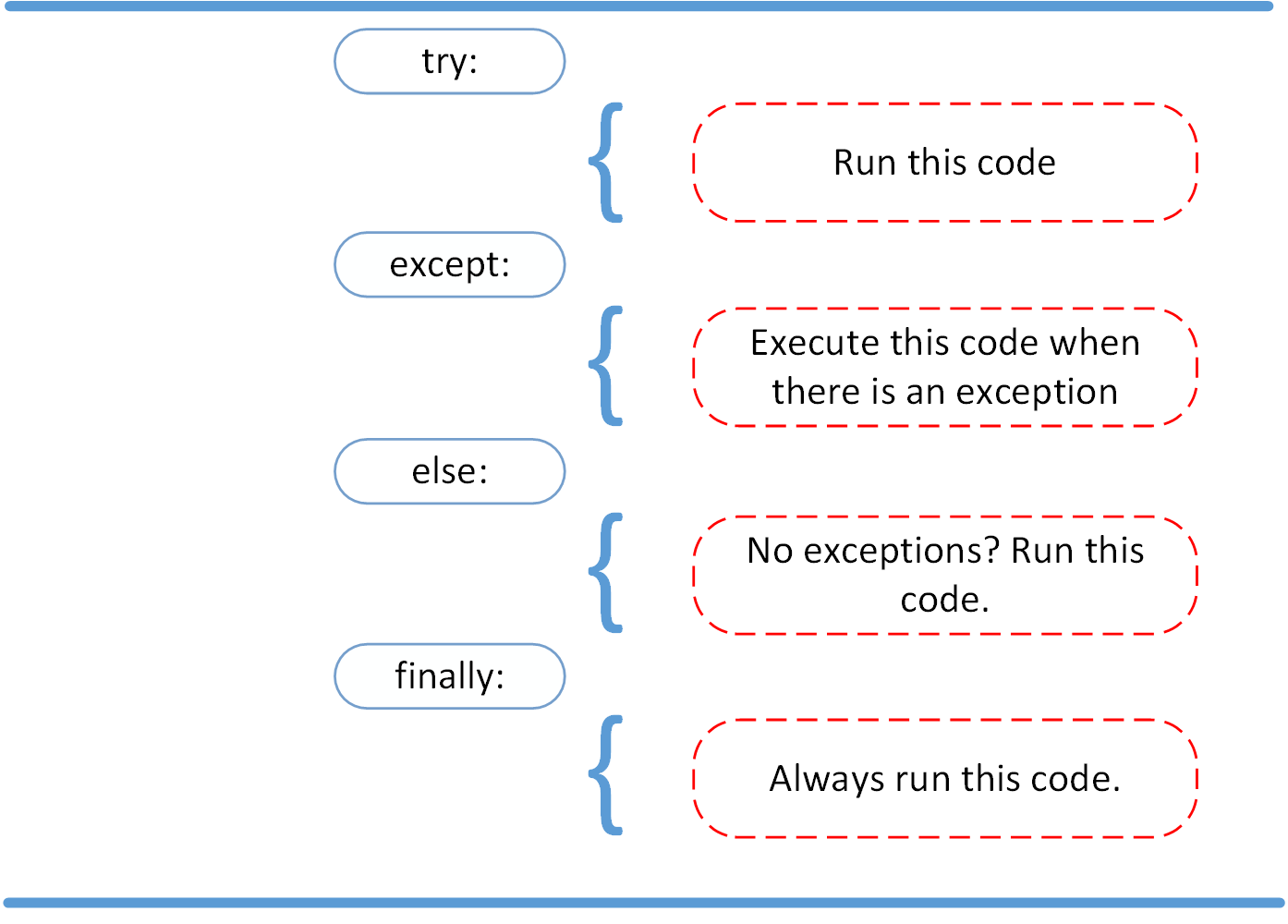
**print**('You cancelled the operation.')

**except**:

**print**('An error occured.')

(www.pythonforbeginners.com, <https://www.pythonforbeginners.com/error-handling/python-try-and-except>, 2019)(External Site)

Each message can be tailored to how you would like to have it displayed to the user depending on what type of code you have in the Try block(area). There are a couple final pieces to the Try/Except that will be discussed below, *else* and *finally*.



