12 Feb Assignment

Q.1 What is an Exception in Python ? write the difference b/w Exceptions & Syntax errors.

→ An exception can be string, a class or an object. Most of the exceptions that the Python core raises are classes, with an argument that is an instance of the class

- Both exceptions & errors are the subclass of a throwable class. The error implies a problem that mostly arises due to the shortage of system resources. On the other hand, the exceptions occur during runtime & Compile time.

Q.2 What happens when an exception is not handled ? explain with an example :-

APPOINTMENT NOTES ~~Which of the following~~. The keyword 'try', 'except' & 'finally' are exception handling keywords in Python whereas the word accept is not a keyword at all

APPOINTMENT MEETING

Q. 3 Which Python statements are used to catch & handle exceptions? explain with an example.

The try & except block in Python is used to catch & handle exceptions. Python executes code following the try statement as a "Normal" part of the program.

ex:-

```
try:
    even_number = [2,4,6,8]
    print(even_numbers[5])

except ZeroDivisionError:
    print("Demominator cannot be 0.")

except IndexError:
    ~~print("Denominator cannot be 0.")~~
    print("Index out of Bound.")
```

Q.4 Explain (a) try & else.
(b) finally
(c) raise

APPOINTMENT NOTES Try & else

In some situation, we might want to run a certain block of code if the code block inside try runs without any errors.

APPOINTMENT/MEETING

```
try:
    num = int(input("Enter a numbe!"))
    assert num % 2 == 0
except:
    print("Not an even numbe!")
else
    reciprocal = 1/num
    print(reciprocal)
```

Finally :- In Python, the finally block is always executed no matter whether there is an exception or not.

The finally block is optional. And, for each try block, there can be only one finally block

```
try:
    numerator = 10
    denominator = 0
    result = numerator/denominator
print(result)
except:
print("Error; Denominator cannot be 0!")
    finally:
        print("This is finally block.")
```

Raising :- The raise statement allows the programmer to force a specific exception to occur. The sole argument in raise indicates the exception to be raised. this must be either an exception instance or an exception class (a class that derives from exception).

19 JULY
THURSDAY
200-165 • WK 29

JUL
M T W T F S S M T W T F S S
1 2 3 4 5 6 7 8
9 10 11 12 13 14 15 16 17 18 19 20 21 22
23 24 25 26 27 28 29 30 31

APPOINTMENT/MEETING deficit Raising Exception

9

=> try:
raise NameError("Hi there")
except NameError:
print("An Exception")
raise

10

11

12

Q.5 What are Custom Exceptions in Python? why do we need Custom Exceptions? explain with an example.

1

In Python, we can define custom exceptions by creating a new class that is derived from the built-in Exception class.

2

3

Here's the Syntax to define custom exceptions.

4

class CustomError(Exception):
...
pass
try:
...
except CustomError:
...

5

Here, CustomError is a User defined error which inherits from the exception class

APPOINTMENT NOTES

Q.6 Create a custom exception class. Use this class to handle exception.

EETING we can further customize this class o accept other arguments as per our needs.

To know exception classes of object-Oriented Programming.

```
ass SalaryNotInRangeError(Exception):
    """Exception raised for errors in the input
                    Salary.
Attributes:
    salary -- input salary which caused the
                    error
    message -- explanation of the error """

f __init__(self, salary, message = "Salary
is not in (5000, 15000) range"):
    self.salary = salary
    self.message = message
    super().__init__(self.message)

alary = int(input("Enter salary amount:"))
    if not 5000 < salary < 15000:
        raise SalaryNotInRangeError
                    (salary)
```

NOTES

AUG SEP OCT NOV DEC