Stephanie Miller

November 27, 2022

IT FDN 100

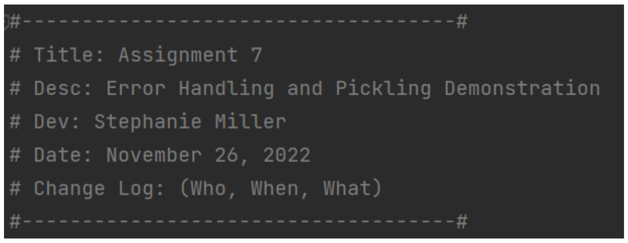
Assignment 7

Git Hub Link:

Assignment 7

Code Header

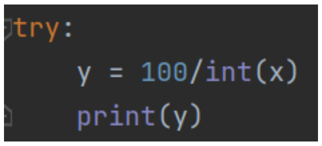
As with every code we much start with the header of the code. In the header we included the title of the script, description of what the code is for, name of the developer, date of the code creation, and a change log for any future changes to the code.



```
#-----------------------------------#
# Title: Assignment 7
# Desc: Error Handling and Pickling Demonstration
# Dev: Stephanie Miller
# Date: November 26, 2022
# Change Log: (Who, When, What)
#-----------------------------------#
```

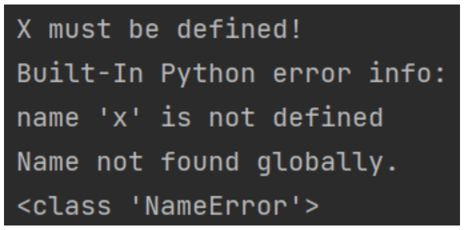
Error Handling

For the error handling demonstration, I decided to focus on specific except clauses. In the try clause I have input a simple equation y = 100/int(x). I decided to turn x into an int in the equation instead of the variable definition for the purpose of the error handling exceptions.



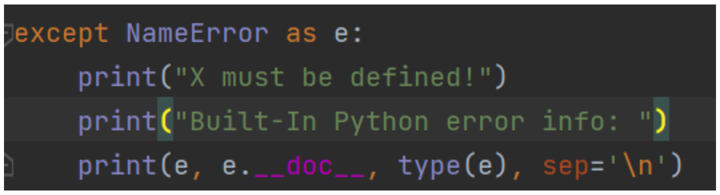
```
try:
    y = 100/int(x)
    print(y)
```

The first except block is defined as NameError. This except block will not be triggered with the current way the code is written because I have not defined the x variable. We see the following when the code is run:



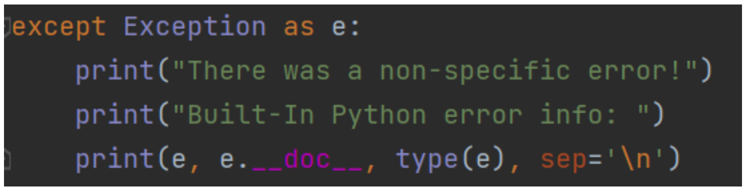
```
X must be defined!
Built-In Python error info:
name 'x' is not defined
Name not found globally.
<class 'NameError'>
```

This is since the x variable can’t be used when we do not set it to a specific value. In this code when the Name Error is triggered, as is the case based on the way we set up the code. The statement will print telling the user that the X variable must be defined. The next statement will then print telling the user what the python defined error is. The code used to display when this error is triggered can be seen below:



```
except NameError as e:
    print("X must be defined!")
    print("Built-In Python error info: ")
    print(e, e.__doc__, type(e), sep='\n')
```

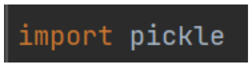
The last except clause in this code is the generalized exception clause that will catch any errors the might be missed. However, with the way the code is currently written the code will always stop at the first error and never make it to the second block in the code.



```
except Exception as e:
    print("There was a non-specific error!")
    print("Built-In Python error info: ")
    print(e, e.__doc__, type(e), sep='\n')
```

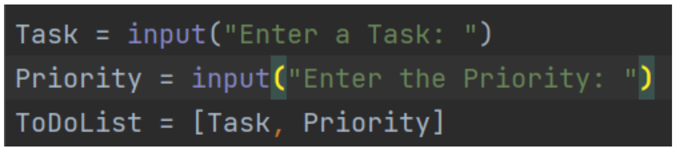
Pickling

The first step when using the pickling is to import pickle. This will allow us to use binary.



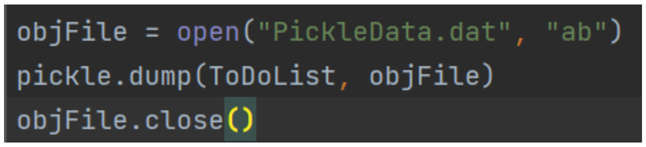
```
import pickle
```

To set up the code I first ask the user to input a task and its priority. I then save the user input to a list that can be referenced later. This will give us some data to work with in the rest of the code.



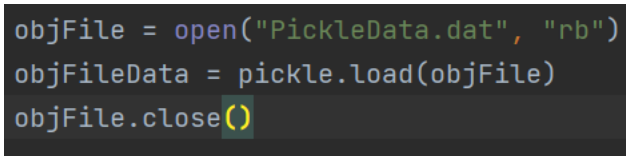
```
Task = input("Enter a Task: ")
Priority = input("Enter the Priority: ")
ToDoList = [Task, Priority]
```

In the next section of code, I open the file I want to save the code to. In this section we need to save the code as .dat so that it will save as binary. We will then open the code in ab which will allow us to append data in binary. In the next part we need to dump the data from the list into the file. Lastly, we will close the file.



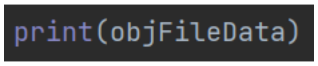
```
objFile = open("PickleData.dat", "ab")
pickle.dump(ToDoList, objFile)
objFile.close()
```

In the next part of the code, I open the file again, this time I open in rb. This will open the binary data in read mode. Next, I need to use pickle load to open the file. We need to do pickle load so the data will be read back in non-binary. Then the file is closed again.



```
objFile = open("PickleData.dat", "rb")
objFileData = pickle.load(objFile)
objFile.close()
```

The last step in the code is to print the file data so we can see what is contained in the file.



```
print(objFileData)
```