(realpython.com , <https://realpython.com/python-exceptions/>, 2019) (External Site)

In addition to the except block, you can add an *else block* and then a *finally block*. Basically, what the *else* does is running another line of code if there are no exceptions and the *finally block* will always run that line of code no matter what. I incorporated an example of how the *else* and *finally* block work down below.

try:

linux\_interaction()

except AssertionError as error:

print(error)

else:

try:

with open('file.log') as file:

read\_data = file.read()

except FileNotFoundError as fnf\_error:

print(fnf\_error)

finally:

print('Cleaning up, irrespective of any exceptions.')

Python Shell(output)

Function can only run on Linux systems.

Cleaning up, irrespective of any exceptions.

(realpython.com, <https://realpython.com/python-exceptions/>, 2019)(External Site)

So this is how the codes read: “ first I will TRY linux\_interaction(), EXCEPT if there is an error I will print the AssertionError message, ELSE if there is not an error I will TRY to open file.log, EXCEPT if there is another error I will print the FIleNotFoundError message, FINALLY at the end I will print “Cleaning up, irrespective of any exceptions.” What the python shell prints out to the user is in the above output.

Pretty simple, right? I will show it in action in my Python script but let’s learn a bit about *pickling* before we go there.

**Pickling**

Here are a couple of explanations of pickling:

**Python pickle** module is used for serializing and de-serializing a **Python** object structure. Any object in **Python** can be **pickled** so that it can be saved on disk. ...**Pickling** is a way to convert a **python** object (list, dict, etc.) into a character stream

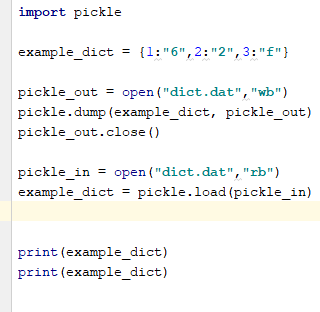
(<https://www.geeksforgeeks.org/understanding-python-pickling-example/>, 2019)(External Site)

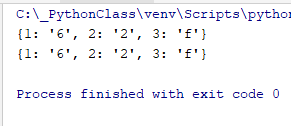
Pickling is the name of the **serialization** process in **Python**. By pickling we can convert an object hierarchy to a binary format (usually not human readable) that can be stored. To pickle an object we just need to import the pickle module and call the dumps() function passing the object to be pickled as a parameter.

(<https://www.thepythoncorner.com/2016/12/object-serialization-in-python/>, 2019)(External Site)

If you are still a bit confused on how pickling works I will try to make it a bit more simple, in python when you pickle some data (list, dictionary, etc) you are taking the data and just converting it to a different form and storing that in a data file. You then have to take that data out and convert it back to a python format.

Here is a quick example code that I got from pythonprogramming.net:

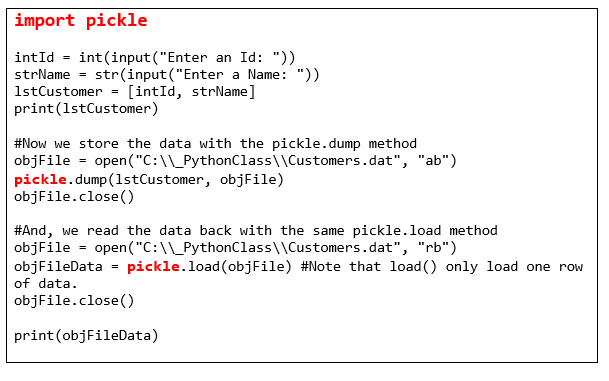




First you can use any name to open the *dat* file(pickle\_out, pickle\_in), in the example above the names are just used to illustrate the load and unloading of the data. You take a python object which is *example\_dict*, then you create text file name and open a file, *dict.dat* which converts the dictionary by using the “***pickle.dump(object, text file)*** and then close the text file created. The data in now converted to Binary! Now to convert the binary data back into python you just open the dict.dat again and use the variable in which the original object was in(example\_dict) and ***pickle.load(pickle\_in)*** to convert it back to python. You can see in the output above that the list are exactly the same. Does this make sense? If it doesn’t , I will be explaining it a bit further with my code that incorporates Try/except and pickling in one code.

