

Batch: A4 Roll No.: 1911064

Experiment / assignment / tutorial No. 5

Grade: AA / AB / BB / BC / CC / CD / DD

Signature of the Staff In-charge with date

TITLE: Implement the program using the concepts of Exception Handling.

AIM: Implement the program using the concepts of Exception Handling.

Expected OUTCOME of Experiment:

CO2 - Interpret Different Decision Making statements, Functions, Object oriented programming in Python

Books/ Journals/ Websites referred:

1. Core Python Applications Programming by Wesley J Chun
 2. <https://www.tutorialspoint.com/python/index.htm>
-

Pre-Lab/ Prior Concepts:

Exceptions in Python

Python has many built-in exceptions that are raised when your program encounters an error (something in the program goes wrong). When these exceptions occur, the Python interpreter stops the current process and passes it to the calling process until it is handled. If not handled, the program will crash.

For example, let us consider a program where we have a function A that calls function B, which in turn call's function C. If an exception occurs in function C but is not handled in C, the exception passes to B and then to A. If never handled, an error message is displayed and our program comes to a sudden unexpected halt.

Catching Exceptions in Python

In Python, exceptions can be handled using a `try` statement. The critical operation which can raise an exception is placed inside the `try` clause. The code that handles the exceptions is written in the `except` clause.

Catching Specific Exceptions in Python

In the above example, we did not mention any specific 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 should catch.

A `try` clause can have any number of `except` clauses to handle different exceptions, however, only one will be executed in case an exception occurs.

Raising Exceptions in Python

In Python programming, exceptions are raised when errors occur at runtime. We can also manually raise exceptions using the `raise` keyword.

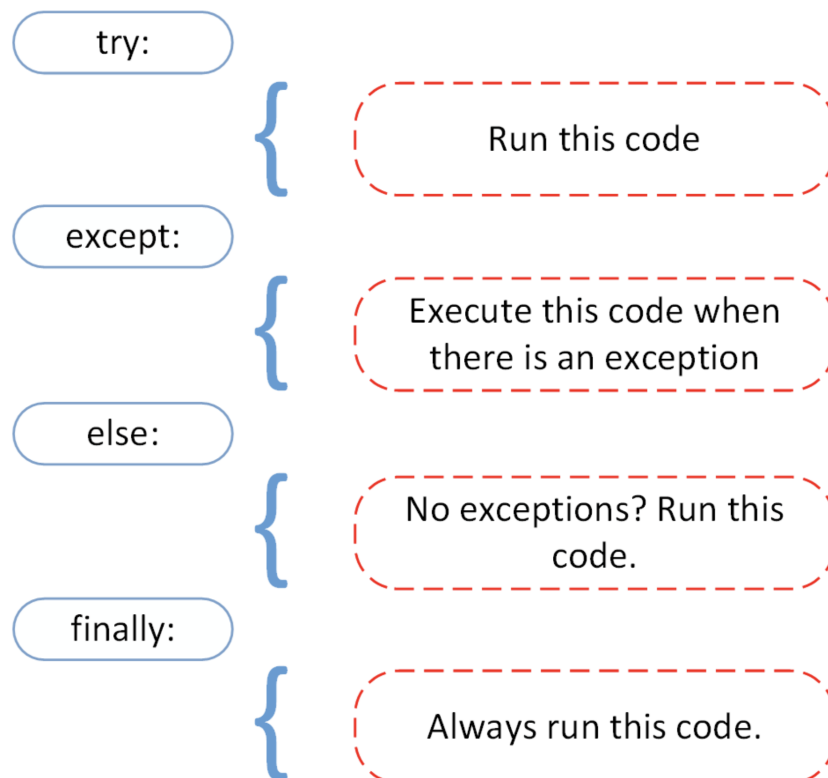
Python try with else clause

In some situations, you might want to run a certain block of code if the code block inside `try` ran without any errors. For these cases, you can use the optional `else` keyword with the `try` statement.

Python 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 a Graphical User Interface (GUI).

In all these circumstances, we must clean up the resource before the program comes to a halt whether it successfully ran or not. These actions (closing a file, GUI or disconnecting from network) are performed in the `finally` clause to guarantee the execution.



Problem Statement:

Write a Program that implements a Queue data structure of specified size. if the queue becomes full and we still try to add an element to it than a user-defined Queue-Error exception should be raised. similarly, if the queue is empty and we try to delete an element from it then a Queue-Error exception should be raised.

Program: (Separate Code File)

```
class QueueFull(Exception):
    def __init__(self, message="Queue is Full, You
cannot add more elements\n"):
        self.message = message
        super().__init__(self.message)

class QueueEmpty(Exception):
    def __init__(self, message="Queue is Empty, You
cannot delete more elements\n"):
        self.message = message
        super().__init__(self.message)

q=[]

def Enqueue():
    try:
        if len(q)==max_size:
            raise QueueFull
        except QueueFull as err:
            print(err)
        else:
            element=input("Enter the element:")
            q.append(element)
            print(element,"is added to the Queue!")

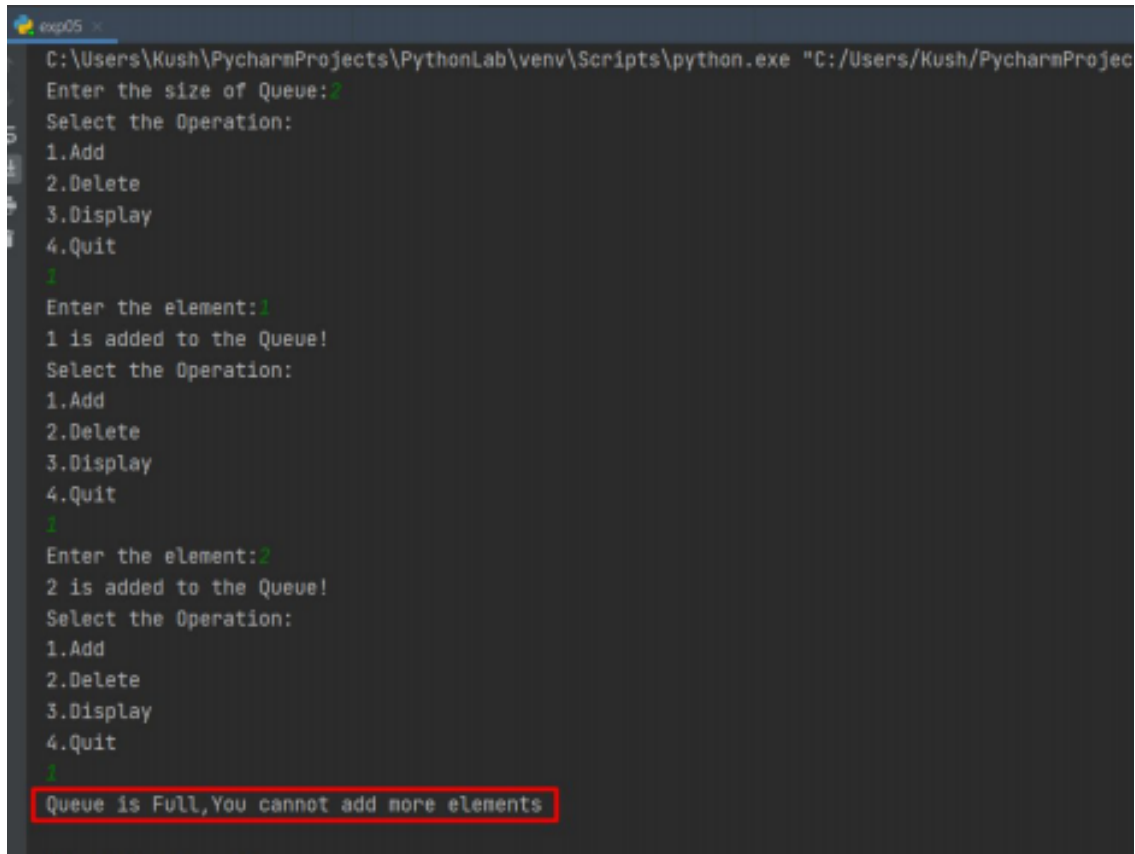
def dequeue():
    try:
        if not q:
            raise QueueEmpty
        except QueueEmpty as err:
            print(err)
        else:
```

```
e=q.pop(0)
print("element removed!!:",e)

def display():
    print(q)
max_size=int(input("Enter the size of Queue:"))
while True:
    print("Select the Operation:\n1.Add \n2.Delete\n3.Display \n4.Quit")
    choice=int(input())
    if choice==1:
        Enqueue()
    elif choice==2:
        dequeue()
    elif choice==3:
        display()
    elif choice==4:
        break
    else:
        print("Invalid Option!!!")
```

