Assignment (Data Encoding & Process.)

**Q1. Define what are following exceptions, when to handle, and handle exceptions, below:**

**SyntaxError**

**Exception**

**RuntimeError**

**ValueError**

**TypeError**

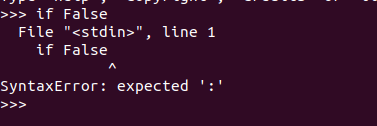
**Warning**

**SyntaxError**

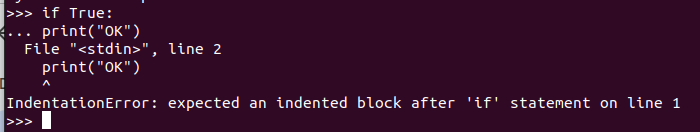
A syntax error is an error that occurs when the code does not follow the syntax rules of the language. Syntax errors are checked by the interpreter or compiler at the time of parsing the code, and they prevent the code from being executed.

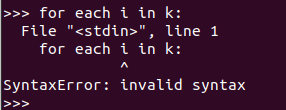
These errors occur because the written code does not conform to the grammatical rules of Python, making it impossible for the interpreter to understand and execute the commands.Some common examples of these errors are:

* Forgetting to put ‘:’ after if,for,else etc..



* Indentation errors, forgetting to provide proper indentation to certain blocks of code may result in syntax error (Indentation error is part of syntax error)



* Providing incorrect syntax to certain keywords in the code  
    
  

**When to Handle & How to handle**

Usually these should be handled immediately when writing the code as these are common errors that happen during writing of the code. Syntax errors are checked before runtime, thus it is not handled with try-except because it occurs before the program starts running. However, careful review of the code will help in minimizing the risk of any syntax errors.

**Exception**

Exceptions are errors or basically an event that disrupts the flow of the normal execution of the program. Exceptions can be caused by different events, some of the built in exceptions are:

**Zero division error**: Occurs when a number is divided by zero

**Type error**: Occurs when an operation is performed on a value of the wrong data type

**Value error**: Occurs when a value is typed cast to the wrong data type

**FileNotFoundError**: Occurs when a file is attempted to be opened without knowing its location

**PermissionError**: Occurs when an attempt is made to write to a locked file

**How to Handle**

Exceptions can be handled multiple ways,

First, we have to use try-except block where we put the main code in the try block or the part of the code where we think exceptions might be raised, and in except block we mention the error that might be raised and how to handle that specific exception.

Second, we can create a custom exception class and inherit ‘Exception class’ and we can basically raise that custom exception and use that class in the except block to control the flow,

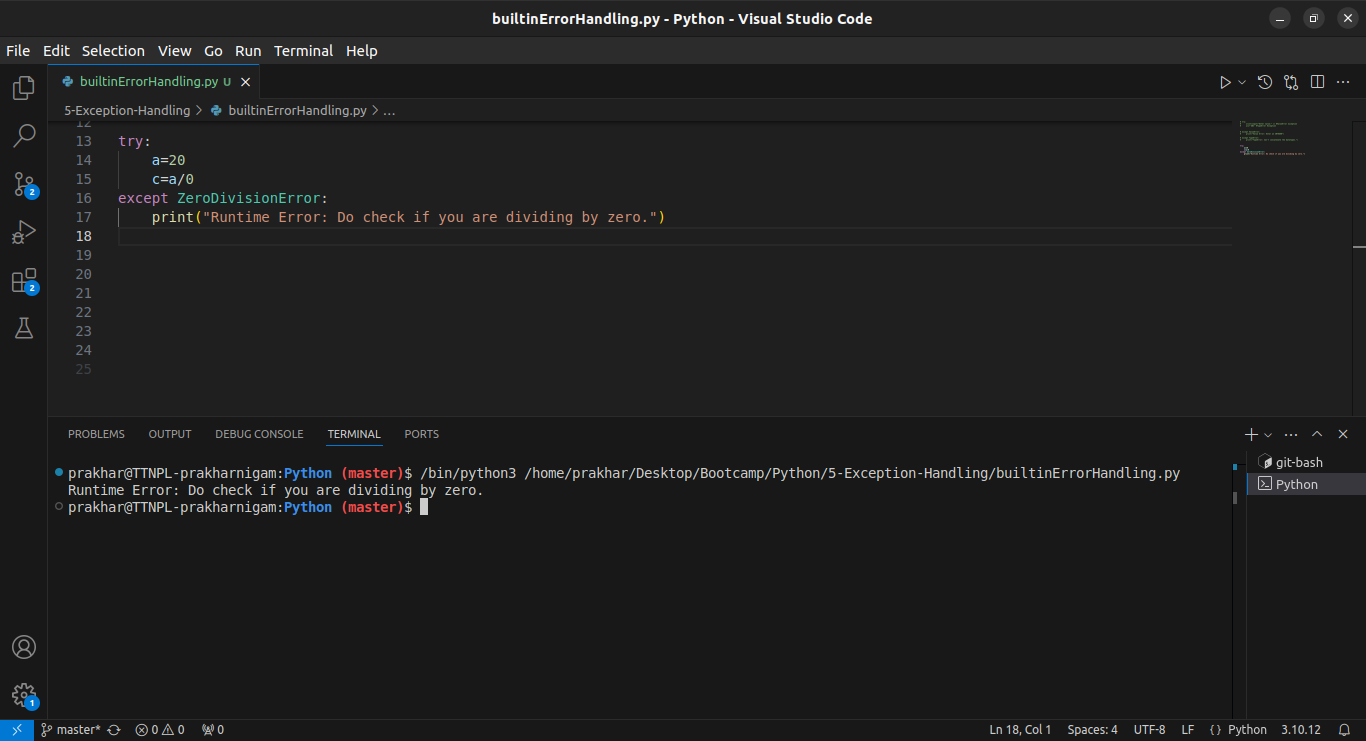
Third, we can basically leave it, there are multiple built-in exceptions that are present in python library and during runtime, python will automatically raise the exception.

**RuntimeError**

The runtimeError are the type of errors that occur during the execution or during the runtime of the program. There are a variety of runtime errors that occur such as logical errors, Input/Output errors, undefined object errors, division by zero errors, and many more.

**How to handle**

Lets take the following example where we are dividing by 0, hence we have to put the code in the try block and create an except (here in this case its ZeroDivisionError, which is a runtime error)

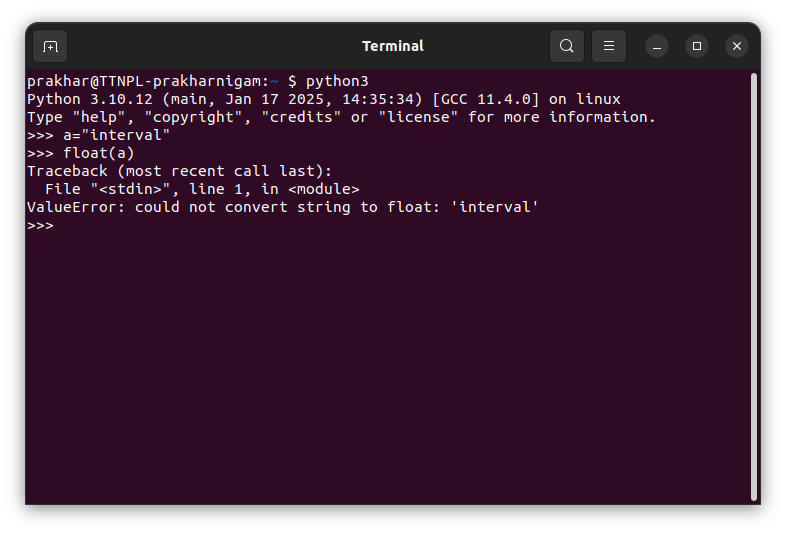


**ValueError**

The ValueError Exception is often raised in Python when an invalid value is assigned to a variable or passed to a function while calling it. It also often occurs during unpacking of sequence data types as well as with functions when a return statement is used. This error usually occurs when:

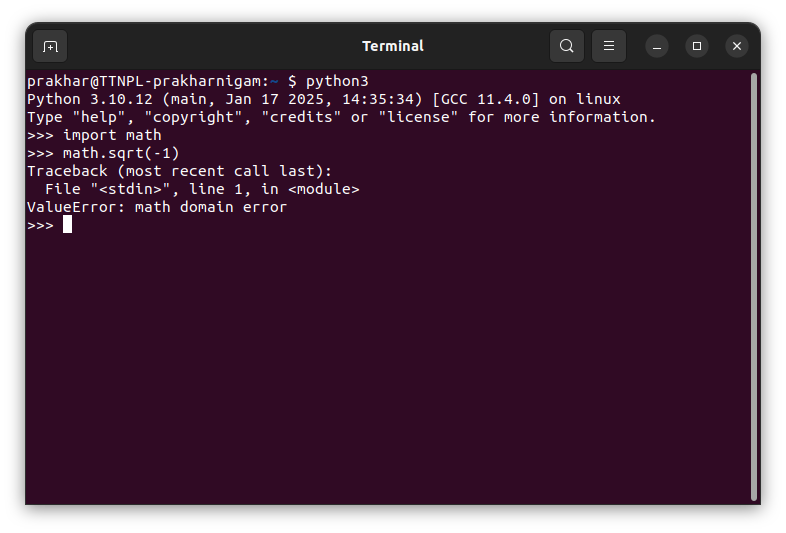
### **Invalid Argument**

A ValueError typically occurs when we pass an invalid argument to a function in Python. As an example, the float() function of Python takes a number and converts it to a float value. But, if we pass a string to this function, it naturally won't be possible for Python to convert a string to a float and thus, it will lead to a ValueError.



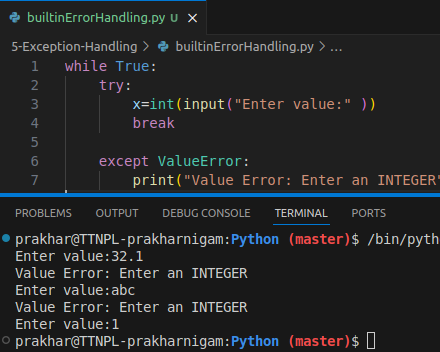
### **Incorrect use of Math Module**

The ValueError exception is raised quite a lot while working with Math module in Python. This is because one might not be aware of the valid arguments for a given function. As an example, the math.sqrt() function return the square root of the number, however if we provide a negative integer. This is not possible as negative numbers cannot have a square root, thus will lead to ValueError



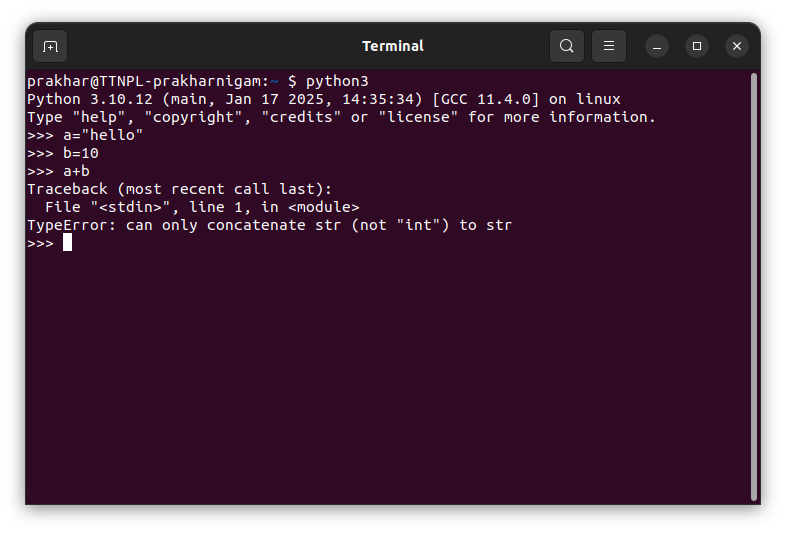
**How to Handle**

We can basically put the part of the code where we think the error might rise in the try block and create an except block for ValueError. Look at the example below