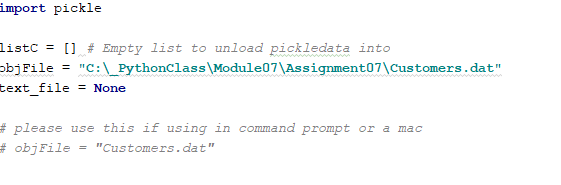
**My Script**

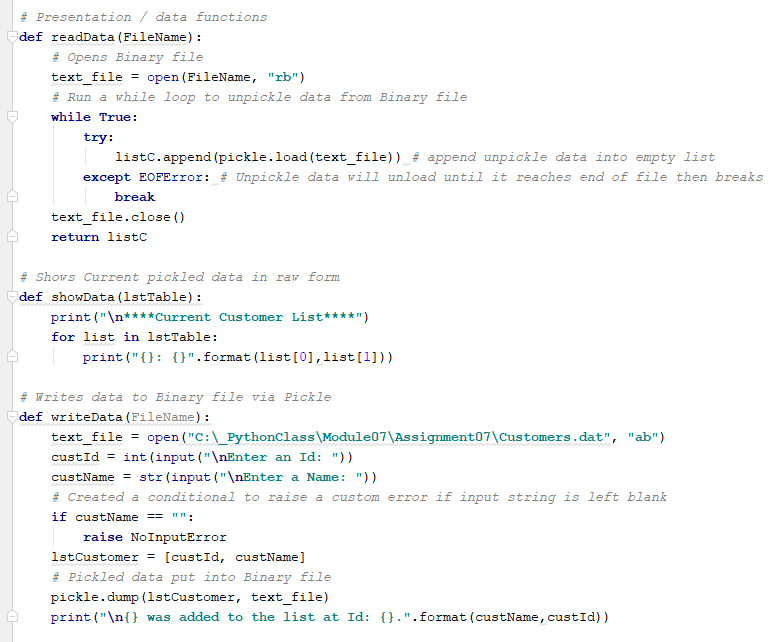
I first started my script by using Professor Roots example from the Programming notes:



***(Professor Randal Root, Programming with Python Chapter 7 notes, 2019)***

I converted his example to my own style of code. I decided to make some changes and added some classes and functions for better practice in python coding. First I imported the pickle function and created an empty list and the text file path. I also commented out another path if you are trying to run this in the command prompt and or Mac. There is usually an issue if you are in a virtual environment with the path address, so I included both. I also set the variable text\_file to None to help with my try and except block.





I took Professor Root’s code and did it in reverse to what he has (and what the other examples have for pickling). I wanted to show the user the list first so they can properly enter the ID number. In unpickling multiple lines of data, you would have to type pickle.load(text file var name) multiple times to get the data out. I was able to find on stackoverflow, a way to get all the lines of data out.



***(stackoverflow***, <https://stackoverflow.com/questions/3871388/reading-from-a-file-using-pickle-and-for-loop-in-python>, ***2019)(External Site)***

I created a function, readData(), to better organize my code and help with the try and except blocks. In my code, a basic while loop helps in getting multiple lines of data. A try and except block is used to help with the unloading of the pickled data. This is how the code reads: **TRY** and append the list data and load it into the empty list, **EXCEPT** if there is an **EOFError** (end of file) **break** out of the loop. Close the data file and return the empty list. Breaking out of the loop when it reaches EOFError ensure that all the data is out of the Binary file.

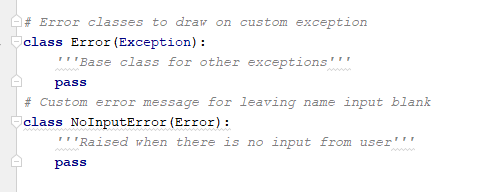
To show the user a better form of the data, I created another function, showData(). The function iterates through the list and displays the data in raw form. This has a better presentation to the user so they can enter the proper ID number.

Now the data conversion into a Binary format is done with the function of writeData(). I open the data file and have the user input and Id number for the customer, which is in an integer format. Then I have them input a Name in a string format. I wanted to proactively capture input errors. I knew I could capture the interger error out of the function, and within the function I set a condition if the user inputted a blank name, with a custom error message (more on the custom error message in the next section). The inputs are then stored in a list and then ***pickled*** to the data file. A message is displayed to the user that the information is in the list.

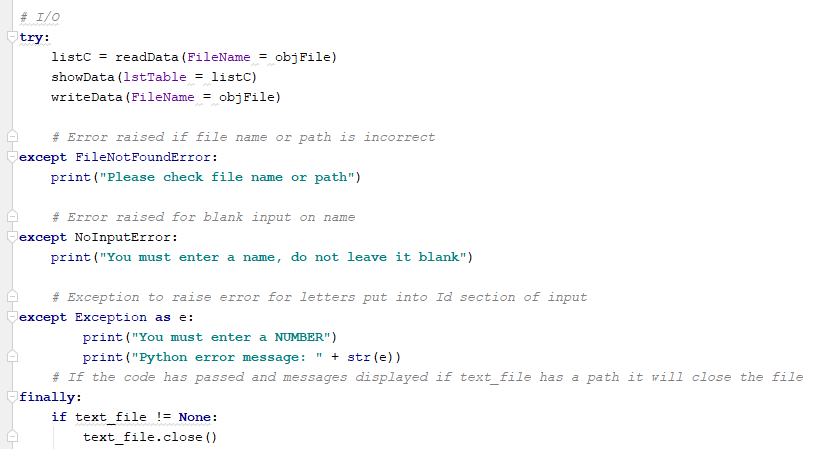
Ok, so here is how I was able to display a custom error message that I found on the web from www.programiz.com:



You have to create a class and call on the Exception function. Then create another class for your error name. Here is how it looks in my script below.

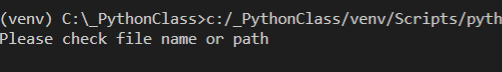


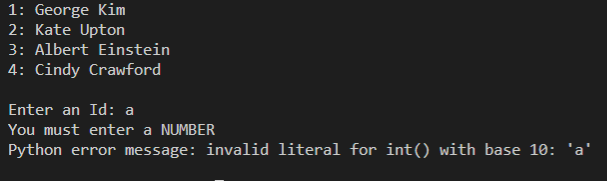
Then I put all the functions together in a try and except block. Here is how it looks:

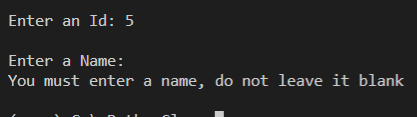


It reads like this: **TRY** to load the list, show the data and write the data from the file path, **EXCEPT** if there is an error with the file path print the **FileNotFoundError** message, also if the is another **error** with the input of the string print the custom error message from the **NoInputError**, also if the the is an **error** with the integer from the ID input, print the message and **Python’s error** message along with it. **FINALLY** if there is a workable file path and everything is correct, close the file.

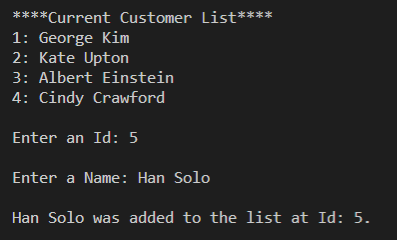
This is how the error messages look like in python output:



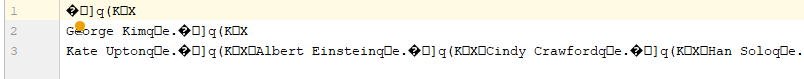




Now here is how the script looks when everything functions properly:



What the binary data looks like:



I hope my explanations and examples helped you understand how the Try and Except work along with pickling.

**Summary**

The Try and Except blocks are a great way to help in capturing errors. There are a variety of messages that can be displayed if there is something wrong with the code, or inputs from the user. This also helps other developers see what errors to expect and how the code was written in how to deal with them. Pickling is another great form of encoding you data if you want to protect the information from the average “Joe”. Yes there are ways to decode that information, but none-the-less, an easy way to protect information. Using a combination of both can make for efficient code in expecting errors and protection information. This was a great project in helping explain how each topic works and gave me a better understanding of them.