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.

Errors in python program

Logical error

also called **semantic errors**, logical errors cause the program to behave incorrectly, but they do not usually crash the program.
a program with syntax errors, a program with logic errors can be run, but it does not operate as intended.

```
x = float(input('Enter a number: '))

y = float(input('Enter a number: '))

z = x+y/2

print ('The average of the two numbers you have entered is:',z)
```

compile-time error

Syntax errors and indentation errors are the most common compile-time errors for beginners. These errors happen when Python does not understand the code you have written. Many of the errors listed below can be detected by a good Python editor or development interface. Such as,

SyntaxError: invalid syntax
IndentationError

Run time error

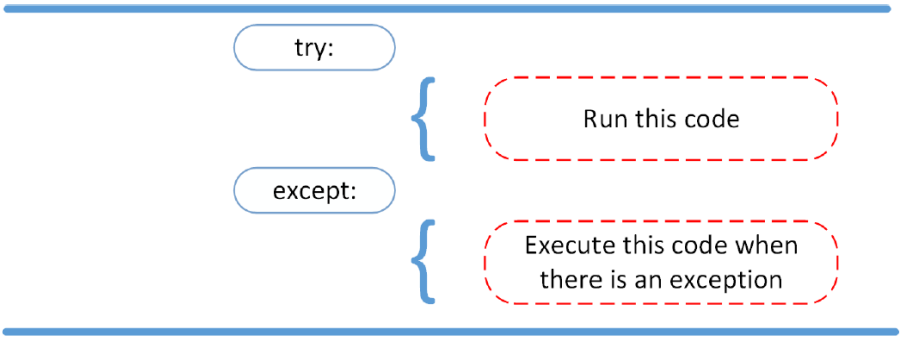
Run-time errors occur when a program executes.
A **run-time error** happens when Python understands what you are saying, but runs into trouble following the given instructions.
Some run-time errors are easy to fix. But others can be the result of incorrect actions that happened earlier in the code and identification may take all your debugging skills.

A program without a runtime error may still have a bug. The program may not print anything, may produce an incorrect result, or may have an infinite loop. Such errors are typically caused by a logic error. Such as,

NameError
IndexError: list index out of range
ZeroDivisionError: integer division or modulo by zero
TypeError
ImportError

Exception Handling

1. The try and except block in Python is used to catch and handle exceptions. Python executes code following the try statement as a "normal" part of the program. The code that follows the except statement is the program's response to any exceptions in the preceding tryclause.



2. As you saw earlier, when syntactically correct code runs into an error, Python will throw an exception error. This exception error will crash the program if it is unhandled. The except clause determines how your program responds to exceptions.

3. the try and except block:

```
def linux_interaction():
        assert ('linux' in sys.platform), "Function can only run on Linux systems."
        print('Doing something.')
```

4. the function a try using the following code:

```
try:
        linux_interaction()
except:
        pass
```

User-define Exception

Python also allows you to create your own exceptions by deriving classes from the standard built-in exceptions.

Here is an example related to *RuntimeError*. Here a class is created that is subclassed from *RuntimeError*. This is useful when you need to display more specific information when an exception is caught.

In the try block, the user-defined exception is raised and caught in the except block. The variable e is used to create an instance of the class Networkerror.

```
class Networkerror(RuntimeError):
        def __init__(self, arg):
                self.args = arg
```

So once you defined above class, you can raise your exception as follows:

```
try:

        Networkerror("Bad hostname")
except Networkerror,e:
        print e.args
```

Types of Exceptions

The table below shows built-in exceptions that are usually raised in Python:

Exception	Description
ArithmeticError	Raised when an error occurs in numeric calculations
AssertionError	Raised when an assert statement fails
AttributeError	Raised when attribute reference or assignment fails
Exception	Base class for all exceptions
EOFError	Raised when the input() method hits an "end of file" condition (EOF)
FloatingPointError	Raised when a floating point calculation fails
GeneratorExit	Raised when a generator is closed (with the close() method)
ImportError	Raised when an imported module does not exist
IndentationError	Raised when indendation is not correct
IndexError	Raised when an index of a sequence does not exist
KeyError	Raised when a key does not exist in a dictionary
KeyboardInterrupt	Raised when the user presses Ctrl+c, Ctrl+z or Delete
LookupError	Raised when errors raised cant be found
MemoryError	Raised when a program runs out of memory
NameError	Raised when a variable does not exist

NotImplementedError	Raised when an abstract method requires an inherited class to override the method
OSError	Raised when a system related operation causes an error
OverflowError	Raised when the result of a numeric calculation is too large
ReferenceError	Raised when a weak reference object does not exist
RuntimeError	Raised when an error occurs that do not belong to any specific expections
StopIteration	Raised when the next() method of an iterator has no further values
SyntaxError	Raised when a syntax error occurs
TabError	Raised when indentation consists of tabs or spaces
SystemError	Raised when a system error occurs
SystemExit	Raised when the sys.exit() function is called
TypeError	Raised when two different types are combined
UnboundLocalError	Raised when a local variable is referenced before assignment
UnicodeError	Raised when a unicode problem occurs
UnicodeEncodeError	Raised when a unicode encoding problem occurs
UnicodeDecodeError	Raised when a unicode decoding problem occurs
UnicodeTranslateError	Raised when a unicode translation problem occurs
ValueError	Raised when there is a wrong value in a specified data type
ZeroDivisionError	Raised when the second operator in a division is zero

Files

A **file** is some information or data which stays in the computer storage devices. **Python** gives you easy ways to manipulate these **files**. Generally we divide **files** in two categories, text **file** and binary **file**. Text **files** are simple text where as the binary **files** contain binary data which is only readable by computer.

Types of files in python

There are two types in python.

1. opening a file
2. closing a file

1. Opening a file

open ( ) will return a file object, so it is most commonly used with two arguments.

An argument is nothing more than a value that has been provided to a function, which is relayed when you call it. So, for instance, if we declare the name of a file as "Test File," that name would be considered an argument.

The syntax to open a file object in Python is:

```
file_object = open("filename", "mode")
```

where ***file_object*** is the variable to add the file object.

Mode

'***r***' – Read mode which is used when the file is only being read

'***w***' – Write mode which is used to edit and write new information to the file (any existing files with the same name will be erased when this mode is activated)

'***a***' – Appending mode, which is used to add new data to the end of the file; that is new information is automatically amended to the end

'***r+***' – Special read and write mode, which is used to handle both actions when working with a file

Example :

```
F = open("workfile","w")
Print f
```

Create a text file

we are going to call it "testfile.txt".

Just create the file and leave it blank.

To manipulate the file, write the following in your Python environment (you can copy and paste if you'd like):

```
file = open("testfile.txt","w")

file.write("Hello World")

file.write("This is our new text file")

file.write("and this is another line.")

file.write("Why? Because we can.")

file.close()
```

Reading a Text File in Python

If you need to extract a string that contains all characters in the file, you can use the following method:

```
file.read()
```

The full code to work with this method will look something like this:

```
file = open("testfile.text", "r")
print file.read()
```

The output of that command will display all the text inside the file, the same text we told the interpreter to add earlier. There's no need to write it all out again, but if you must know, everything will be shown except for the "$ cat testfile.txt" line.

Using the File Write Method

One thing you'll notice about the file write method is that it only requires a single parameter, which is the string you want to be written.

This method is used to add information or content to an existing file. To start a new line after you write data to the file, you can add an EOL character.

```
file = open("testfile.txt", "w")

file.write("This is a test")

file.write("To add more lines.")

file.close()
```

2. Closing a File

When you're done working, you can use the ***fh.close()*** command to end things. What this does is close the file completely, terminating resources in use, in turn freeing them up for the system to deploy elsewhere.

It's important to understand that when you use the ***fh.close()*** method, any further attempts to use the file object will fail.

Opening a text file:

```
fh = open("hello.txt", "r")
```

Reading a text file:

```
Fh = open("hello.txt", "r")

print fh.read()
```

To write new content or text to a file:

```
fh = open("hello.txt", "w")

fh.write("Put the text you want to add here")

fh.write("and more lines if need be.")

fh.close()
```

You can also use this to write multiple lines to a file at once:

```
fh = open("hello.txt","w")

lines_of_text = ["One line of text here", "and another line here", "and yet another here", "and so on and so forth"]

fh.writelines(lines_of_text)

fh.close()
```

To append a file:

```
fh = open("hello.txt", "a")

fh.write("We Meet Again World")

fh.close
```

To close a file completely when you are done:

```
fh = open("hello.txt", "r")

print fh.read()

fh.close()
```

Working with MYSQL Database

Procedure To Follow In Python To Work With MySQL
1. Connect to the database.

2. Create an object for your database.

3. Execute the **SQL** query.

4. Fetch records from the result.

5. Informing the Database if you make any changes in the table.

1. Installing MySQL

MySQL is one of the most popular databases.

Download and install **MySQL** from the **MySQL's** official website. You need to install the **MySQL** server to follow this tutorial.

Next, you have to install **mysql.connector** for **Python**. We need **mysql.connector** to connect **Python Script** to the **MySQL** database. Download the **mysql.connector** from here and install it on your computer.

```
import mysql.connector
```

2. Connecting And Creating

Now, we will connect to the database using **username** and **password** of **MySQL**. If you don't remember your **username** or **password**, create a new **user** with a password.

```
import mysql.connector as mysql
db = mysql.connect(

host = "localhost",
user = "root",
```

```
passwd = "dbms")
print(db)
```

Using MYSQL from Python

MySQL data **insertion**, data **retrieval**, data **update** and data **deletion** From MySQL table using Python.

Insertion data

This section demonstrates how to execute a MySQL INSERT Query from python to add a new row to the database table.

After reading this section you can insert single and multiple rows into MySQL table

You can also learn how to use Python variables in the parameterized query to insert dynamic data into a table.

Retrieval data

This section demonstrates how to execute a SQL SELECT query from Python application to fetch rows from MySQL table.

The syntax for writing the Select query in python.

You can process SELECT query results, Fetch all rows or single row from the table and count total rows of a table.

You will also learn How to use Python variable in the Select Query.

Update data

This section demonstrates how to execute a MySQL UPDATE command from python to update MySQL database and table's data.

After reading this article you will learn how to Update single row, multiple rows, single column, and multiple columns.

Also, you will learn how to use python variables in the parameterized query to update table data.

Delete data

The main purpose of this section is to demonstrate how to use a SQL DELETE statement from your python to delete MySQL *tables and database data.*

After reading this section you are able to Delete single row, multiple rows, single column, and multiple columns Delete all Rows, Delete table and an entire database from MySQL using python.

Retrieving All Rows from a Table

we are fetching all the rows from the python_developers table and copying it into python variables. so we can use it our program.

Example :

```
import mysql.connector
mydb = mysql.connector.connect(
          host="localhost",
          user="yourusername",
          passwd="yourpassword",
          database="mydatabase"  )

mycursor = mydb.cursor()
mycursor.execute("SELECT * FROM customers")
myresult = mycursor.fetchall()
for x in myresult:
          print(x)
mydb.close()
```

import mysql.connector this line imports the MySQL Connector Python module in your program so you can use the functions of this module to communicate with the MySQL database.

Next, We used the **mysql.connector.connect()** function to **connect the MySQL Database from Python**.

Then, We prepared SQL SELECT query to fetch all rows from a python_developers. This table contains four columns.

Next, we used a **mydb.cursor** function to get a cursor object from the connection.

We executed the select operation using a **execute()** function of a Cursor object.

After that, Using a **mycursor.fetchall()** method we can fetch all the records present under a "python_developer" table. On successful execution of a select query, the **execute()** method **returns a ResultSet object which contains all the rows**.

In the end, we used for loop to iterate over all the records present under resultset object and printed them one by one.
Once all record fetched successfully, we closed the MySQL Cursor object using a **cursor.close()** and MySQL database connection using the **mySQLconnection.close()**.

Inserting Rows into a Table

To perform a SQL INSERT query from Python, you just need to follow these simple steps:

First, Establish the MySQL database connection in Python Then, Define the SQL INSERT Query (here you need to know table's column details) Execute the INSERT query. in return, you will get a number of rows affected After successful execution commit your changes to the database Close the MySQL database connection Most important, Catch SQL exceptions if any At last, verify the result by selecting data from MySQL table.

Example :
Insert a record in the "customers" table:

```
import mysql.connector

mydb = mysql.connector.connect(

        host="localhost",

        user="yourusername",

        passwd="yourpassword",

        database="mydatabase"  )

mycursor = mydb.cursor()

sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"

 val = ("John", "Highway 21")

mycursor.execute(sql, val)

mydb.commit()

print(mycursor.rowcount, "record inserted.")
```

Deleting Rows from A Table

To delete rows in a MySQL table from Python, you need to do the following steps:

Connect to the database by creating a new MySQLConnection object.

Instantiate a new cursor object and call its execute() method. To commit the changes, you should always call the commit() method of the MySQLConnection object after calling the execute()method.

Close the cursor and database connection by calling close() method of the corresponding objects.

Example :

Delete any record where the address is "Mountain 21":

```
import mysql.connector

mydb = mysql.connector.connect(

        host="localhost",

        user="yourusername",

        passwd="yourpassword",

        database="mydatabase" )

mycursor = mydb.cursor()

sql = "DELETE FROM customers WHERE address = 'Mountain 21'"

mycursor.execute(sql)

mydb.commit()

print(mycursor.rowcount, "record(s) deleted")
```

Updating Rows from a table

To update data in a MySQL table in Python, you follow the steps below:

Connect to the database by creating a new MySQLConnection object.

Create a new MySQLCursor object from the MySQLConnection object and call the execute()method of the MySQLCursor object. To accept the changes, you call the commit() method of the MySQLConnection object after calling the execute() method. Otherwise, no changes will be made to the database.

Close the cursor and database connection.

Example :

Overwrite the address column from "Valley 345" to "Canyoun 123":

```
import mysql.connector

mydb = mysql.connector.connect(

        host="localhost",
```

```
        user="yourusername",
        passwd="yourpassword",
        database="mydatabase" )
mycursor = mydb.cursor()
sql = "UPDATE customers SET address = 'Canyon 123' WHERE address = 'Valley 345'"
mycursor.execute(sql)
mydb.commit()
print(mycursor.rowcount, "record(s) affected")
```

Creating Database Tables through Python

To create a table in MySQL, use the "CREATE TABLE" statement.

Make sure you define the name of the database when you create the connection

Example :

Create a table named "customers":

```
import mysql.connector
mydb = mysql.connector.connect(
        host="localhost",
        user="yourusername",
        passwd="yourpassword",
        database="mydatabase" )
mycursor = mydb.cursor()
mycursor.execute("CREATE TABLE customers (id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(255), address VARCHAR(255))")
```