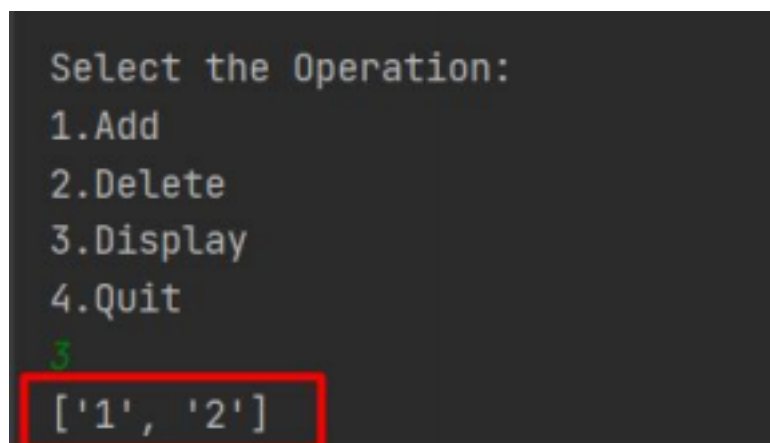
Output (Screenshots)

1) Inserting elements to the queue and raising error if OVERFLOW



```
exp05 > C:\Users\Kush\PycharmProjects\PythonLab\venv\Scripts\python.exe "C:/Users/Kush/PycharmProjec
Enter the size of Queue:2
Select the Operation:
1.Add
2.Delete
3.Display
4.Quit
1
Enter the element:1
1 is added to the Queue!
Select the Operation:
1.Add
2.Delete
3.Display
4.Quit
1
Enter the element:2
2 is added to the Queue!
Select the Operation:
1.Add
2.Delete
3.Display
4.Quit
1
Queue is Full, You cannot add more elements
```

2) Displaying elements of the queue



```
Select the Operation:
1.Add
2.Delete
3.Display
4.Quit
3
['1', '2']
```

3) Deleting elements from the queue and raising error if underflow

```
Select the Operation:
1.Add
2.Delete
3.Display
4.Quit
2
element removed!!: 1
Select the Operation:
1.Add
2.Delete
3.Display
4.Quit
2
element removed!!: 2
Select the Operation:
1.Add
2.Delete
3.Display
4.Quit
2
Queue is Empty,You cannot delete more elements
```

Conclusion :

Hence we have successfully understood about the various concepts of exception handling and implemented the queue data structure where on adding an element to a completely filled queue and removing an element from an empty queue throws an error which is handled using try and except.

Date: 17/03/2021

Signature of faculty in-charge

Post Lab Descriptive Questions:

- 1) When is the finally block executed?
 - a) when there is no exception
 - b) when there is an exception
 - c) only if some condition that has been specified is satisfied
 - d) always

Ans ☐ d) always

2. What will be the output of the following Python code?

```
try:
    if '1' != 1:
        raise "someError"
    else:
        print("someError has not occurred")
except "someError":
    print ("someError has occurred")
```

- a) someError has occurred
- b) someError has not occurred
- c) invalid code
- d) none of the mentioned

Ans ☐ **Opt** ☒ **C**

Explanation: A new exception class must inherit from a BaseException. There is no such inheritance here.

3. Explain raise function in python

→

The raise statement allows the programmer to force a specified exception to occur. The sole argument to raise indicates the exception to be raised. This must be either an exception instance or an exception class (a class that derives from Exception). If an exception class is passed, it will be implicitly instantiated by calling its constructor with no arguments: