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

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

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

Q4. Describe two methods for specifying the text of an exception object&#39;s error message.

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

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

1. **User-defined exceptions must inherit from BaseException or Exception**:
   * In Python 3.X, all custom exceptions must be derived from the built-in BaseException class (or more commonly from Exception, which is a subclass of BaseException). This ensures that custom exceptions integrate properly into Python's exception-handling mechanism.

Example:  
python  
Copy code  
class MyCustomError(Exception):

pass

1. **User-defined exceptions should not be old-style classes**:
   * In Python 3.X, old-style classes (those that do not explicitly inherit from object) no longer exist. Therefore, user-defined exceptions must be new-style classes, which means they must explicitly inherit from a base class, such as Exception.

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

Class-based exceptions are matched to handlers through **exception inheritance**. When an exception is raised, Python looks for the first except block that matches the type of the raised exception or any of its parent classes. This means that if a handler for a more specific exception type is not found, Python will continue to look up the inheritance chain for a more general exception that can handle it.

Example:

python

Copy code

class MyCustomError(Exception):

pass

class MySpecificError(MyCustomError):

pass

try:

raise MySpecificError("An error occurred")

except MySpecificError:

print("Caught MySpecificError")

except MyCustomError:

print("Caught MyCustomError")

# Outputs: Caught MySpecificError

In the example above, the MySpecificError exception is caught by the except MySpecificError block. If this block were not present, it would be caught by the except MyCustomError block.

### **Q3. Describe two methods for attaching context information to exception artifacts.**

1. **Using Exception Arguments**:
   * When raising an exception, you can pass context information as arguments to the exception object. These arguments can then be accessed later when handling the exception.

Example:  
python  
Copy code  
class MyCustomError(Exception):

def \_\_init\_\_(self, message, error\_code):

super().\_\_init\_\_(message)

self.error\_code = error\_code

try:

raise MyCustomError("An error occurred", 404)

except MyCustomError as e:

print(f"Error: {e}, Code: {e.error\_code}")

1. **Using the \_\_context\_\_ or \_\_cause\_\_ Attributes**:
   * Python exceptions have \_\_context\_\_ and \_\_cause\_\_ attributes that automatically store information about the original exception when another exception is raised while handling the first one. This allows you to chain exceptions and retain context.

Example:  
python  
Copy code  
try:

1 / 0

except ZeroDivisionError as e:

raise ValueError("A value error occurred") from e

try:

1 / 0

except ZeroDivisionError:

raise ValueError("A value error occurred")

# In the first example, e.\_\_cause\_\_ points to the ZeroDivisionError.

# In the second example, e.\_\_context\_\_ points to the ZeroDivisionError.

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

1. **Passing a Message String to the Exception Constructor**:
   * When raising an exception, you can specify the error message by passing a string to the exception's constructor. This message can then be accessed using the exception object’s str or repr methods.

Example:  
python  
Copy code  
try:

raise ValueError("Invalid input provided")

except ValueError as e:

print(str(e)) # Outputs: Invalid input provided

1. **Overriding the \_\_str\_\_ or \_\_repr\_\_ Methods in a Custom Exception Class**:
   * You can customize the error message by overriding the \_\_str\_\_ or \_\_repr\_\_ methods in your custom exception class. This allows you to dynamically generate the error message based on attributes or other conditions.

Example:  
python  
Copy code  
class MyCustomError(Exception):

def \_\_init\_\_(self, error\_code):

self.error\_code = error\_code

def \_\_str\_\_(self):

return f"Error with code: {self.error\_code}"

try:

raise MyCustomError(404)

except MyCustomError as e:

print(e) # Outputs: Error with code: 404

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

String-based exceptions were used in earlier versions of Python (prior to Python 2.0) but have been deprecated and removed in favor of **class-based exceptions**. The reasons for this shift include:

1. **Lack of Inheritance**:
   * String-based exceptions do not support inheritance, which means that you cannot create a hierarchy of exceptions. This makes it difficult to handle related groups of exceptions in a structured way.
2. **No Structured Information**:
   * String-based exceptions can only convey information through the exception string itself, limiting the ability to include additional context or metadata. Class-based exceptions allow you to store additional information (e.g., error codes, error context) as attributes of the exception object.
3. **Ambiguity and Error-Prone**:
   * String-based exceptions are prone to typos and other errors because they rely on matching strings exactly. Class-based exceptions, on the other hand, rely on the type of the exception, which is much less error-prone and easier to manage.
4. **Improved Compatibility and Consistency**:
   * Class-based exceptions provide a consistent and more powerful mechanism for error handling, which aligns with modern object-oriented programming principles. This makes the code more readable, maintainable, and compatible with Python's standard libraries.

Due to these advantages, class-based exceptions are the standard and recommended approach in Python.