**Error Handling Assignment Questions**

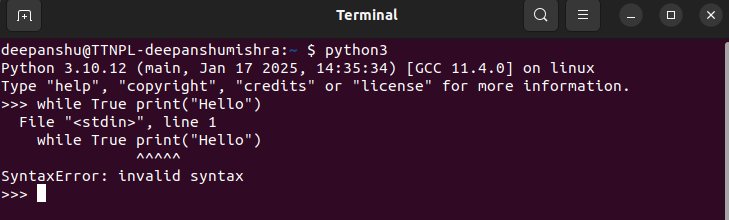
<https://docs.google.com/document/d/1Q51S_wuBfv-ijh9WBb1ALwU1Pk13GhgzaUhtUP_yetk/edit?tab=t.0>

<https://github.com/Deepanshu-TTN/bootcamp-git/tree/master/week-4/Exception%20Handling>

Q1)Define what are following exceptions, when to handle, and handle exceptions, below: SyntaxError Exception RuntimeError ValueError TypeError Warning

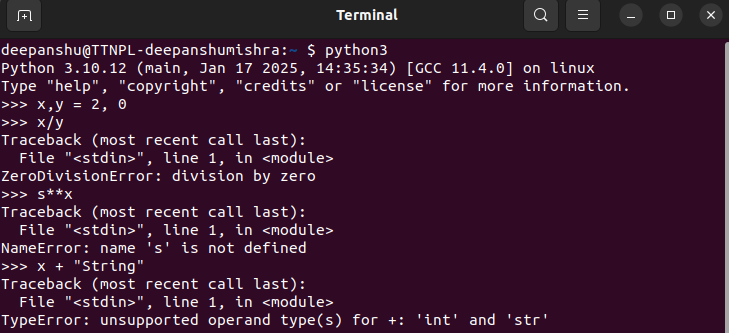
**SyntaxError**: This is the most common kind of error and happens before a program is executed and is caused due to abnormal text present in the executing code. These syntactic errors come to light when the parser is parsing the code and an unexpected order of tokens are received.

When running in the python interpreter you can see where a syntax error occurs after pressing the enter key by the highlighting arrows and the expected exception

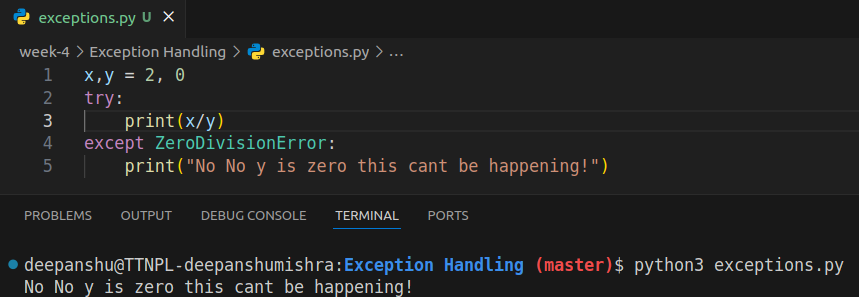


How to Handle - These are common errors which are to be handled while writing the code. Since these errors are checked before runtime, they are not handled with try-except. However, proof reading the code will help in minimizing the risk of any syntax errors.

**Exception**: Exceptions are present in python to prevent errors in the code which are syntactically correct but the logic inside the code is ambiguous, these happen during the runtime and may not be fatal. There are many types of Exceptions in python.



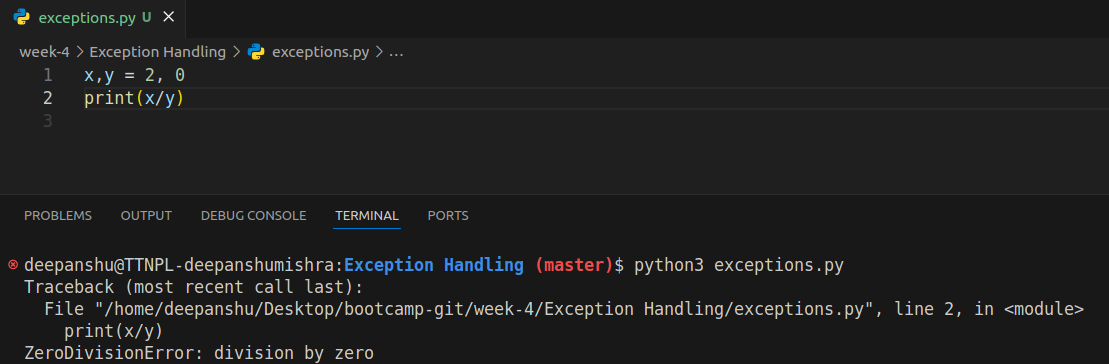
How to handle - We can use try-except block where we put the ambiguous code inside the try block for example where there’s division in the code but the value fetched can be 0 in the variable, and in except block we mention the error that might be raised which is ZeroDivisionError in this case and how to handle that specific exception.



We can also create a custom exception and inherit ‘Exception class’ and we can raise a custom exception and use that class in the except block to control the flow.



Or we can just leave it since there are multiple built-in exceptions that are present in the python library and during runtime, python will automatically raise an exception.

