|  |  |  |
| --- | --- | --- |
| Document name | : | HR.Skills.A3W10ST |
| Document description | : | This document contains the supporting topic: “Exception handling” |
| Document version | : | V 1.0 |
| Written by | : | Danny de Snoo |
| Date(s) written | : | 07-06-2024 |

Supporting topic description:

**Introduction:**

In Python, exceptions are unexpected events that can happen during the execution of a program, disrupting the normal flow of code. These can include errors like attempting to access a non-existent file. To handle these exceptions and prevent the program from crashing, Python provides a mechanism called the try-except block. The code that might raise an exception is placed inside the try block. If an exception occurs within the try block, Python immediately jumps to the corresponding except block, where the specific exception can be caught and handled gracefully. This allows developers to predict potential errors and define appropriate responses, ensuring the program can continue running smoothly even when unexpected issues arise. Using try-except blocks enhances the robustness of Python programs by providing a way to manage errors and prevent them from causing the entire program to terminate abruptly.

* Perform a brief research about exception handling in Python.
* Implement a simple example with an intentional error, like trying to open a file with a wrong name. Run the program and see what will be the result of the execution.
* Improve your program by adding try-except block.
* Raised exceptions are object instances. They expose methods to detect their types and messages. Use these object instances to print proper error messages to the user when an exception occurs.
* Improve the code given above for code analysis with exception handling.

Code:

|  |
| --- |
| class TemperatureDataAnalyzer:  def \_\_init\_\_(self, file\_path):  self.file\_path = file\_path  self.temperature\_data = []  # Method to open the file and load lines as an attribute  def load\_data(self):  with open(self.file\_path, 'r') as file:  data = [line.strip().split() for line in file]  self.temperature\_data = [list(map(int,d[:-1]))+[float(d[len(d)-1])] for d in data]  # Method to perform the analysis and construct the list  def construct\_temperature\_list(self):  temperature\_list = []  for data in self.temperature\_data:  month, day, year, temperature = data[:]  if year not in [item[0] for item in temperature\_list]:  temperature\_list.append((year, {}))  if month not in temperature\_list[-1][1]:  temperature\_list[-1][1][month] = 0.0  temperature\_list[-1][1][month] = max(temperature , temperature\_list[-1][1][month])  return temperature\_list  def main():  file\_path = './temps.txt'  analyzer = TemperatureDataAnalyzer(file\_path)  analyzer.load\_data()  temperature\_list = analyzer.construct\_temperature\_list()  print(temperature\_list)  if \_\_name\_\_ == '\_\_main\_\_':  main() |

• Perform a brief research about exception handling in Python.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Exceptions are unexpected events in a python script. This also known as errors.  When an exception or error takes place the program will immediately crash and report the offending line like this:   These exceptions can be caught by using a try-except structure. This will catch the error and run the code you set in the except when an exception takes place:    These are some of the important built-in exceptions/error’s:   |  |  | | --- | --- | | Exception | Description | | AssertionError | Raised when the assert statement fails. | | AttributeError | Raised on the attribute assignment or reference fails. | | EOFError | Raised when the input() function hits the end-of-file condition. | | FloatingPointError | Raised when a floating point operation fails. | | GeneratorExit | Raised when a generator's close() method is called. | | ImportError | Raised when the imported module is not found. | | IndexError | Raised when the index of a sequence is out of range. | | KeyError | Raised when a key is not found in a dictionary. | | KeyboardInterrupt | Raised when the user hits the interrupt key (Ctrl+c or delete). | | MemoryError | Raised when an operation runs out of memory. | | NameError | Raised when a variable is not found in the local or global scope. | | NotImplementedError | Raised by abstract methods. | | OSError | Raised when a system operation causes a system-related error. | | OverflowError | Raised when the result of an arithmetic operation is too large to be represented. | | ReferenceError | Raised when a weak reference proxy is used to access a garbage collected referent. | | RuntimeError | Raised when an error does not fall under any other category. | | StopIteration | Raised by the next() function to indicate that there is no further item to be returned by the iterator. | | SyntaxError | Raised by the parser when a syntax error is encountered. | | IndentationError | Raised when there is an incorrect indentation. | | TabError | Raised when the indentation consists of inconsistent tabs and spaces. | | SystemError | Raised when the interpreter detects internal error. | | SystemExit | Raised by the sys.exit() function. | | TypeError | Raised when a function or operation is applied to an object of an incorrect type. | | UnboundLocalError | Raised when a reference is made to a local variable in a function or method, but no value has been bound to that variable. | | UnicodeError | Raised when a Unicode-related encoding or decoding error occurs. | | UnicodeEncodeError | Raised when a Unicode-related error occurs during encoding. | | UnicodeDecodeError | Raised when a Unicode-related error occurs during decoding. | | UnicodeTranslateError | Raised when a Unicode-related error occurs during translation. | | ValueError | Raised when a function gets an argument of correct type but improper value. | | ZeroDivisionError | Raised when the second operand of a division or module operation is zero. | |

