Assignment 9 Exception handling-2

September 16, 2023

Q1. Explain why we have to use the Exception class while creating a Custom Exception. Note: Here Exception class refers to the base class for all the exceptions.

ANS:

In Python, when creating a custom exception, it is recommended to derive your custom exception class from the Exception class (or one of its subclasses) for several important reasons:

1. Consistency with Built-in Exceptions: Python has a rich hierarchy of built-in exception classes, and they all inherit from the BaseException or Exception class. This hierarchy provides a consistent way to handle and categorize exceptions. When you create a custom exception that inherits from the Exception class, it aligns with this established convention and ensures that your custom exception is treated like other exceptions in Python.
2. Exception Handling: Deriving from the Exception class allows your custom exception to be caught and handled using the same mechanisms that handle other exceptions. This means you can use try-except blocks to catch your custom exception just like you would for built-in exceptions. It also makes it easier for developers to understand how to handle your custom exception because they can follow the same patterns they use for other exceptions.
3. Clarity and Documentation: Deriving from Exception or a related subclass provides a clear indication that your class represents an exception. It serves as a form of self-documentation, making it obvious to other developers that the class is meant to be used for error conditions. This can improve the readability and maintainability of your code.
4. Compatibility: Python's exception handling mechanisms are designed to work with classes that inherit from Exception. If you create a custom exception that doesn't inherit from it, you may encounter unexpected behavior when trying to catch or handle the exception using standard Python exception handling constructs.

[26]:
```python
class CustomError(Exception):
    def __init__(self, message):
        super().__init__(message)

try:
    raise CustomError("This is a custom exception.")
except CustomError as e:
    print("Caught custom exception:", e)
```

```
Caught custom exception: This is a custom exception.
```

[ ]:

```

```

Q2. Write a python program to print Python Exception Hierarchy.

ANS:

[29]:

```python
import builtins

def print_exception_hierarchy(base_exception, depth=0):
    indent = "  " * depth
    print(indent + base_exception.__name__)

    # Get the subclasses of the base_exception
    subclasses = base_exception.__subclasses__()

    for subclass in subclasses:
        print_exception_hierarchy(subclass, depth + 1)

print("Python Exception Hierarchy:")
print_exception_hierarchy(BaseException)
```

```
Python Exception Hierarchy:
BaseException
  Exception
    TypeError
      FloatOperation
      MultipartConversionError
    StopAsyncIteration
    StopIteration
    ImportError
      ModuleNotFoundError
      ZipImportError
    OSError
      ConnectionError
        BrokenPipeError
        ConnectionAbortedError
        ConnectionRefusedError
        ConnectionResetError
          RemoteDisconnected
      BlockingIOError
      ChildProcessError
      FileExistsError
      FileNotFoundError
      IsADirectoryError
      NotADirectoryError
      InterruptedError
        InterruptedSystemCall
      PermissionError
      ProcessLookupError
      TimeoutError
      UnsupportedOperation
      itimer_error
      herror
      gaierror
      SSLError
        SSLCertVerificationError
        SSLZeroReturnError
        SSLWantWriteError
        SSLWantReadError
        SSLSyscallError
        SSLEOFError
      Error
        SameFileError
      SpecialFileError
      ExecError
      ReadError
      URLError
        HTTPError
        ContentTooShortError
      BadGzipFile
    EOFError
      IncompleteReadError
    RuntimeError
      RecursionError
      NotImplementedError
        ZMQVersionError
        StdinNotImplementedError
      _DeadlockError
      BrokenBarrierError
      BrokenExecutor
        BrokenThreadPool
      SendfileNotAvailableError
      ExtractionError
      VariableError
    NameError
      UnboundLocalError
    AttributeError
      FrozenInstanceError
    SyntaxError
      IndentationError
        TabError
    LookupError
      IndexError
      KeyError
        NoSuchKernel
        UnknownBackend
      CodecRegistryError
    ValueError
      UnicodeError
        UnicodeEncodeError
        UnicodeDecodeError
        UnicodeTranslateError
      UnsupportedOperation
      JSONDecodeError
      SSLCertVerificationError
      Error
      UnsupportedDigestmodError
      IllegalMonthError
      IllegalWeekdayError
      ParserError
      ClassNotFound
      ClipboardEmpty
      MessageDefect
        NoBoundaryInMultipartDefect
        StartBoundaryNotFoundDefect
        CloseBoundaryNotFoundDefect
        FirstHeaderLineIsContinuationDefect
        MisplacedEnvelopeHeaderDefect
        MissingHeaderBodySeparatorDefect
        MultipartInvariantViolationDefect
        InvalidMultipartContentTransferEncodingDefect
        UndecodableBytesDefect
        InvalidBase64PaddingDefect
        InvalidBase64CharactersDefect
        InvalidBase64LengthDefect
        HeaderDefect
          InvalidHeaderDefect
          HeaderMissingRequiredValue
          NonPrintableDefect
          ObsoleteHeaderDefect
          NonASCIILocalPartDefect
          InvalidDateDefect
      MacroToEdit
      InvalidFileException
      UnequalIterablesError
      InvalidVersion
      _InvalidELFFileHeader
      InvalidWheelFilename
      InvalidSdistFilename
      InvalidSpecifier
      InvalidMarker
      UndefinedComparison
      UndefinedEnvironmentName
      InvalidRequirement
```

```
    RequirementParseError
  InvalidVersion
AssertionError
ArithmeticError
  FloatingPointError
  OverflowError
  ZeroDivisionError
    DivisionByZero
    DivisionUndefined
  DecimalException
    Clamped
    Rounded
      Underflow
      Overflow
    Inexact
      Underflow
      Overflow
    Subnormal
      Underflow
    DivisionByZero
    FloatOperation
    InvalidOperation
      ConversionSyntax
      DivisionImpossible
      DivisionUndefined
      InvalidContext
SystemError
  CodecRegistryError
ReferenceError
MemoryError
BufferError
Warning
  UserWarning
    GetPassWarning
    FormatterWarning
  EncodingWarning
  DeprecationWarning
    ProvisionalWarning
  PendingDeprecationWarning
  SyntaxWarning
  RuntimeWarning
    ProactorSelectorThreadWarning
    UnknownTimezoneWarning
    PEP440Warning
  FutureWarning
    ProvisionalCompleterWarning
  ImportWarning
  UnicodeWarning
  BytesWarning
  ResourceWarning
  DeprecatedTzFormatWarning
  PkgResourcesDeprecationWarning
_OptionError
_Error
error
Verbose
Error
SubprocessError
  CalledProcessError
  TimeoutExpired
TokenError
StopTokenizing
ClassFoundException
EndOfBlock
TraitError
Error
Error
  CancelledError
  TimeoutError
  InvalidStateError
_GiveupOnSendfile
error
Incomplete
TimeoutError
InvalidStateError
LimitOverrunError
QueueEmpty
QueueFull
Empty
Full
ArgumentError
ZMQBaseError
  ZMQError
    ContextTerminated
    Again
    InterruptedSystemCall
  ZMQBindError
  NotDone
PickleError
  PicklingError
  UnpicklingError
_Stop
ArgumentError
ArgumentTypeError
ConfigError
  ConfigLoaderError
```

```
    ArgumentError
  ConfigFileNotFound
ConfigurableError
  MultipleInstanceError
ApplicationError
error
TimeoutError
error
ReturnValueIgnoredError
KeyReuseError
UnknownKeyError
LeakedCallbackError
BadYieldError
ReturnValueIgnoredError
Return
InvalidPortNumber
error
LZMAError
RegistryError
_GiveupOnFastCopy
Error
  NoSectionError
  DuplicateSectionError
  DuplicateOptionError
  NoOptionError
  InterpolationError
    InterpolationMissingOptionError
    InterpolationSyntaxError
    InterpolationDepthError
  ParsingError
    MissingSectionHeaderError
NoIPAddresses
BadZipFile
LargeZipFile
BadEntryPoint
NoSuchEntryPoint
DuplicateKernelError
ErrorDuringImport
NotOneValueFound
CannotEval
OptionError
BdbQuit
Restart
ExceptionPexpect
  EOF
  TIMEOUT
PtyProcessError
FindCmdError
```

```
HomeDirError
ProfileDirError
IPythonCoreError
  TryNext
  UsageError
  StdinNotImplementedError
InputRejected
GetoptError
ErrorToken
PrefilterError
AliasError
  InvalidAliasError
Error
  InterfaceError
  DatabaseError
    InternalError
    OperationalError
    ProgrammingError
    IntegrityError
    DataError
    NotSupportedError
Warning
SpaceInInput
DOMException
  IndexSizeErr
  DomstringSizeErr
  HierarchyRequestErr
  WrongDocumentErr
  InvalidCharacterErr
  NoDataAllowedErr
  NoModificationAllowedErr
  NotFoundErr
  NotSupportedErr
  InuseAttributeErr
  InvalidStateErr
  SyntaxErr
  InvalidModificationErr
  NamespaceErr
  InvalidAccessErr
  ValidationErr
ValidationError
EditReadOnlyBuffer
_Retry
InvalidLayoutError
HeightIsUnknownError
ParserSyntaxError
InternalParseError
_PositionUpdatingFinished
```

```
SimpleGetItemNotFound
UncaughtAttributeError
HasNoContext
ParamIssue
_JediError
  InternalError
  WrongVersion
  RefactoringError
OnErrorLeaf
InvalidPythonEnvironment
MessageError
  MessageParseError
    HeaderParseError
    BoundaryError
  MultipartConversionError
  CharsetError
Error
HTTPException
  NotConnected
  InvalidURL
  UnknownProtocol
  UnknownTransferEncoding
  UnimplementedFileMode
  IncompleteRead
  ImproperConnectionState
    CannotSendRequest
    CannotSendHeader
    ResponseNotReady
  BadStatusLine
    RemoteDisconnected
  LineTooLong
InteractivelyDefined
KillEmbedded
Error
  NoSuchProcess
    ZombieProcess
  AccessDenied
  TimeoutExpired
_Ipv6UnsupportedError
QueueEmpty
QueueFull
DebuggerInitializationError
ExpatError
Error
  ProtocolError
  ResponseError
  Fault
ParseBaseException
```

```
        ParseException
        ParseFatalException
          ParseSyntaxException
      RecursiveGrammarException
      ResolutionError
        VersionConflict
          ContextualVersionConflict
        DistributionNotFound
        UnknownExtra
      _Error
      UnableToResolveVariableException
      InvalidTypeInArgsException
      CustomError
      InvalidAgeError
      InvalidAgeError
      CustomError
    GeneratorExit
    SystemExit
    KeyboardInterrupt
    CancelledError
    AbortThread
```

[ ]:

```

```

Q3. What errors are defined in the ArithmeticError class? Explain any two with an example.

ANS:

The ArithmeticError class in Python is a base class for exceptions that are related to arithmetic operations. It serves as a parent class for various arithmetic-related exception classes. Two common exceptions derived from ArithmeticError are ZeroDivisionError and OverflowError.

0.0.1 ZeroDivisionError:

ZeroDivisionError is raised when you attempt to divide a number by zero, which is mathematically undefined.

[30]:

```python
try:
    numerator = 10
    denominator = 0
    result = numerator / denominator  # Attempting to divide by zero
except ZeroDivisionError as e:
    print(f"Error: {e}")
else:
    print("Result:", result)
```

```
Error: division by zero
```

0.0.2 OverflowError:

OverflowError is raised when an arithmetic operation exceeds the limits of the data type being used.

[31]:
```python
try:
    large_number = 2 ** 1000  # Attempting to calculate a very large power of 2
except OverflowError as e:
    print(f"Error: {e}")
else:
    print("Result:", large_number)
```

```
Result: 107150860718626732094842504906000181056140481170553360744375038837035105
1124936122493198378815695858127594672917553146825187145285692314043598457757469857
4803934567774824230985421074605062371141877954182153046474983581941267398767559165
5439460770629145711964776865421676604298316526243868372056680693 76
```

[ ]:
```

```

Q4. Why LookupError class is used? Explain with an example KeyError and IndexError.

ANS:

The LookupError class in Python is a base class for exceptions that occur when you try to access an element or key in a collection (such as a list or dictionary) and the element/key does not exist. LookupError itself is not meant to be directly raised; instead, it serves as a parent class for more specific lookup-related exceptions, such as KeyError and IndexError.

0.0.3 KeyError:

KeyError is raised when you try to access a dictionary with a key that does not exist in the dictionary.

[33]:
```python
student_grades = {"Alice": 85, "Bob": 92, "Charlie": 78}

try:
    grade = student_grades["David"]
    print(f"David's grade: {grade}")
except KeyError as e:
    print(f"KeyError: {e}")
```

```
KeyError: 'David'
```

[ ]:
```

```

0.0.4 IndexError:

IndexError is raised when you try to access an element in a sequence (e.g., a list or tuple) using an index that is out of range.

[34]:
```python
my_list = [10, 20, 30, 40, 50]

try:
    value = my_list[10]
    print(f"Value at index 10: {value}")
except IndexError as e:
    print(f"IndexError: {e}")
```

```
IndexError: list index out of range
```

[ ]:
```python

```

Q5. Explain ImportError. What is ModuleNotFoundError?

ANS:

ImportError and ModuleNotFoundError are both exceptions in Python that occur when there is an issue with importing modules or packages. However, there are differences between them:

0.0.5 ImportError:

ImportError is a base class for exceptions related to importing modules. It is raised when Python encounters an issue while trying to import a module or when there are problems within the imported module.

ImportError can have various subtypes, such as AttributeError, NameError, or ModuleNotFoundError, depending on the specific issue that occurred during the import.

[35]:
```python
try:
    import non_existent_module
except ImportError as e:
    print(f"ImportError: {e}")
```

```
ImportError: No module named 'non_existent_module'
```

0.0.6 ModuleNotFoundError:

ModuleNotFoundError is a specific subtype of ImportError that is raised when Python cannot find the module that you are trying to import. This exception was introduced in Python 3.6 to provide more specific and informative error messages.

[36]:
```python
try:
    import non_existent_module
except ModuleNotFoundError as e:
    print(f"ModuleNotFoundError: {e}")
```

```
ModuleNotFoundError: No module named 'non_existent_module'
```

[ ]:
```python

```

Q6. List down some best practices for exception handling in python.

ANS:

Here are some best practices for effective exception handling in Python:

1. Use Specific Exceptions: Catch specific exceptions whenever possible rather than catching generic ones like Exception or BaseException. This allows you to handle errors more precisely and prevents unintended side effects.
2. Keep Exception Blocks Short: Limit the amount of code within your try-except blocks. Only include the code that might raise an exception, and avoid wrapping large sections of code in a single try block.
3. Handle Exceptions Appropriately: Handle exceptions appropriately based on the specific error. Avoid simply catching exceptions and ignoring them, as this can hide bugs and make debugging difficult.
4. Use else Clause: Use the else clause in a try-except block to include code that should execute when no exceptions are raised. This can help improve code readability.
5. Use finally for Cleanup: When you need to ensure certain actions (e.g., closing files or releasing resources) always occur, use the finally block. It executes regardless of whether an exception was raised or not.
6. Avoid Bare except: Avoid using a bare except clause (i.e., except:) without specifying the exception type. It can catch unexpected exceptions and make debugging difficult. Be explicit about which exceptions you're handling.
7. Use Context Managers (with Statements): Utilize context managers (e.g., with statements) for resource management, like opening and closing files. Context managers automatically handle cleanup.
8. Log Exceptions: Log exceptions with a logging library (e.g., logging) to keep track of errors and their context. Logging can aid in debugging and troubleshooting.
9. Reraise Exceptions Carefully: If you need to catch an exception but still want it to propagate up the call stack, you can re-raise it using raise without any arguments. This is helpful for debugging and preserving the original exception's information.
10. Custom Exceptions: Create custom exception classes when you need to handle application-specific errors. This improves code readability and allows you to provide meaningful error messages.
11. Use try-except Around External Dependencies: When interacting with external resources or services, wrap those interactions in try-except blocks to gracefully handle issues and provide feedback to the user.
12. Don't Suppress Errors: Avoid suppressing errors by catching exceptions and not doing anything with them. If an error occurs, it's often best to let it propagate so you can diagnose and fix the underlying issue.
13. Document Exception Handling: Document your exception handling approach, especially if it's non-trivial. Comments or docstrings can help other developers understand your code's error-handling strategy.

```
[ ]:
```