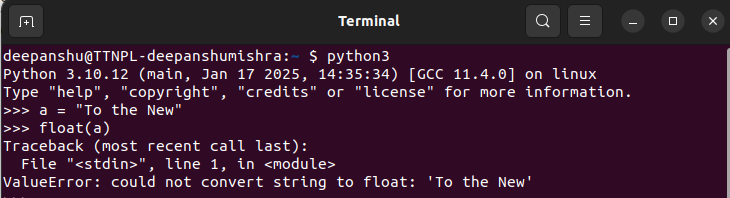


Runtime Errors: These are the errors that arise during the runtime and may need handling. All the errors that happen due to wrong semantics and the python language has a rich variety of exceptions which can arise due to runtime errors and we can write different error handling code using these exceptions to prevent our code from stopping like the divide by zero error in the above example.

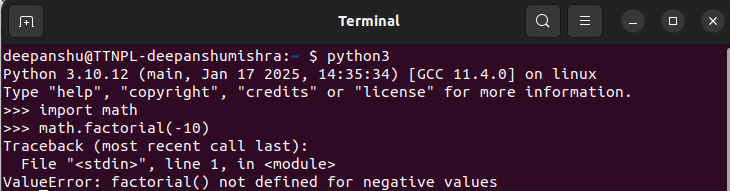
How to handle - Using exceptions

ValueError: These errors often arise when a wrong argument is passed into a function which the function cannot work on. This could be due to invalid arguments of course, wrong usage of the math module in python or when unpacking an iterable object most commonly. Let’s see with examples:

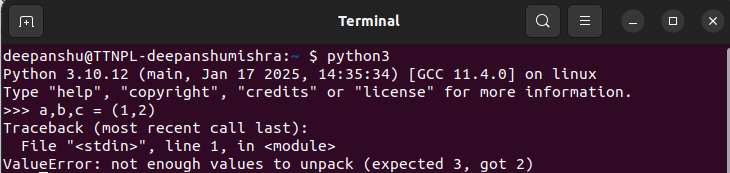
Invalid Arguments:



Incorrect use of the python math module:

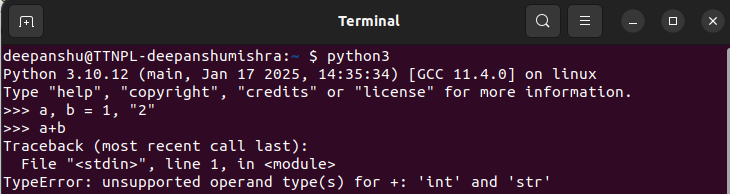


Unpacking Iterables:

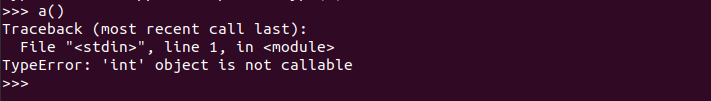


TypeError: Raised whenever an action is performed on an incorrect or unsupported data types. Now this is one of the most common type of error and may arise in a variety of situations

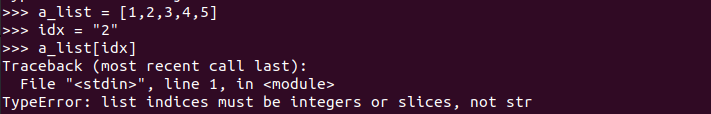
Unsupported Operation between 2 types



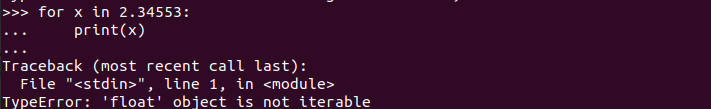
Calling a non callable identifier



Incorrect list index

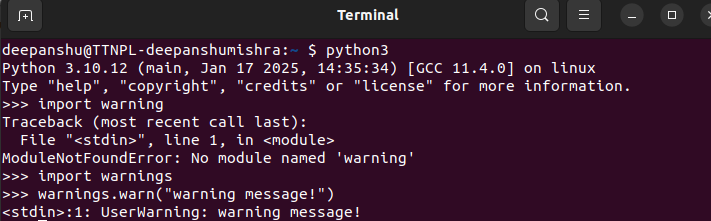


Iterating through a non iterable



How to handle - Through Exceptions

Warning: Warnings are used to warn the developer of situations that aren’t necessarily exceptions. A warning occurs when there are some obsolete programming elements, such as keyword, function or class, etc. It is different from an exception, a program terminates immediately if an error occurs whereas, a warning is not critical. It shows some message, but the program runs. The warn() function defined in warning module is used to show warning messages where the warning module is actually a subclass of the exception class.



Q2)How to define a custom exception? What are the occasion we should define a custom exception? Explain with code

We can define custom exceptions by creating a class which inherits from the built in exception class in python and define the exception behaviour in there. We need to make custom exceptions for 3 major reasons:

Clarity: They provide clear, specific error messages that are relevant to the application which may not already be in the python library or isn’t descriptive enough

Granularity: Allows better error handling as it is easier to pinpoint and address specific issues

Reusability: They can be reused across different parts of the application or even in different projects.

example:

