BCA SEM-VI

Subject: Python Programming (BCA-601)

Created by: Smit Trivedi (Asst.Prof BCA/PGDCA College Idar)

UNIT 4

Topic:
Exception Handling
Standard Library
&
Python Database connectivity

Exception Handling

Exception handling in Python refers to managing runtime errors that may occur during the execution of a program.

In Python, exceptions are raised when errors or unexpected situations arise during program execution, such as division by zero, trying to access a file that does not exist, or attempting to perform an operation on incompatible data types.

What is Exception?

An event that Occurs during Program execution that disrupts the normal flow of program..

it is a type of error that Occurs at run time.

Types of Exception

Overflow Error

Zerodivision Error

FloatingPoint Error

Key Error

Index Error

File NotFound Error

Class NotFound Error

Memory Error

Component of exception Handling

try : error prone code will be under try block.

except : to handle the exception

else : if there is no exception in code.

Finally: it runs in both condition.

Example of exception:

```
a=input("Enter a Number")
try:
   x=a/5
except TypeError:
   print("invalid Input type")
else:
   print(x)
```

Output: Enter a Number 45
invalid input type

String Formatting

- The Format() Method formats the specified values and insert them inside the string’s placeholder..
- The placeholder is defined using curly brackets{}.
- #named of indexed:
- txt1="Welcome to {fname} {lname}".format(fname="Smit",lname="Trivedi")
- #numberd indexed:
- txt2="Welcome to {0} {1}".format("Smit","Trivedi")
- #empty placeholder:
- txt3="Welcome to {} {}".format("Smit","Trivedi")
- print(txt1)
- print(txt2)
- print(txt3)

- Hence, Python offers following string formatting techniques −
- Using % Operator
- Using Format() Method of str class
- Using f-string
- Using String Template class

File wildcards

- An asterisk (*) matches zero or more characters in a segment of a name. For example, dir/*.

```
import glob
for name in glob.glob('dir/*'):
print name
```

The pattern matches every pathname (file or directory) in the directory dir, without recursing further into subdirectories.

Wildcard Characters:

* → Matches any number of characters (e.g., *.txt matches file.txt, notes.txt).

? → Matches exactly one character (e.g., file?.txt matches file1.txt, fileA.txt but not file12.txt).

[abc] → Matches any single character inside the brackets (e.g., file[12].txt matches file1.txt and file2.txt).

[a-z] → Matches any single character in the given range (e.g., file[a-d].txt matches filea.txt, fileb.txt, etc.).

Command line arguments

- Python Command Line Arguments provides a convenient way to accept some information at the command line while running the program. We usually pass these values along with the name of the Python script.
- Here Python script name is **script.py** and rest of the three arguments - arg1 arg2 arg3 are command line arguments for the program.
- If the program needs to accept input from the user, Python's input() function is used. When the program is executed from command line, user input is accepted from the command terminal.
- Example
- **name = input("Enter your name: ")**
- **print ("Hello {}. How are you?".format(name))**

- The script used input() function to accept user input after the script is run. Let us change it to accept input from command line.
- import sys
- print ('argument list', sys.argv)
- name = sys.argv
- print ("Hello {}. How are you?".format(name))

Python RegEx

- A **Reg**ular **Ex**pression (RegEx) is a sequence of characters that defines a search pattern.
- For example, **^a...s$**

  The above code defines a RegEx pattern. The pattern is: **any five letter string starting with a and ending with s**

  A pattern defined using RegEx can be used to match against a string

Expression	String	Matched?
	abs	No match
	alias	Match
^a...s$	abyss	Match
	Alias	No match
	An abacus	No match

Python has a module named re to work with RegEx.

Python has a built-in package called re , which can be used to work with Regular Expressions. Import the re module: **import re**.

Python and MySQL

Introduction

The Python standard for database interfaces is the Python DB-API. Most Python database interfaces adhere to this standard.

You can choose the right database for your application.

Python Database API supports a wide range of database servers such as

- **GadFly**
- **mSQL**
- **MySQL**
- **PostgreSQL**
- **Microsoft SQL Server 2000**
- **Informix**
- **Interbase**
- **Oracle**
- **Sybase**

Installing MySQL Driver

- First of all, you need to make sure you have already installed python in your machine.
- Verify PIP

  PIP is a package manager in python using which you can install various modules/packages in Python.

Step 1: Open a terminal or command prompt and run

Step **2: pip install mysql-connector-python**

Step 3: pip install pymysql

Verifying the Connector Installation:

```
import mysql. Connector
print("MySQL Connector is installed and working!")
```

Example

- **import mysql.Connector**

```
Conn = mysql.connector.connect(
host="localhost",
user="your_username",
password="your_password",
database="your_database")

cursor = conn.cursor()
cursor.execute("SELECT DATABASE();")
print(cursor.fetchone())

conn.close()
```

Retrieving all rows from a table & Creating a table in MySQL using python

- The method named ***execute()*** (invoked on the cursor object) accepts two variables

1 A String value representing the query to be executed.

2 An optional args parameter which can be a tuple or, list or, dictionary, representing the parameters of the query (values of the place holders).

