Discussion-7 Python Exception Handling

Discussion Topic:

Review the following content on Python 3 Errors and ExceptionsLinks to an external site..
After studying this technical note, explain various ways in which Python 3 programming exceptions can be handled. Why is it important to handle Python 3 exceptions? Provide a code example of Python 3 exception handling and at least one reference to support your findings.
Be sure to post an initial, substantive response by Thursday at 11:59 p.m. MT and respond to 2 or more peers with substantive responses by Sunday at 11:59 p.m. MT. Respond to these posts in any of the following ways:

- Build on something your classmate said
- Explain why and how you see things differently
- Ask a probing or clarifying question
- Share an insight from having read your classmates' postings
- Offer and support an opinion
- Validate an idea with your own experience
- Expand on your classmates' postings
- Ask for evidence that supports the post.

My Post:

Hello Class,
Python can handle exceptions in severe ways that involve a try block.
- Try-except block, the code that may raise an exception is placed in the try block, and the except block specifies the actions to take if an exception occurs (Python Software Foundation, n.d.). For example:

```
try:
   result = 1 / 0
except ZeroDivisionError:
   print("Cannot divide by zero.")
```

- Catching multiple exceptions in one block, by using a try block with several except blocks to generate specific responses for each exception type. Or by using a tuple to catch multiple exceptions in a single exception. For example:

```
# One try block and several except blocks
try:
   result = 1 / 'a'
except ZeroDivisionError:
   print("Cannot divide by zero.")
except TypeError:
   print("Type error occurred.")

# One try block and one except tuple block
try:
   # some operation
   result = 1 / 'a'
except (ZeroDivisionError, TypeError) as e:
   print(f"Error occurred: {e}")
```

- The else clause, is placed after the try-except blocks and runs if the try block does not raise an exception. For example:

```
try:
   result = 1 / 2
except ZeroDivisionError:
   print(“Cannot divide by zero.”)
else:
   print(“Division successful.”)
```

- The finally clause, is always placed, after the try block or any except block. It contains code that runs no matter what, typically for cleaning up resources like files or network connections, even if an exception was raised. For example:

```
try:
   result = 1 / ‘a’
except ZeroDivisionError:
   print(“Cannot divide by zero.”)
except TypeError:
   print(“Type error occurred.”)
```

```
else:
    print("Division successful.")
finally:
    print("Goodbye, world!")
```

- Raising exceptions: the raise clause raises exceptions by forcing an exception to occur, usually to indicate that a certain condition has not been met. For example:

```
if 'a' > 5:
    raise ValueError("A must not exceed 5")
```

- Exception chaining with raise: you can chain exceptions with the clause raise. This is useful for adding context to an original error. For Example

```
try:
    open('myfile.txt')
except FileNotFoundError as e:
    raise RuntimeError("Failed to open file") from e
```

- Custom exceptions, You can define your own exception classes by inheriting from the Exception class or any other built-in exception class (Mitchell, 2022). For example:

```
class My_custom_ (Exception):
    pass

try:
    raise MyCustomError("An error occurred")
except MyCustomError as e:
    print(e)
```

- Enriching exceptions, you can add information or context to an exception by using the add_note() method to 'append' custom messages or notes to the exception object aka e. For example:

```
def divide_numbers(a, b):
    try:
        result = a / b
    except ZeroDivisionError as e:
        e.add_note("Cannot divide by zero")
        e.add_note("Please provide a non-zero divisor")
        raise

try:
    num1 = 10
    num2 = 0
    divide_numbers(num1, num2)
except ZeroDivisionError as e:
    print("An error occurred:")
    print(str(e))
```

Handling exceptions is important for several reasons:

1. Prevents program crashes: Unhandled exceptions can cause the program to crash, leading to data loss and a poor user experience.
2. Provides meaningful error messages: By handling exceptions, you can provide users with informative error messages that help them understand what went wrong and how to fix it.
3. Allows for graceful degradation: Exception handling enables the program to continue running even if an error occurs.

A simple program error handling example:

```
##----------------------------------------
# Pseudocode:
# 1. Define a custom exception class called CustomError.
# 2. Create a function that raises the CustomError exception based on a condition.
# 3. Use a try-except block to handle the CustomError exception.
#----------------------------------------
# Program Inputs:
# - num: An integer value.
#----------------------------------------
# Program Outputs:
# - Custom exception message when the CustomError exception is raised.
# - Success message when no exception is raised.
#----------------------------------------

class CustomError(Exception):
    '''
        A custom exception class.
    '''
    pass

def check_number(num):
    '''
        Checks if the given number is positive.

        :param int: num, an integer value.
        :raises CustomError: If the number is not positive.
        :Return: None

    '''
    if num <= 0:
        raise CustomError("Number must be positive.")
    print("Number is valid.")

#--- Main Program
def main() -> None:
    '''
```

```
    The main function that demonstrates the usage of custom exceptions.
  '''
  try:
    check_number(5)  # Valid number
    check_number(-2)  # Raises CustomError
  except CustomError as e:
    print(f"Error: {str(e)}")

#--- Execute the program
if __name__ == "__main__": main()
>>>
Number is valid.
Error: Number must be positive.
```

-Alex

References:

Mitchell R (2022, June 13). Custom exceptions. *Python Essential Training* [VIDEO]. LinkedIn Learning. https://www.linkedin.com/learning/python-essential-training-14898805/custom-exceptions?autoSkip=true&resume=false&u=2245842

Python Software Foundation. (n.d.). 8. Errors and Exceptions. *Python*. python.org.