K. J. Somaiya College of Engineering, Mumbai-77

Department of Computer Engineering

Roll No.: 1911027

Experiment No. 5

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

Signature of the Staff In-charge with

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, then a user-defined **QueueError** exception should be raised. Similarly, if the queue is empty and we try to delete an element from it then a **QueueError** exception should be raised .

AIM: Use concepts of python errors and exceptions

Expected OUTCOME of Experiment:

CO1: Describe the Numbers, Math functions, Strings, List, Tuples and Dictionaries in Python.

CO2: Interpret different Decision Making statements, Functions, Object oriented programming in python.

Books/ Journals/ Websites referred:

1)https://www.geeksforgeeks.org/errors-and-exceptions-in-python
2)https://www.programiz.com/python-programming/exceptions
3)https://geek-university.com/python/types-of-errors
4)https://python-textbok.readthedocs.io
5)https://www.programiz.com/python-programming/exception-handling

---

Pre Lab/ Prior Concepts:

Errors:

Errors are the problems in a program due to which the program will stop the execution. We can make certain mistakes while writing a program that lead to errors when we try to run it. A python program terminates as soon as it encounters an unhandled error.

Python errors are classified into following classes:

1. Compile time errors

2. Run time errors

3. Logical errors

Compile time errors: Errors that occur when you ask Python to run the application. Before the program can be run, the source code must be compiled into the machine code. If the conversion cannot performed, Python will inform you that your application can not be run before the error is fixed.

Common Python compile time errors include:

- leaving out a keyword
- putting a keyword in the wrong place
- leaving out a symbol, such as a colon, comma or brackets
- misspelling a keyword
- incorrect indentation
- empty block

The most common errors of this type are syntax errors – for example, if you don't end an if statement with the colon. Here is an example:

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


if x%2 == 0
    print('You have entered an even number.')
else:
```

```
    print ('You have entered an odd number.')
```

The code above checks if the number the user enters is an odd or an even number. However, notice how the if statement is missing the colon (:) at the end of the line. Because of it, the program won't run and the interpreter will even inform use what the problem is:

```
File "file.py", line 3
if x%2 == 0
^
SyntaxError: invalid syntax
```

Runtime errors: Errors that occur after the code has been compiled and the program is running. The error of this type will cause your program to behave unexpectedly or even crash. If a program is syntactically correct – that is, free of syntax errors – it will be run by the Python interpreter. However, the program may exit unexpectedly during execution if it encounters a runtime error – a problem which was not detected when the program was parsed, but is only revealed when a particular line is executed. When a program comes to a halt because of a runtime error, we say that it has crashed. Runtime errors often creep in if you don't consider all possible values that a variable could contain, especially when you are processing user input. You should always try to add checks to your code to make sure that it can deal with bad input and edge cases gracefully.

Some examples of Python runtime errors:

- division by zero
- performing an operation on incompatible types

- using an identifier which has not been defined
- accessing a list element, dictionary value or object attribute which doesn't exist
- trying to access a file which doesn't exist

An example of a runtime error is the division by zero. Consider the following example:

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

print (x,'divided by',y,'equals: ',z)
```

The program above runs fine until the user enters 0 as the second number:

```
>>>
Enter a number: 9
Enter a number: 2
9.0 divided by 2.0 equals: 4.5
>>>
Enter a number: 11
Enter a number: 3
11.0 divided by 3.0 equals: 3.6666666666666665
>>>
Enter a number: 5
Enter a number: 0
Traceback (most recent call last):
```

```
File "C:/file.py", line 3, in <module>
z = x/y
ZeroDivisionError: float division by zero
>>>
```

Logical errors: Logical errors are the most difficult to fix. They occur when the program runs without crashing, but produces an incorrect result. The error is caused by a mistake in the program's logic. You won't get an error message, because no syntax or runtime error has occurred. You will have to find the problem on your own by reviewing all the relevant parts of your code – although some tools can flag suspicious code which looks like it could cause unexpected behaviour. Sometimes there can be absolutely nothing wrong with your Python implementation of an algorithm – the algorithm itself can be incorrect. However, more frequently these kinds of errors are caused by programmer carelessness. Here are some examples of mistakes which lead to logical errors:

- using the wrong variable name
- indenting a block to the wrong level
- using integer division instead of floating-point division
- getting operator precedence wrong
- making a mistake in a boolean expression
- off-by-one, and other numerical errors

E.g.:

```
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)
```

The example above should calculate the average of the two numbers the user enters. But, because of the order of operations in arithmetic (the division is evaluated before addition) the program will not give the right answer:

```
>>>
Enter a number: 3
Enter a number: 4
The average of the two numbers you have entered is:
5.0
>>>
```

To rectify this problem, we will simply add the parentheses: z = (x+y)/2.

Exceptions:

There are some situations in which runtime errors are likely to occur. Whenever we try to read a file or get input from a user, there is a chance that something unexpected will happen – the file may have been moved or deleted, and the user may enter data which is not in the right format. Good programmers should add safeguards to their programs so that common

situations like this can be handled gracefully – a program which crashes whenever it encounters an easily foreseeable problem is not very pleasant to use. Most users expect programs to be robust enough to recover from these kinds of setbacks. If we know that a particular section of our program is likely to cause an error, we can tell Python what to do if it does happen. Instead of letting the error crash our program we can intercept it, do something about it, and allow the program to continue. All the runtime (and syntax) errors that we have encountered are called exceptions in Python – Python uses them to indicate that something exceptional has occurred, and that your program cannot continue unless it is handled.

Built-in exceptions in python:

Exception	Cause of Error
`AssertionError`	Raised when an `assert` statement fails.
`AttributeError`	Raised when attribute assignment or reference fails.
`EOFError`	Raised when the `input()` function hits end-of-file condition.
`FloatingPointError`	Raised when a floating point operation fails.
`GeneratorExit`	Raise when a generator's `close()` method is called.
`ImportError`	Raised when the imported module is not found.
`IndexError`	Raised when the index of a sequence is out of range.
`KeyError`	Raised when a key is not found in a dictionary.
`KeyboardInterrupt`	Raised when the user hits the interrupt key (`Ctrl+C` or `Delete`).
`MemoryError`	Raised when an operation runs out of memory.
`NameError`	Raised when a variable is not found in local or global scope.
`NotImplementedError`	Raised by abstract methods.

`OSError`	Raised when system operation causes system related error.
`OverflowError`	Raised when the result of an arithmetic operation is too large to be represented.
`ReferenceError`	Raised when a weak reference proxy is used to access a garbage collected referent.
`RuntimeError`	Raised when an error does not fall under any other category.
`StopIteration`	Raised by `next()` function to indicate that there is no further item to be returned by iterator.
`SyntaxError`	Raised by parser when syntax error is encountered.
`IndentationError`	Raised when there is incorrect indentation.
`TabError`	Raised when indentation consists of inconsistent tabs and spaces.
`SystemError`	Raised when interpreter detects internal error.
`SystemExit`	Raised by `sys.exit()` function.
`TypeError`	Raised when a function or operation is applied to an object of incorrect type.
`UnboundLocalError`	Raised when a reference is made to a local variable in a function or method, but no value has been bound to that variable.
`UnicodeError`	Raised when a Unicode-related encoding or decoding error occurs.
`UnicodeEncodeError`	Raised when a Unicode-related error occurs during encoding.
`UnicodeDecodeError`	Raised when a Unicode-related error occurs during decoding.
`UnicodeTranslateError`	Raised when a Unicode-related error occurs during translating.
`ValueError`	Raised when a function gets an argument of correct type but improper value.
`ZeroDivisionError`	Raised when the second operand of division or modulo operation is zero.

Exception Handing:

How exceptions are handles in python: When an exception occurs, the normal flow of execution is interrupted. Python checks to see if the line of code which caused the exception is inside a try block. If it is, it checks to see if any of the except blocks associated with the try block can handle that type of exception. If an appropriate handler is found, the exception is handled, and the program continues from the next statement after the end of that try-except. If there is no such handler, or if the line of code was not in a try block, Python will go up one level of scope: if the line of code which caused the exception was inside a function, that function will exit immediately, and the line which called the function will be treated as if it had thrown the exception. Python will check if that line is inside a try block, and so on. When a function is called, it is placed on Python's stack. Python traverses this stack when it tries to handle an exception. If an exception is thrown by a line which is in the main body of your program, not inside a function, the program will terminate. When the exception message is printed, you should also see a traceback – a list which shows the path the exception has taken, all the way back to the original line which caused the error.

Try-Except block: 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. We can thus choose what operations to perform once we have caught the exception.

Syntax:

try:

Statements that may cause exceptions

except ErrorClass:

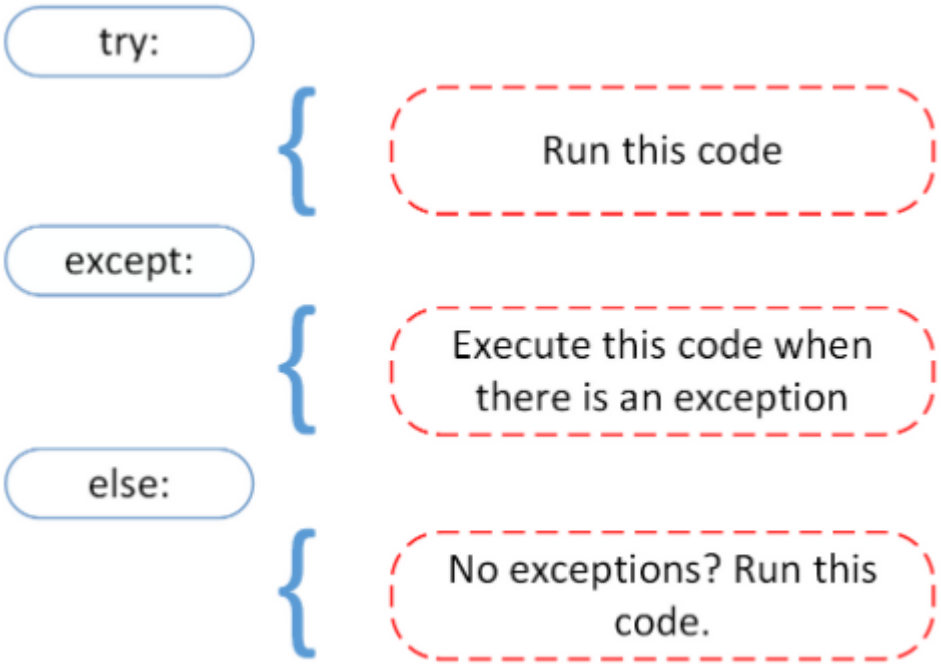
Statements that handles exception

Here is an example:

```
try:
    a=10/0
except ZeroDivisionError:
    print("Cant Divide")
```

```
Cant Divide
```

Else clause: In Python, using the else statement, you can instruct a program to execute a certain block of code only in the absence of exceptions. Here is the block structure

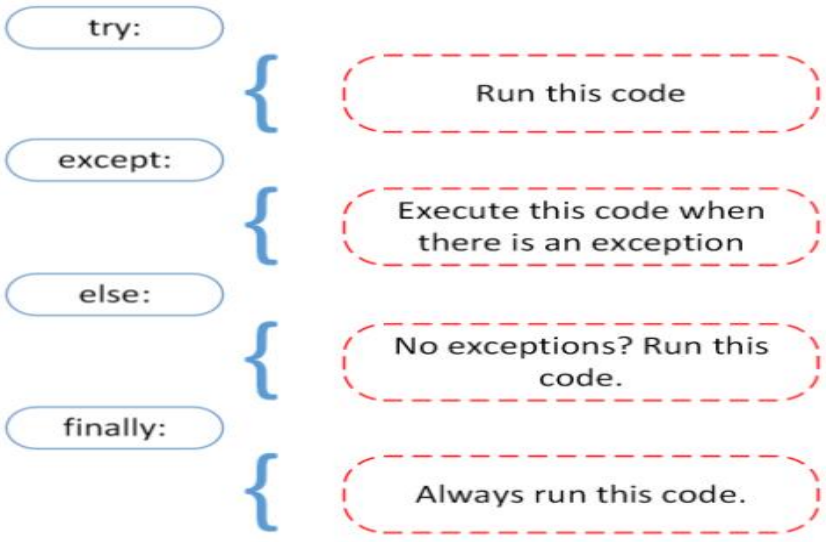


E.g.: As seen in below code inside try block no exception occurs so try block will execute and except will not execute as exception doesn't occurs so after that directly else block will execute.

```
try:
    a=10/2
except ZeroDivisionError:
    print("Cant Divide")
else:
    print("No exception occurs")
```

```
No exception occurs
```

Finally clause: Imagine that you always had to implement some sort of action to clean up after executing your code. Python enables you to do so using the finally clause. Here is the block structure



E.g.: As you can see in the below code exception occurs in try block so control will go to except block and that block will be executed as finally block will always executed regardless of exception occurs or not so after that finally block will execute.

```
try:
    a=10/0
except ZeroDivisionError:
    print("Cant Divide")
finally:
    print("Always will be executed")
```

```
Cant Divide
Always will be executed
```

Important points: 1) We can write a try block without any except blocks.

2) Finally block is always executed.

3) Multiple except blocks can be used to handle multiple exceptions.

4) Else blocks and finally blocks are not compulsary.

5) When there is no exception, else block is executed after try block.

6) A single try block can be followed by several except blocks.

7) We can not write except blocks without a try block.

Program:

```
class Error(Exception):
    """Base class for other exceptions"""
    pass


class QueueFull(Error):
    """This is to be raised if queue is full and there is no space for other element to insert."""
    pass


class QueueEmpty(Error):
    """This is to be raised if queue is empty and there is no element to be deleted or to be displayed."""
    pass


class Queue():
    def __init__(self):
        self.queue=[]
        self.front=-1
        self.rear=-1
```

```
    def enqueue(self,ele):
        try:
            if(self.rear==-1 and self.front==-1):
                self.rear=0
                self.front=0
                self.queue[self.rear]=ele;
                print("Element ",ele," added successfully!!!!!")
                print("----------------------------------------------------------------------
----------------")
            elif(self.rear==n-1):
                raise QueueFull
            else:
                self.rear=self.rear+1
                self.queue[self.rear]=ele
                print("Element ",ele," added successfully!!!!!")
                print("----------------------------------------------------------------------
----------------")
        except QueueFull:
```

```
        print("QueueFull Error : There is no space in the queue for the element queue is full!!!!!!")
        print("-------------------------------------------------------------------------------")

    def dequeue(self):
        try:
            if(self.front==-1 and self.rear==-1):
                raise QueueEmpty
            elif(self.front==self.rear):
                temp=self.queue[self.front]
                self.front=-1
                self.rear=-1
                return temp,True
            else:
                temp=self.queue[self.front]
                self.front=self.front+1
                return temp,True
        except QueueEmpty:
            print("QueueEmpty Error : Queue is empty nothing to delete!!!!!!")
```

```
        print("-------------------------------------------------------------------
-------------")
        return None,False


    def display(self):
        try:
            if(self.front==-1 and self.rear==-1):
                raise QueueEmpty
            else:
                print("Queue contains : ",end=" ")
                for i in range(self.front,self.rear+1):
                    print(self.queue[i],end=" ")
                print("\n-------------------------------------------------------------------
-------------------")
        except QueueEmpty:
            print("QueueEmpty Error : Nothing to display queue is
empty!!!!!!!")
            print("-------------------------------------------------------------------
-------------")
```

```
print("--------------------------------------------------------------------------------")

n=int(input("Enter size of the queue : "))

print("--------------------------------------------------------------------------------")

que=Queue()

que.queue=[None]*n

while True:

    choice=int(input("Enter choice 1. Enqueue\n2. Dequeue\n3. Display all elements\n4. Exit"))

    print("--------------------------------------------------------------------------------")

    if(choice==1):

        ele=int(input("Enter element that you want to insert : "))

        print("--------------------------------------------------------------------------------")

        que.enqueue(ele)

    elif(choice==2):

        del_ele,che=que.dequeue()

        if(che==True):
```

```
            print("Deleted element : ",del_ele)
            print("--------------------------------------------------------------------------------")
    elif(choice==3):
        que.display()
    elif(choice==4):
        print("Exited successfully!!!")
        print("--------------------------------------------------------------------------------")
        break
    else:
        print("Please enter a valid number!!")
        print("--------------------------------------------------------------------------------")
```

Output:

```
PS D:\KJSCe\OSTPL Lab> python main.py
--------------------------------------------------------------------------------
Enter size of the queue : 5
--------------------------------------------------------------------------------
Enter choice 1. Enqueue
2. Dequeue
3. Display all elements
4. Exit1
--------------------------------------------------------------------------------
Enter element that you want to insert : 45
--------------------------------------------------------------------------------
Element  45  added successfully!!!!!
--------------------------------------------------------------------------------
Enter choice 1. Enqueue
2. Dequeue
3. Display all elements
4. Exit1
--------------------------------------------------------------------------------
Enter element that you want to insert : 78
--------------------------------------------------------------------------------
Element  78  added successfully!!!!!
--------------------------------------------------------------------------------
Enter choice 1. Enqueue
2. Dequeue
3. Display all elements
4. Exit1
--------------------------------------------------------------------------------
Enter element that you want to insert : 89
--------------------------------------------------------------------------------
Element  89  added successfully!!!!!
--------------------------------------------------------------------------------
Enter choice 1. Enqueue
2. Dequeue
3. Display all elements
4. Exit1
--------------------------------------------------------------------------------
Enter element that you want to insert : 56
--------------------------------------------------------------------------------
Element  56  added successfully!!!!!
--------------------------------------------------------------------------------
```

```
---------------------------------------------------------------------------------------
Element  56  added successfully!!!!!
---------------------------------------------------------------------------------------
Enter choice 1. Enqueue
2. Dequeue
3. Display all elements
4. Exit1
---------------------------------------------------------------------------------------
Enter element that you want to insert : 23
---------------------------------------------------------------------------------------
Element  23  added successfully!!!!!
---------------------------------------------------------------------------------------
Enter choice 1. Enqueue
2. Dequeue
3. Display all elements
4. Exit1
---------------------------------------------------------------------------------------
Enter element that you want to insert : 12
---------------------------------------------------------------------------------------
QueueFull Error : There is no space in the queue for the element queue is full!!!!!!
---------------------------------------------------------------------------------------
Enter choice 1. Enqueue
2. Dequeue
3. Display all elements
4. Exit3
---------------------------------------------------------------------------------------
Queue contains :  45 78 89 56 23
---------------------------------------------------------------------------------------
Enter choice 1. Enqueue
2. Dequeue
3. Display all elements
4. Exit2
---------------------------------------------------------------------------------------
Deleted element :  45
---------------------------------------------------------------------------------------
Enter choice 1. Enqueue
2. Dequeue
3. Display all elements
4. Exit2
---------------------------------------------------------------------------------------
Deleted element :  78
```

```
--------------------------------------------------------------------------------------------
Deleted element :  78
--------------------------------------------------------------------------------------------
Enter choice 1. Enqueue
2. Dequeue
3. Display all elements
4. Exit2
--------------------------------------------------------------------------------------------
Deleted element :  89
--------------------------------------------------------------------------------------------
Enter choice 1. Enqueue
2. Dequeue
3. Display all elements
4. Exit2
--------------------------------------------------------------------------------------------
Deleted element :  56
--------------------------------------------------------------------------------------------
Enter choice 1. Enqueue
2. Dequeue
3. Display all elements
4. Exit2
--------------------------------------------------------------------------------------------
Deleted element :  23
--------------------------------------------------------------------------------------------
Enter choice 1. Enqueue
2. Dequeue
3. Display all elements
4. Exit2
--------------------------------------------------------------------------------------------
QueueEmpty Error : Queue is empty nothing to delete!!!!!!
--------------------------------------------------------------------------------------------
Enter choice 1. Enqueue
2. Dequeue
3. Display all elements
4. Exit3
--------------------------------------------------------------------------------------------
QueueEmpty Error : Nothing to display queue is empty!!!!!!!
--------------------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------------------
QueueEmpty Error : Nothing to display queue is empty!!!!!!!
--------------------------------------------------------------------------------------------
Enter choice 1. Enqueue
2. Dequeue
3. Display all elements
4. Exit6
--------------------------------------------------------------------------------------------
Please enter a valid number!!
--------------------------------------------------------------------------------------------
Enter choice 1. Enqueue
2. Dequeue
3. Display all elements
4. Exit4
--------------------------------------------------------------------------------------------
Exited successfully!!!
--------------------------------------------------------------------------------------------
```

Conclusion: By performing this experiment we understood and implemented the concepts of errors and exceptions in python. Also understood the insights of queue data structure. We have also touched some concepts of classes and functions in python.

Date: 6 / 4 / 2021 **Signature of faculty in-charge**

Post Lab Descriptive Questions

Q1. When will the else part of try-except-else be executed?

a) always
b) when an exception occurs
c) when no exception occurs
d) when an exception occurs in to except block

ANS) c) when no exception occurs

Q2. Is the following Python code valid?

```
try:
    # Do something
except:
    # Do something
else:
    # Do something
```

a) no, there is no such thing as else
b) no, else cannot be used with except
c) no, else must come before except
d) yes

ANS) b) no, else cannot be used with except

Q3. 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

Q4. What will be the output of the following Python code?

```
def foo():
    try:
        return 1
    finally:
        return 2
k = foo()
print(k)
```

a) 1
b) 2
c) 3
d) error, there is more than one return statement in a single try-finally block

ANS) b) 2

Q5. What will be the output of the following Python code?

```
def foo():
    try:
        print(1)
    finally:
        print(2)
foo()
```

a) 1 2
b) 1
c) 2
d) none of the mentioned

ANS) a) 1 2

Q6. 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) c) invalid code

Q7. What will be output for the folllowing code?

```
try:
  print(x)
except:
  print(""An exception occurred"")
```

A. x
B. An exception occurred
C. Error
D. None of the above

ANS) B. An exception occurred

Q8. What will be output for the following code?

```
x = ""hello""
if not type(x) is int:
  raise TypeError(""Only integers are allowed"")
```

A. hello
B. garbage value
C. Only integers are allowed
D. Error

ANS) C. Only integers are allowed

Q9. What will be output for the folllowing code?

```
try:
  f = open(""demofile.txt"")
  f.write(""Lorum Ipsum"")
except:
  print(""Something went wrong when writing to the file"")
finally:
  f.close()
```

A. demofile.txt
B. Lorum Ipsum
C. Garbage value
D. Something went wrong when writing to the file

ANS) D. Something went wrong when writing to the file

Q10. What will be output for the folllowing code?

```
x=10
y=8
assert x>y, 'X too small'
```

A. Assertion Error
B. 10 8
C. No output
D. 108

ANS) C. No output