Double-click (or enter) to edit

1. Write a python program with Exception handling to input marks for five subjects Physics, Chemistry, Biology, Mathematics, and Computer. Calculate the percentage and grade according to the following:

i) Percentage >= 90%: Grade A

ii) Percentage >= 80%: Grade B

iii) Percentage >= 70%: Grade C

iv) Percentage >= 60%: Grade D

v) Percentage >= 40%: Grade E

vi) Percentage < 40%: Grade F

```
try:
    physics = float(input("Enter Physics marks: "))
    chemistry = float(input("Enter Chemistry marks: "))
    biology = float(input("Enter Biology marks: "))
    mathematics = float(input("Enter Mathematics marks: "))
    computer = float(input("Enter Computer marks: "))

    if any(mark < 0 or mark > 100 for mark in [physics, chemistry, biology, mathematics, c
        raise ValueError("Marks should be between 0 and 100.")

    total_marks = physics + chemistry + biology + mathematics + computer
    percentage = (total_marks / 500) * 100

    if percentage >= 90:
        grade = "A"
    elif percentage >= 80:
        grade = "B"
    elif percentage >= 70:
        grade = "C"
    elif percentage >= 60:
        grade = "D"
    elif percentage >= 40:
        grade = "E"
    else:
        grade = "F"
```

```
    print(f"Total Marks: {total_marks}")
    print(f"Percentage: {percentage:.2f}%")
    print(f"Grade: {grade}")

except ValueError as ve:
    print(f"Invalid input: {ve}")
except Exception as e:
    print(f"An error occurred: {str(e)}")
```

```
 Enter Physics marks: 96
 Enter Chemistry marks: 94
 Enter Biology marks: 98
 Enter Mathematics marks: 90
 Enter Computer marks: 100
 Total Marks: 478.0
 Percentage: 95.60%
 Grade: A
```

2. Write a python program with Exception handling to input electricity unit charges and calculate the total electricity bill according to the given condition:

i) For the first 50 units Rs. 0.50/unit

ii) For the next 100 units Rs. 0.75/unit

iii) For the next 100 units Rs. 1.20/unit

iv) For units above 250 Rs. 1.50/unit

v) An additional surcharge of 20% is added to the bill.

```
try:
    units_consumed = float(input("Enter electricity units consumed: "))
    if units_consumed <= 0:
        raise ValueError("Units consumed should be a positive number.")

    if units_consumed <= 50:
        total_bill = units_consumed * 0.50
    elif units_consumed <= 150:
        total_bill = 50 * 0.50 + (units_consumed - 50) * 0.75
    elif units_consumed <= 250:
        total_bill = 50 * 0.50 + 100 * 0.75 + (units_consumed - 150) * 1.20
    else:
        total bill = 50 * 0.50 + 100 * 0.75 + 100 * 1.20 + (units consumed - 250) * 1.50
```

```
    total_bill_with_surcharge = total_bill + (0.20 * total_bill)

    print(f"Total electricity bill: Rs. {total_bill_with_surcharge:.2f}")

except ValueError as ve:
    print(f"Invalid input: {ve}")
except Exception as e:
    print(f"An error occurred: {str(e)}")
```

```
 Enter electricity units consumed: 100
 Total electricity bill: Rs. 75.00
```

3. Write a python program with Exception handling to input the week number and print the weekday.

```
try:
    week_number = int(input("Enter the week number (1-7): "))

    if week_number < 1 or week_number > 7:
        raise ValueError("Week number should be between 1 and 7.")

    weekdays = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday

    print(f"Weekday for week number {week_number} is {weekdays[week_number - 1]}")

except ValueError as ve:
    print(f"Invalid input: {ve}")
except Exception as e:
    print(f"An error occurred: {str(e)}")
```

```
 Enter the week number (1-7): 5
 Weekday for week number 5 is Friday
```

4. Write a Python program to implement word count using command line arguments.

i) Create a text document "apple.txt" whch contains text for wordcount.

ii) Create a wordcount program which calls the "apple.txt" document by opening the file.

iii) If the word is present again in the "aaple.txt",the wordcount is incremented by I until all the

words are counted in the document.

iv) Close the file.

v) Create a command.py program which imports the wordcount.py program. Count the number of words using command line arguments.

vii) Print each word and its count.

```
file = ('command.py','r')
file = ('apple.txt','r')
file = ('wordcount.py','r')
# wordcount.py

def word_count(filename):
    word_counts = {}

    try:
        with open(filename, 'r') as file:
            for line in file:
                words = line.strip().split()
                for word in words:
                    word = word.lower()
                    if word in word_counts:
                        word_counts[word] += 1
                    else:
                        word_counts[word] = 1

    except FileNotFoundError:
        print(f"Error: The file '{filename}' does not exist.")
        return

    return word_counts

if __name__ == "__main__":
    import sys

    if len(sys.argv) != 2:
        print("Usage: python wordcount.py <filename>")
    else:
        filename = sys.argv[1]
        word_counts = word_count(filename)

        if word_counts:
            for word, count in word_counts.items():
                print(f"{word}: {count}")
# command.py

import wordcount
```

```
if __name__ == "__main__":
    import sys

    if len(sys.argv) != 2:
        print("Usage: python command.py <filename>")
    else:
        filename = sys.argv[1]
        word_counts = wordcount.word_count(filename)

        if word_counts:
            for word, count in word_counts.items():
                print(f"{word}: {count}")
```

```
Usage: python wordcount.py <filename>
Usage: python command.py <filename>
```

5. Write a Python program for finding the most frequent words in a text read from a file.

i) Initially open the text file in read mode.

ii) Make all the letters in the document into lowercase letters and split the words in each line.

iii) Get the words in an order.

iv) Sort the words for finding the most frequent words in the file.

v) print the most frequent words in the file

```
# Function to count and return the most frequent words
def most_frequent_words(file_path, num_words):
    try:
        # Open the text file in read mode
        with open(file_path, 'r') as file:
            # Read the content and make it lowercase
            text = file.read().lower()

            # Split the text into words
            words = text.split()

            # Create a dictionary to count word occurrences
            word_count = {}
            for word in words:
```

```
                if word in word_count:
                    word_count[word] += 1
                else:
                    word_count[word] = 1

            # Sort the words by frequency in descending order
            sorted_words = sorted(word_count.items(), key=lambda x: x[1], reverse=True)

            # Print the most frequent words
            for i in range(min(num_words, len(sorted_words))):
                print(f"{sorted_words[i][0]}: {sorted_words[i][1]}")

    except FileNotFoundError:
        print(f"Error: File '{file_path}' not found.")
    except Exception as e:
        print(f"An error occurred: {str(e)}")

# Define the path to your text file and the number of most frequent words to print
file_path = "/content/sample program.txt"
num_most_frequent_words = 10  # Change this to the desired number of most frequent words

# Call the function to find and print the most frequent words
most_frequent_words(file_path, num_most_frequent_words)
```

```
the: 861
lee: 405
of: 377
and: 369
in: 337
to: 308
a: 269
bruce: 231
on: 220
from: 188
```

Double-click (or enter) to edit

6. File Processing with Command-Line Arguments- Scenario: You are developing a command-line utility that processes text files. Users can specify input and output file paths as command-line arguments. Your program should handle exceptions gracefully.

i. Design a Python program that takes two command-line arguments: the input file path and the output file path. Ensure that the program checks if both arguments are provided and that the input file exists.

ii. Implement error handling to deal with scenarios such as missing input files, invalid file paths, or permission issues when writing to the output file.

If an error occurs during file processing, display a user-friendly error message, and exit the program with a non-zero exit code.

iv. Write test cases that cover various scenarios, including providing valid and invalid file paths as command-line arguments

```
file = ('input.txt','r')
file = ('output.txt','r')


import argparse
import os
import shutil

def main(input_file, output_file):
  """
  This program copies the contents of the input file to the output file.

  Args:
    input_file: The path to the input file.
    output_file: The path to the output file.
  """

  # Check if both arguments are provided.
  if not input_file or not output_file:
    print("Error: Both input and output file paths must be provided.")
    exit(1)

  # Check if the input file exists.
  if not os.path.exists(input_file):
    print("Error: Input file does not exist.")
    exit(1)

  # Try to copy the file.
  try:
    shutil.copyfile(input_file, output_file)
  except FileNotFoundError as e:
    print(e)
    exit(1)
  except PermissionError as e:
    print(e)
    exit(1)
```

```
if __name__ == "__main__":
  parser = argparse.ArgumentParser(description="Copy the contents of a file.")
  parser.add_argument("input_file", help="/content/input.py")
  parser.add_argument("output_file", help="/content/output.py")
  args = parser.parse_args()

  main(args.input_file, args.output_file)
```

```
usage: ipykernel_launcher.py [-h] input_file output_file
ipykernel_launcher.py: error: the following arguments are required: output_file
An exception has occurred, use %tb to see the full traceback.

SystemExit: 2
```

SEARCH STACK OVERFLOW