|  |  |  |
| --- | --- | --- |
| Sources |  | URL |
| Python official documentation | : | https://docs.python.org/3/tutorial/errors.html |
| TutorialsTeacher | : | https://www.tutorialsteacher.com/python/error-types-in-python |

• Implement a simple example with an intentional error, like trying to open a file with a wrong name. Run the program and see what will be the result of the execution.

|  |  |  |
| --- | --- | --- |
| Code:   |  | | --- | | File = open(“masterpiece.txt”) |   Error:   |  | | --- | | Traceback (most recent call last):  File "/Users/dannydesnoo/Documents/GitHub/HR.Basecamp/A3/Learning activities/Supporting topic - Exception handling/example1.py", line 1, in <module>  file = open("masterpiece.txt")  ^^^^^^^^^^^^^^^^^^^^^^^  FileNotFoundError: [Errno 2] No such file or directory: 'masterpiece.txt' | |

• Improve your program by adding try-except block.

|  |  |  |
| --- | --- | --- |
| Code:   |  | | --- | | try:  file = open("masterpiece.txt")  except:  print("An error has occurred") |   Console:   |  | | --- | | An error has occurred  Process finished with exit code 0 | |

• Raised exceptions are object instances. They expose methods to detect their types and messages. Use these object instances to print proper error messages to the user when an exception occurs.

|  |  |  |
| --- | --- | --- |
| Code:   |  | | --- | | try:  file = open("masterpiece.txt")  except FileNotFoundError as e:  print("file not found")  except:  print("unknown error") |   Console:   |  | | --- | | file not found  Process finished with exit code 0 | |

• Improve the code given above for code analysis with exception handling.

|  |  |
| --- | --- |
| Improved code:   |  | | --- | | class TemperatureDataAnalyzer:  def \_\_init\_\_(self, file\_path):  self.file\_path = file\_path  self.temperature\_data = []  # Method to open the file and load lines as an attribute  def load\_data(self):  with open(self.file\_path, 'r') as file:  data = [line.strip().split() for line in file]  self.temperature\_data = [list(map(int,d[:-1]))+[float(d[len(d)-1])] for d in data]  # Method to perform the analysis and construct the list  def construct\_temperature\_list(self):  temperature\_list = []  for data in self.temperature\_data:  month, day, year, temperature = data[:]  if year not in [item[0] for item in temperature\_list]:  temperature\_list.append((year, {}))  if month not in temperature\_list[-1][1]:  temperature\_list[-1][1][month] = 0.0  temperature\_list[-1][1][month] = max(temperature , temperature\_list[-1][1][month])  return temperature\_list  def main():  try:  file\_path = './temps.txt'  analyzer = TemperatureDataAnalyzer(file\_path)  analyzer.load\_data()  temperature\_list = analyzer.construct\_temperature\_list()  print(temperature\_list)  except FileNotFoundError:  print("File not found")  except TypeError:  print("Invalid data")  except ZeroDivisionError:  print("Invalid data")  except:  print("An unknown error has occurred")  if \_\_name\_\_ == '\_\_main\_\_':  main() | |