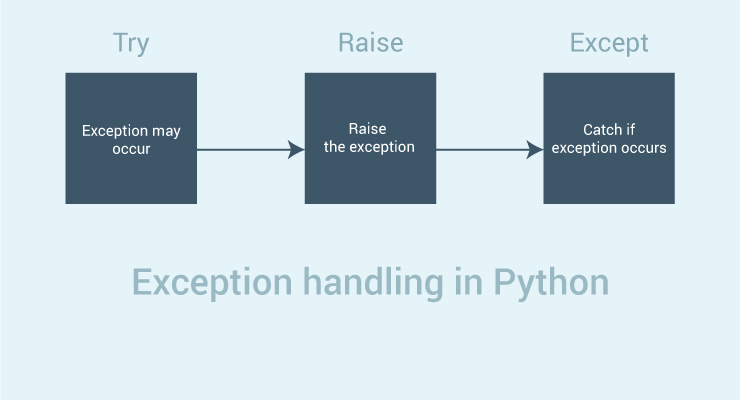
[Programiz Logo](https://www.programiz.com/python-programming" \o "Python Programming)

* [C PROGRAMMING](https://www.programiz.com/c-programming)
* [C++ PROGRAMMING](https://www.programiz.com/cpp-programming)
* [R PROGRAMMING](https://www.programiz.com/r-programming)
* [home](https://www.programiz.com/python-programming)
* [TUTORIAL](https://www.programiz.com/python-programming#learn-python-tutorial)
* [EXAMPLES](https://www.programiz.com/python-programming/examples)
* [METHODS](https://www.programiz.com/python-programming/methods)
* [QUIZ](https://www.programiz.com/python-programming/quiz)

[search](https://www.programiz.com/python-programming/exception-handling)

Python Exception Handling - Try, Except and Finally

**In this article, you'll learn how to handle exceptions in your Python program using try, except and finally statements. This will motivate you to write clean, readable and efficient code in Python.**



Python has many [built-in exceptions](https://www.programiz.com/python-programming/exceptions" \o "Python built-in exceptions) which forces your program to output an error when something in it goes wrong.

When these exceptions occur, it causes the current process to stop and passes it to the calling process until it is handled. If not handled, our program will crash.

For example, if [function](https://www.programiz.com/python-programming/function" \o "Python function) A calls function B which in turn calls function C and an exception occurs in function C. If it is not handled in C, the exception passes to B and then to A.

If never handled, an error message is spit out and our program come to a sudden, unexpected halt.

**Catching Exceptions in Python**

In Python, exceptions can be handled using a try statement.

A critical operation which can raise exception is placed inside the try clause and the code that handles exception is written in except clause.

It is up to us, what operations we perform once we have caught the exception. Here is a simple example.

* [script.py](https://www.programiz.com/python-programming/exception-handling)
* [IPython Shell](https://www.programiz.com/python-programming/exception-handling)



1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

# import module sys to get the type of exception

import sys

randomList = ['a', 0, 2]

for entry in randomList:

try:

print("The entry is", entry)

r = 1/int(entry)

break

except:

print("Oops!",sys.exc\_info()[0],"occured.")

print("Next entry.")

print()

print("The reciprocal of",entry,"is",r)

Run

[Powered by DataCamp](https://www.datacamp.com/)

**Output**

The entry is a

Oops! <class 'ValueError'> occured.

Next entry.

The entry is 0

Oops! <class 'ZeroDivisionError' > occured.

Next entry.

The entry is 2

The reciprocal of 2 is 0.5

In this program, we loop until the user enters an integer that has a valid reciprocal. The portion that can cause exception is placed inside try block.

If no exception occurs, except block is skipped and normal flow continues. But if any exception occurs, it is caught by the except block.

Here, we print the name of the exception using ex\_info() function inside sys module and ask the user to try again. We can see that the values 'a' and '1.3' causes ValueError and '0' causes ZeroDivisionError.

**Catching Specific Exceptions in Python**

In the above example, we did not mention any exception in the except clause.

This is not a good programming practice as it will catch all exceptions and handle every case in the same way. We can specify which exceptions an except clause will catch.

A try clause can have any number of except clause to handle them differently but only one will be executed in case an exception occurs.

We can use a tuple of values to specify multiple exceptions in an except clause. Here is an example pseudo code.

try:

# do something

pass

except ValueError:

# handle ValueError exception

pass

except (TypeError, ZeroDivisionError):

# handle multiple exceptions

# TypeError and ZeroDivisionError

pass

except:

# handle all other exceptions

pass

**Raising Exceptions**

In Python programming, exceptions are raised when corresponding errors occur at run time, but we can forcefully raise it using the keyword raise.

We can also optionally pass in value to the exception to clarify why that exception was raised.

>>> raise KeyboardInterrupt

Traceback (most recent call last):

...

KeyboardInterrupt

>>> raise MemoryError("This is an argument")

Traceback (most recent call last):

...