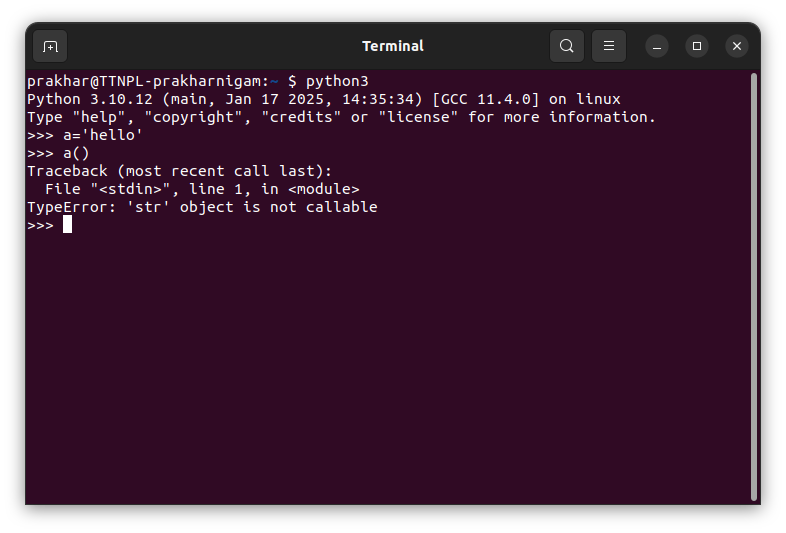


**TypeError**

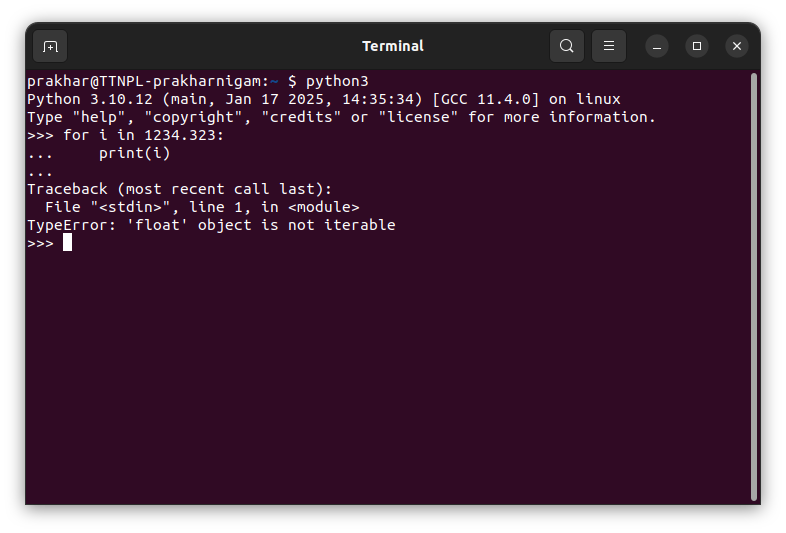
TypeError is one of the common exceptions in python. TypeError is raised whenever an operation is performed on an incorrect/unsupported object type. Example, using the + operator on a string and an integer value will raise a TypeError.



