Assertions in Python: An assertion is a sanity-check that you can turn on or turn off when you are done with your testing of the program.

* The easiest way to think of an assertion is to liken it to a raise-if statement (or to be more accurate, a raise-if-not statement). An expression is tested, and if the result comes up false, an exception is raised.
* Assertions are carried out by the assert statement, the newest keyword to Python, introduced in version 1.5.
* Programmers often place assertions at the start of a function to check for valid input, and after a function call to check for valid output.

The assert Statement When it encounters an assert statement, Python evaluates the accompanying expression, which is hopefully true. If the expression is false, Python raises anAssertionError exception.

The syntax for assert is:

assert Expression[, Arguments]

Note: If the assertion fails, Python uses ArgumentExpression as the argument for the AssertionError. AssertionError exceptions can be caught and handled like any other exception, using the try-except statement. If they are not handled, they will terminate the program and produce a traceback.

Example: Here is a function that converts a given temperature from degrees Kelvin to degrees Fahrenheit. Since 0° K is as cold as it gets, the function bails out if it sees a negative temperature

def KelvinToFahrenheit(Temperature):

assert(Temperature >=0),"Colder than absolute zero!"

return((Temperature-273)\*1.8)+32

print(KelvinToFahrenheit(273))

print(int(KelvinToFahrenheit(505.78)))

print(KelvinToFahrenheit(-5))

What is Exception?

An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions. In general, when a Python script encounters a situation that it cannot cope with, it raises an exception. An exception is a Python object that represents an error.

When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.

Handling an Exception If you have some suspicious code that may raise an exception, you can defend your program by placing the suspicious code in a try: block. After the try: block, include an except: statement, followed by a block of code which handles the problem as elegantly as possible.

Syntax: Here is simple syntax of try…..expect….else blocks

try:

You do your operations here

………………………………………………………

except Exception1:

If there is Exception1, then execute this block.

except Exception2:

If there is Exception2, then execute this block.

……………………………………………

else:

If there is no exception then execute this block.

Here are few important points about the above metioned syntax

* A single try statement can have multiple except statements. This is useful when the try block contains statements that may throw different types of exceptions.
* You can provide a generic except clause, which handles any exception.
* After the except clause(s) you can include an else-clause. The code in the else-block executes of the code in the try: block does not raise an exception
* The else-block is a good place for code that does not need to try: block’s protection.

Example: This example opens a file, writes content in the file and comes out gracefully because there is no problem at all.

try:

fh= open("testfile","w")

fh.write("This is my test file for exception handling!")

except IOError:

print("Error: can\'t find file or read data")

else:

print("Written content in the file successfully")

fh.close()

Example: This example tries to open a file where you do not have the write permission, so it raises an exception

try:

fh= open("testfile","r")

fh.write("This is my test file for exception handling!")

except IOError:

print("Error: can't find file or read data")

else:

print("Written content in the file successfully")

fh.close()

The except Clause with No Exceptions

You can also use the except statement with no expections defined as follows

try:

You do your operations here

......................

except:

If there is any exception, then execute this block.

......................

else:

If there is no exception then execute this block.

Note: This kind of a try-except statement catches all the exceptions that occur. Using this kind of try-except statement is not considered a good programming practice though, because it catches all exceptions but does not make the programmer identify the root cause of the problem that may occur

The except clause with Multiple Exceptions:

You can also use the same except statement to handle multiple exceptions as follows

try:

You do your operations here

......................

except(Exception1[, Exception2[,...ExceptionN]]]):

If there is any exception from the given exception list,

then execute this block.

......................

else:

If there is no exception then execute this block.

The try-finally Clause:

You can use a finally: block along with try: The finally: block is a place to put any code that must execute, whether the try-block raised an exception or not. The syntax of the try-finally statement is this.

try:

You do your operations here;

......................

Due to any exception, this may be skipped.

finally:

This would always be executed.

......................

Note: You can provide except clause(s) or finally clause, but not both. You cannot use else clause as well along with a finally clause.

Same example can be written more cleanly as follows

try:

fh=open("testfile","w")

try:

fh.write("This is my test file for exception handling!!")

finally:

print("Going to close the file")

fh.close()

except IOError:

print("Error: can't find file or read data")

When an exception is thrown in the try block , the execution immediately passes to the finally block. After all the statements in the finally block are executed , the exception is raised again and is handled in the except statements if present in the next higher layer of the try-except statement.

Argument of an Exception: An exception can have an argument which is a value that gives additional information about the problem. The contents of the argument vary by exception. You capture an exception’s argument by supplying a variable in the except clause as follows.

try:

You do your operations here

......................

except ExceptionType as Argument:

You can print value of Argument here..

If you write the code to handle a single exception, you can have a variable follow the name of the exception in the except statement. If you are trapping multiple exceptions, you can have a variable follow the tuple of the exception.

This variable receives the value of the exception mostly containing the cause of the exception. The variable can receive a single value or multiple values in the form of tuple. This tuple usually contains the error string, the error number, and an error location

def temp\_convert(var):

try:

returnint(var)

except ValueError as Argument:

print("The argument does not contain numbers\n",Argument)

temp\_convert("xyz")