MemoryError: This is an argument

>>> try:

... a = int(input("Enter a positive integer: "))

... if a <= 0:

... raise ValueError("That is not a positive number!")

... except ValueError as ve:

... print(ve)

...

Enter a positive integer: -2

That is not a positive number!

**try...finally**

The try statement in Python can have an optional finally clause. This clause is executed no matter what, and is generally used to release external resources.

For example, we may be connected to a remote data center through the network or working with a file or working with a Graphical User Interface (GUI).

In all these circumstances, we must clean up the resource once used, whether it was successful or not. These actions (closing a file, GUI or disconnecting from network) are performed in the finally clause to guarantee execution.

Here is an example of [file operations](https://www.programiz.com/python-programming/file-operation" \o "Python file operations) to illustrate this.

try:

f = open("test.txt",encoding = 'utf-8')

# perform file operations

finally:

f.close()

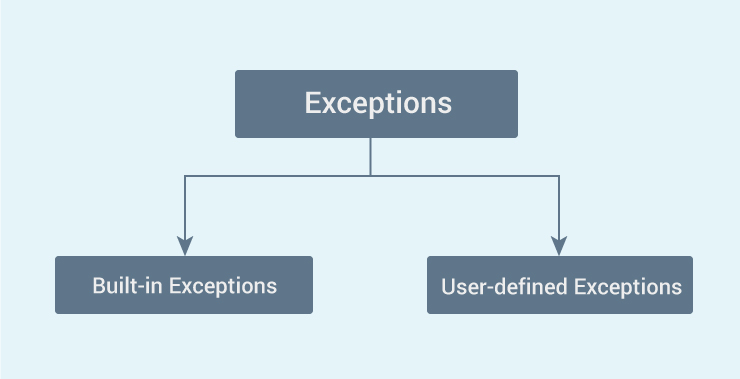
This type of construct makes sure the file is closed even if an exception occurs.

[❮ Previous](https://www.programiz.com/python-programming/exceptions" \o "Python Built-in Exceptions)

[Next Page ❯](https://www.programiz.com/python-programming/user-defined-exception" \o "Python User-Defined Exception)

Python Custom Exceptions

**In this article, you will learn to define custom exceptions depending upon your requirements.**



Python has many [built-in exceptions](https://www.programiz.com/python-programming/exceptions) which forces your program to output an error when something in it goes wrong.

However, sometimes you may need to create custom exceptions that serves your purpose.

In Python, users can define such exceptions by creating a new class. This exception class has to be derived, either directly or indirectly, from Exception class. Most of the built-in exceptions are also derived form this class.

>>> class CustomError(Exception):

... pass

...

>>> raise CustomError

Traceback (most recent call last):

...

\_\_main\_\_.CustomError

>>> raise CustomError("An error occurred")

Traceback (most recent call last):

...

\_\_main\_\_.CustomError: An error occurred

Here, we have created a user-defined exception called CustomError which is derived from the Exception class. This new exception can be raised, like other exceptions, using the raise statement with an optional error message.

When we are developing a large Python program, it is a good practice to place all the user-defined exceptions that our program raises in a separate file. Many standard modules do this. They define their exceptions separately as exceptions.py or errors.py (generally but not always).

User-defined exception class can implement everything a normal class can do, but we generally make them simple and concise. Most implementations declare a custom base class and derive others exception classes from this base class. This concept is made clearer in the following example.

**Example: User-Defined Exception in Python**

In this example, we will illustrate how user-defined exceptions can be used in a program to raise and catch errors.

This program will ask the user to enter a number until they guess a stored number correctly. To help them figure it out, hint is provided whether their guess is greater than or less than the stored number.

# define Python user-defined exceptions

class Error(Exception):

"""Base class for other exceptions"""

pass

class ValueTooSmallError(Error):

"""Raised when the input value is too small"""

pass

class ValueTooLargeError(Error):

"""Raised when the input value is too large"""

pass

# our main program

# user guesses a number until he/she gets it right

# you need to guess this number

number = 10

while True:

try:

i\_num = int(input("Enter a number: "))

if i\_num < number:

raise ValueTooSmallError

elif i\_num > number:

raise ValueTooLargeError

break

except ValueTooSmallError:

print("This value is too small, try again!")

print()

except ValueTooLargeError:

print("This value is too large, try again!")

print()

print("Congratulations! You guessed it correctly.")

Here is a sample run of this program.

Enter a number: 12

This value is too large, try again!

Enter a number: 0

This value is too small, try again!

Enter a number: 8

This value is too small, try again!

Enter a number: 10

Congratulations! You guessed it correctly.

Here, we have defined a base class called Error.

The other two exceptions (ValueTooSmallError and ValueTooLargeError) that are actually raised by our program are derived from this class. This is the standard way to define user-defined exceptions in Python programming, but you are not limited to this way only.