**FSDS MAY BATCH 2022(Python Assignment -8)**

**Submitted by: Shubham Tiwari**

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

Ans: In Python 3.X, there are two main constraints for user-defined exceptions:

1. All user-defined exceptions should be derived from the **BaseException** class or one of its subclasses (such as **Exception**). This is because **BaseException** is the base class for all built-in exceptions in Python, and it ensures that user-defined exceptions are consistent with the built-in exceptions in terms of behavior and handling.
2. User-defined exceptions should be derived directly or indirectly from the **Exception** class. This is because the **BaseException** is the base class for all built-in exceptions and it's not intended to be instantiated or subclassed directly.

It's not recommended to raise or catch the **BaseException** class. The **BaseException** class is intended to be used as the root of the exception class hierarchy.

Also, it's generally a good practice to define your own exception classes and use them specifically when raising exceptions. This allows you to clearly distinguish between different types of errors and handle them in an appropriate way.

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

Ans: In Python, class-based exceptions are matched to handlers using the exception's class hierarchy. When an exception is raised, the interpreter starts at the current frame and looks for an exception handler that can handle the exception. It looks for a **try** block that has a corresponding **except** block that mentions the class of the exception or one of its superclasses.

The interpreter looks for the most specific handler that can handle the raised exception. If the interpreter finds a handler that can handle the exception, it will execute the corresponding **except** block and then continue execution after the **try-except** block. If the interpreter does not find a matching handler, it will go to the next higher frame and repeat the process until it finds a matching handler or until it reaches the top-level frame.

If no matching handler is found in the entire call stack, the interpreter will raise a **SystemExit** exception and terminate the program.

It's important to note that if you have multiple **except** blocks with the same class or super class, the interpreter will execute the first one that is matched.

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

1. Using the **raise** statement with a tuple: In python, the **raise** statement can be used to raise an exception and attach additional context information to it by passing a tuple as the argument. The first element of the tuple should be the exception class or instance, and the remaining elements are used to store the context information.

Example:

**raise ValueError("Invalid value", value)**

1. Using the with statement with **contextlib.contextmanager:** The contextlib module provides the contextmanager decorator that can be used to define a context manager that attaches context information to an exception. This method allows you to attach context information to an exception without having to explicitly pass it as an argument to the raise statement.

Example:

**import contextlib**

**@contextlib.contextmanager**

**def my\_context\_manager():**

**try:**

**yield**

**except Exception as e:**

**e.context\_information = "Additional context information"**

**raise**

1. You can use the with statement to use the my\_context\_manager to attach the context information to the exception .

**with my\_context\_manager():**

**raise Exception("Something went wrong")**

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

Ans: Notably, EnvironmentError (with subclasses IOError and OSError) has a first argument of errno, second of strerror. There is no message... strerror is roughly similar to what would normally be a message.   I think the proper way of handling this is to identify the specific Exception subclasses we want to catch, and then catch only those instead of everything with an except Exception, then utilize whatever attributes that specific subclass defines however we want.

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

Ans: This is because a string is a sequence, we must assume that error messages are always more than two characters long.