Q11: Defining a constants module

Create a Python module named `constants.py`:

```
# constants.py

pi = 3.141592653589793
speed_of_light = 299792458  # in meters per second
```

Q12: Creating a calculator module

Create a Python module named `calculator.py`:

```
# calculator.py


def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def multiply(a, b):
    return a * b

def divide(a, b):
    if b == 0:
        raise ValueError("Cannot divide by zero")
    return a / b
```

Q13: Implementing an ecommerce package structure

Create the following directory structure for the `ecommerce` package:

```
ecommerce/

    __init__.py
    product_management.py
    order_processing.py
```

Contents of `__init__.py` (this file can be empty):

```
# ecommerce/__init__.py
```

Contents of `product_management.py`:

```
# ecommerce/product_management.py
class Product:
    def __init__(self, product_id, name, price, stock):
        self.product_id = product_id
        self.name = name
        self.price = price
        self.stock = stock

    def update_stock(self, quantity):
        self.stock += quantity

    def update_price(self, new_price):
        self.price = new_price
```

Contents of `order_processing.py`:

```
# ecommerce/order_processing.py

class Order:
    def __init__(self, order_id, product, quantity):
        self.order_id = order_id
        self.product = product
        self.quantity = quantity
        self.status = "Pending"

    def process_order(self):
        if self.product.stock >= self.quantity:
            self.product.stock -= self.quantity
            self.status = "Processed"
            print(f"Order {self.order_id} has been processed.")
        else:
            self.status = "Failed"
            print(f"Order {self.order_id} failed due to insufficient
stock.")
```

Q14: Implementing a string_utils module

Create a Python module named `string_utils.py`:

```
# string_utils.py

def reverse_string(s):
    return s[::-1]

def capitalize_string(s):
    return s.capitalize()
```

Q15: Implementing a file_operations module

Create a Python module named `file_operations.py`:

```
# file_operations.py

def read_file(filename):
    with open(filename, 'r') as file:
        return file.read()

def write_file(filename, content):
    with open(filename, 'w') as file:
        file.write(content)

def append_file(filename, content):
    with open(filename, 'a') as file:
        file.write(content)
```

Q16: Writing employee details to a file

Create a Python program to write employee details to `employees.txt`:

```
# write_employees.py

employees = [
    {"name": "Alice", "age": 30, "salary": 50000},
    {"name": "Bob", "age": 25, "salary": 45000},
    {"name": "Charlie", "age": 35, "salary": 60000},
]
```

```
with open("employees.txt", "w") as file:
    for emp in employees:
        file.write(f"Name: {emp['name']}, Age: {emp['age']}, Salary:
{emp['salary']}\n")
```

Q17: Reading contents of `inventory.txt`

Create a Python script to read and display the contents of `inventory.txt` line by line:

```
# read_inventory.py
with open("inventory.txt", "r") as file:
    for line in file:
        print(line, end="")
```

Q18: Calculating total expenses from `expenses.txt`

Create a Python script to calculate the total amount spent on expenses:

```
# calculate_expenses.py

total_expense = 0
with open("expenses.txt", "r") as file:
    for line in file:
        total_expense += float(line.strip())
print(f"Total expenses: ${total_expense:.2f}")
```

Q19: Counting word occurrences in `paragraph.txt`

Create a Python program to count word occurrences and display the results in alphabetical order:

```
# word_count.py
from collections import Counter
with open("paragraph.txt", "r") as file:
    text = file.read().lower()
words = text.split()
word_count = Counter(words)
for word in sorted(word_count):
    print(f"{word}: {word_count[word]}")
```

Q20: Measure of Central Tendency and Measures of Dispersion

Measure of Central Tendency:

- Measures of central tendency describe the center of a data set.
- Common measures include the mean (average), median (middle value), and mode (most frequent value).

Measures of Dispersion:

- Measures of dispersion describe the spread of a data set.
- Common measures include the range (difference between the highest and lowest values), variance (average of squared deviations from the mean), and standard deviation (square root of variance).

Calculations:

- **Mean**: Sum of all values divided by the number of values.
- **Median**: Middle value when data is sorted.
- **Mode**: Most frequently occurring value.
- **Range**: Difference between maximum and minimum values.
- **Variance**: Average of squared differences from the mean.
- **Standard Deviation**: Square root of variance.

Example of calculating these measures in Python:

```
# central_tendency_dispersion.py
import numpy as np
from scipy import stats

data = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

mean = np.mean(data)
median = np.median(data)
mode = stats.mode(data)[0][0]
range_ = np.ptp(data)
variance = np.var(data)
std_dev = np.std(data)

print(f"Mean: {mean}")
print(f"Median: {median}")
print(f"Mode: {mode}")
print(f"Range: {range_}")
print(f"Variance: {variance}")
print(f"Standard Deviation: {std_dev}")
```