Then, execute the *CREATE TABLE* statement by passing it as a parameter to the **execute()** method.

example: Create a table Employee in the mydb.

```
import mysql.connector
conn = mysql.connector.connect( user='root', password= ‘ ', host=‘ ', database='mydb' )
cursor = conn.cursor()
```

- #Creating table as per requirement

```
sql ='''CREATE TABLE EMPLOYEE(
FIRST_NAME CHAR(20) NOT NULL,
LAST_NAME CHAR(20),
AGE INT,
INCOME FLOAT
)'''‘
cursor.execute(sql)
#Closing the connection
conn.close()
```

Inserting rows into a table

- You can add new rows to an existing table of MySQL using the **INSERT INTO** statement. In this, you need to specify the name of the table, column names, and values (in the same order as column names).
- **Syntax** : INSERT INTO TABLE_NAME (column1, column2,column3,...columnN) VALUES (value1, value2, value3,...valueN);
- **Example**

  Following query inserts a record into the table named EMPLOYEE.

  INSERT INTO EMPLOYEE(FIRST_NAME, LAST_NAME, AGE, INCOME)

  VALUES (' Smit', ‘Trivedi', 28, 'M', 25000);

```
mysql> select * from Employee;
FIRST_NAME | LAST_NAME | AGE | SEX | INCOME
      Smit|          Trivedi|          28 | M | 25000 |
 1 row in set (0.00 sec)
```

Inserting data in MySQL table using python:

The **execute()** method (invoked on the cursor object) accepts a query as parameter and executes the given query. To insert data, you need to pass the MySQL INSERT statement as a parameter to it.

```
cursor.execute("""INSERT INTO EMPLOYEE(FIRST_NAME,
LAST_NAME, AGE, SEX, INCOME) VALUES (‘Smit', ‘Trivedi', 28,
'M', 25000)""")
```

Example

```
import mysql.connector
conn = mysql.connector.connect(
user='root', password='password', host=‘ ‘ , database='mydb')

    #Creating a cursor object using the cursor() method
     cursor = conn.cursor()
  # Preparing SQL query to INSERT a record into the database.
  sql = """INSERT INTO EMPLOYEE(
   FIRST_NAME, LAST_NAME, AGE, SEX, INCOME)
   VALUES ('Smit', 'Trivedi', 28, 'M', 25000)"""
   try:
 # Executing the SQL command
 cursor.execute(sql)
# Commit your changes in the database
 conn. Commit()
except:
 # Rolling back in case of error
 conn.rollback()
# Closing the connection
conn.close()
```

Updating rows in a table.

UPDATE Operation on any database updates one or more records, which are already available in the database. You can update the values of existing records in MySQL using the UPDATE statement. To update specific rows, you need to use the WHERE clause along with it.

Syntax: UPDATE table_name

SET column1 = value1, column2 = value2...., columnN = valueN WHERE [condition];

mysql> CREATE TABLE EMPLOYEE

(FIRST_NAME CHAR(20) NOT NULL, LAST_NAME CHAR(20), AGE INT, SEX CHAR(1), INCOME FLOAT);

Query OK, 0 rows affected (0.36 sec)

- mysql> UPDATE EMPLOYEE SET AGE = AGE + 1 WHERE SEX = 'M';
- Query OK,

```
import mysql.connector

#establishing the connection
conn = mysql.connector.connect(
user='root', password='password', host='127.0.0.1', database='mydb')

    #Creating a cursor object using the cursor() method
     cursor = conn.cursor()

    #Preparing the query to update the records
      sql = '''UPDATE EMPLOYEE SET AGE = AGE + 1 WHERE SEX = 'M' '''
     try:
    # Execute the SQL command
     cursor.execute(sql)

  # Commit your changes in the database
    conn.commit()
  except:
# Rollback in case there is any error
conn.rollback()

#Retrieving data
sql = '''SELECT * from EMPLOYEE'''

    #Executing the query

    #Displaying the result
print(cursor.fetchall())

    #Closing the connection
conn.close()
```

Deleting rows from table

- To delete records from a MySQL table, you need to use the **DELETE FROM** statement. To remove specific records, you need to use WHERE clause along with it.
- Syntax: DELETE FROM table_name [WHERE Clause]
- mysql> DELETE FROM EMPLOYEE WHERE FIRST_NAME = ‘Smit';
- Query OK, 1 row affected