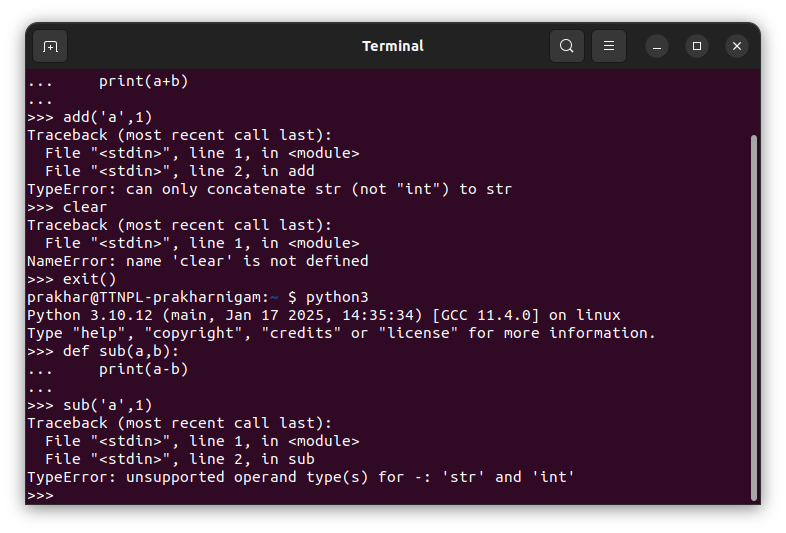
Unsupported Concatenation of datatypes



Calling a non-callable identifier

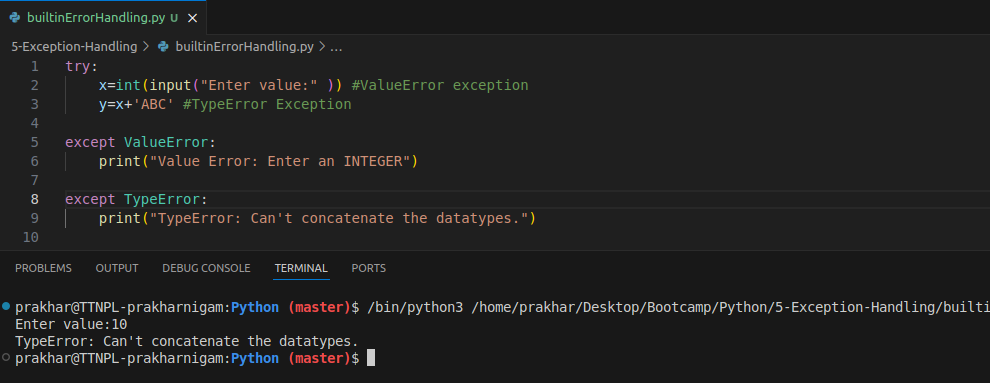


Iterating through a non iterable datatype



Passing wrong type of arguments to function

**How to Handle**  
  
TypeErrors are raised mostly in situations where the programmer fails to check the type of object before performing an operation on them. They can be handled specifically by mentioning them in the except block. In the following example, when one of the indices is found to be an incorrect type, an exception is raised and handled by the program.



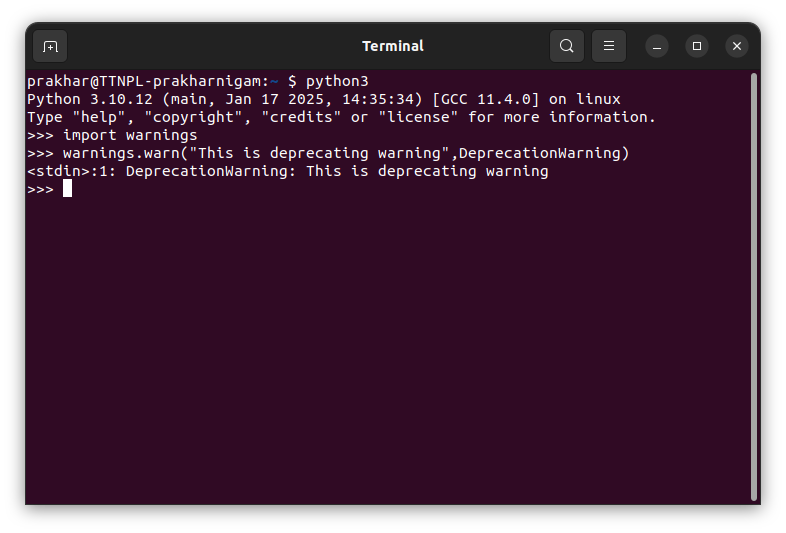
**Warning**

Warning is a special kind of exception in Python. It's not an error, but rather a message indicating something that should be noted. Warnings are usually issued when something in the code is working correctly but might cause issues in the future or be deprecated.

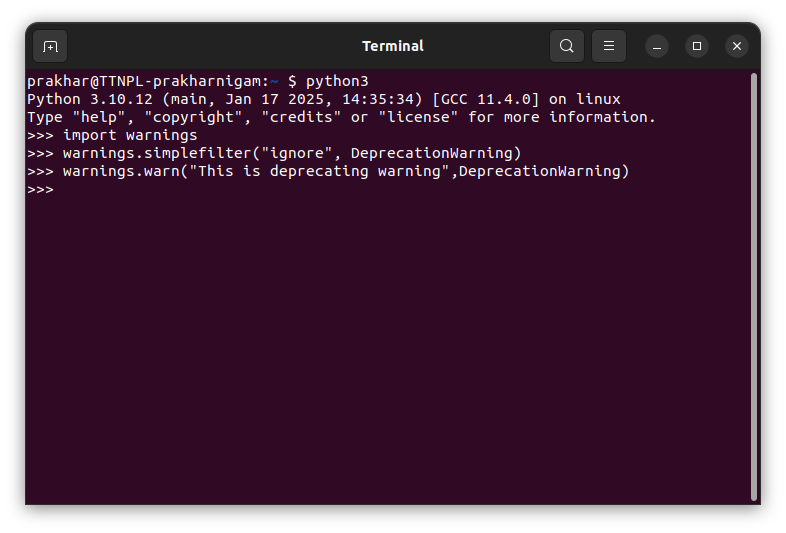
We don't handle warnings with try-except as it's not an error that prevents code execution. However, you can use the warnings module to catch and filter warnings.

**How to use**

We can control or suppress warnings using the warnings module, or you can issue custom warnings.



Here we see the that we can issue custom warnings

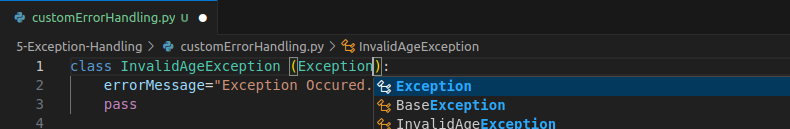


Now after creating a filter list, we see that DeprecationWarnings, which has been added to the filter, doesn't generate a warning.

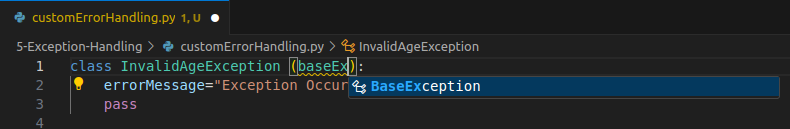
**Q2. How to define a custom exception? What are the occasions we should define a custom exception? Explain with code**

Custom exceptions are basically defined to help us throw a custom exception that we want, ultimately to simplify the code. We usually use the custom exceptions when we want to handle the exceptions which are not properly handled by the builtin exceptions of the python exception library. These exceptions could be specific to tailor each program’s needs depending on the business requirements. Thus, increasing the readability and mentalibility of the code.

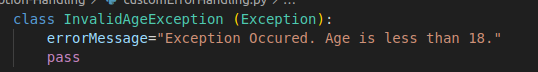
We can create custom exceptions by creating a class and inheriting the Exception class.



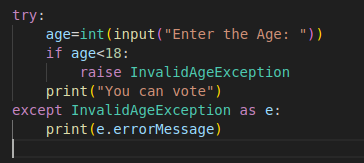
Usually we inherit Exception class however we can also BaseException class which is used majorly for fatal cases



Then if we want, we can define an error message that we want to generate when the exception is raised.



Then in the try block, we explicitly raise the exception class that we defined and in ‘except’ block we call the error message that we defined or we can simply print the error if we want.



Here's the output of the program with the custom exception defined.

