Q1. What are the two latest user-defined exception constraints in Python 3.X?

Ans:- To generate a user defined exception, we use **the “raise” keyword when a certain condition is** met. The exception is then handled by the except block of the code. We then use pass statement. pass statement is used to show that we will not implement anything in our custom exception class.

By raising an inbuilt exception explicitly using raise keyword, we can use them anywhere in the program to enforce constraints on the values of variables.

For example, suppose we have to calculate the year of birth of a person from its age, we can do it as following:

try:

age= -10 print("Age is:")

print(age)

if age<0:

raise ValueError

yearOfBirth= 2021-age

print("Year of Birth is:")

print(yearOfBirth)

except

ValueError: print("Input Correct age.")

Output:- Age is: -10

Input Correct age.

We can use assert statement to implement constraints on values of our variable in python. When, the condition given in assert statement is not met, the program gives AssertionError in output.

try:

age= -10

print("Age is:")

print(age)

assert age>0

yearOfBirth= 2021-age

print("Year of Birth is:")

print(yearOfBirth)

except AssertionError:

print("Input Correct age.")

output:- Age is: -10

Input Correct age.

Q2. How are class-based exceptions that have been raised matched to handlers?

Ans:- The exceptions that can be handled are class-based and can be handled between TRY and ENDTRY. The associated exception classes are predefined in the system and begin with the prefix CX\_SY\_, such as CX\_SY\_ZERODIVIDE. In the ABAP keyword documentation, the exception classes whose exceptions may occur when a corresponding ABAP statement is executed are listed for each keyword.

Class-based exceptions are **realized as instances of exception classes**. ... Class-based exceptions are raised either by the ABAP runtime environment or by a program. Exception situations recognized by the system, and whose causes can be handled in the program, raise predefined class-based exceptions.

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.

Q3. Describe two methods for attaching context information to exception artefacts.

Ans:-

|  |
| --- |
| **Exception Name & Description** |
| 1 | Exception Base class for all exceptions |
| 2 | StopIteration Raised when the next() method of an iterator does not point to any object. |
| 3 | SystemExit Raised by the sys.exit() function. |

Q4. Describe two methods for specifying the text of an exception object's error message.

Ans:- Errors happen all the time in the software world. It might be an invalid user input or an external system that is not responding, or it’s a simple programming error. In all these situations, the errors occur at runtime and the application needs to handle them. Otherwise, it crashes and can’t process further requests.

Even if a statement or expression is syntactically correct, it may cause an error when an attempt is made to execute it. Errors detected during execution are called *exceptions* and are not unconditionally fatal. Most exceptions are not handled by programs, however, and result in error messages as shown here:

>>>

**>>>** 10 \* (1/0)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

ZeroDivisionError: division by zero

**>>>** 4 + spam\*3

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

NameError: name 'spam' is not defined

**>>>** '2' + 2

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: can only concatenate str (not "int") to str

The last line of the error message indicates what happened. Exceptions come in different types, and the type is printed as part of the message: the types in the example are [ZeroDivisionError](https://docs.python.org/3/library/exceptions.html" \l "ZeroDivisionError" \o "ZeroDivisionError), [NameError](https://docs.python.org/3/library/exceptions.html" \l "NameError" \o "NameError) and [TypeError](https://docs.python.org/3/library/exceptions.html" \l "TypeError" \o "TypeError). The string printed as the exception type is the name of the built-in exception that occurred. This is true for all built-in exceptions, but need not be true for user-defined exceptions (although it is a useful convention). Standard exception names are built-in identifiers (not reserved keywords).

Q5. Why do you no longer use string-based exceptions?

Ans:- **Except for adding elements in string**, we can perform all the operations