Python Programming Basics for Big Data - Question Bank

CSCI316: Big Data Mining Techniques and Implementation

QUESTION 1: Python Fundamentals and Data Structures (8 marks)

Part A (3 marks)

Explain why Python is preferred for data science and big data applications. Discuss at least three advantages and one major drawback compared to languages like Java or C.

Part B (5 marks)

Given the following Python code, trace through the execution and show the output at each step:

```python
data = [3, 1, 4, 1, 5, 9, 2, 6]
processed = []

for i, value in enumerate(data):
    if value % 2 == 0:
        processed.append(value * 2)
    else:
        processed.append(value + 1)
    print(f"Step {i+1}: processed = {processed}")

final_result = sum(processed) / len(processed)
print(f"Final average: {final_result}")
```

Show your working for each iteration and calculate the final average manually.

QUESTION 2: Lists, Tuples, and Advanced Data Structures (10 marks)

Part A (4 marks)

Write a Python function `analyze_student_grades(grades_list)` that takes a list of student grades and returns a tuple containing:

- The highest grade
- The lowest grade
- The average grade (rounded to 2 decimal places)
- A list of grades above average

Example:

```python
grades = [78, 92, 85, 67, 94, 88, 76, 90]
result = analyze_student_grades(grades)
# Should return: (94, 67, 83.75, [92, 85, 94, 88, 90])
```

Part B (3 marks)

Explain the difference between lists and tuples in Python. Provide a practical example where you would choose tuples over lists and justify your choice.

Part C (3 marks)

Given the following nested list representing a matrix:

```python
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Write code to:

1. Extract the diagonal elements (1, 5, 9)
2. Calculate the sum of each row
3. Create a flattened version of the matrix

QUESTION 3: Dictionaries and Sets in Data Processing (9 marks)

Part A (5 marks)

You are analyzing social media data. Write a Python function `analyze_hashtags(tweets)` that processes a list of tweet dictionaries and returns analytics about hashtag usage.

```python
tweets = [
    {"user": "alice", "text": "Learning #python #datascience", "hashtags": ["#python", "#datascience"]},
    {"user": "bob", "text": "Big data with #spark #python", "hashtags": ["#spark", "#python"]},
    {"user": "carol", "text": "Machine learning #ML #python #datascience", "hashtags": ["#ML", "#python", "#datas
]
```

Your function should return a dictionary containing:

- `total_hashtags`: Total number of hashtag occurrences
- `unique_hashtags`: Number of unique hashtags

- `most_popular`: The most frequently used hashtag
- `hashtag_counts`: A dictionary with hashtag frequencies

Part B (4 marks)

Explain the performance difference between checking membership in a list versus a set. Provide a code example demonstrating this difference using a dataset of 1000 stopwords. Calculate the time complexity for both approaches.

QUESTION 4: List Comprehensions and Functional Programming (8 marks)

Part A (4 marks)

Transform the following traditional loops into equivalent list comprehensions:

```python
# Traditional approach
result1 = []
for x in range(10):
    if x % 3 == 0 and x != 0:
        result1.append(x ** 2)

# Traditional approach
result2 = []
for i in range(5):
    row = []
    for j in range(5):
        if i == j:
            row.append(1)
        else:
            row.append(0)
    result2.append(row)
```

Part B (4 marks)

Given a list of student records, use map and filter functions to:

1. Filter students with grades above 80
2. Extract only the names of these high-performing students
3. Convert all names to uppercase

```python
```

```
students = [
    {"name": "Alice", "grade": 85, "subject": "Math"},
    {"name": "Bob", "grade": 92, "subject": "Physics"},
    {"name": "Carol", "grade": 78, "subject": "Chemistry"},
    {"name": "David", "grade": 88, "subject": "Biology"}
]
```

Show both the functional programming approach and an equivalent list comprehension solution.

QUESTION 5: Object-Oriented Programming for Data Structures (10 marks)

Part A (6 marks)

Implement a `DataProcessor` class for handling student enrollment data with the following requirements:

python

```python
class DataProcessor:
    def __init__(self):
        # Initialize with empty student records
        pass

    def add_student(self, student_id, name, courses):
        # Add a new student with their enrolled courses
        pass

    def get_student_courses(self, student_id):
        # Return list of courses for a student
        pass

    def get_course_enrollment(self, course_name):
        # Return list of students enrolled in a course
        pass

    def get_statistics(self):
        # Return dictionary with total students, total courses,
        # and average courses per student
        pass
```

Part B (4 marks)

Create a test scenario that demonstrates all methods of your `DataProcessor` class:

- Add at least 4 students with different course combinations
- Show how to retrieve student courses and course enrollments
- Display the enrollment statistics

Calculate the statistics manually to verify your implementation.

QUESTION 6: NumPy for Big Data Processing (12 marks)

Part A (4 marks)

Explain the advantages of NumPy arrays over Python lists for big data processing. Discuss memory efficiency, performance, and vectorized operations with examples.

Part B (4 marks)

Given two matrices representing student scores in different subjects:

```python
import numpy as np
math_scores = np.array([[85, 92, 78], [90, 87, 95], [82, 89, 91]])
science_scores = np.array([[88, 85, 90], [92, 91, 89], [86, 93, 87]])
```

Where each row represents a student and each column represents a test. Calculate:

1. The average score for each student across both subjects
2. The overall class average for each subject (Math and Science)
3. Create a combined performance matrix showing the difference between each student's math and science scores
4. Identify which students performed better in math versus science

Show all calculations step by step.

Part C (4 marks)

Write a NumPy-based function `analyze_grade_distribution(grades_matrix)` that:

- Calculates mean, median, and standard deviation for the entire dataset
- Finds the transpose of the matrix to analyze by test instead of by student
- Uses boolean indexing to find all scores above the 75th percentile
- Returns a summary dictionary with all statistics

Test your function with the provided math_scores matrix and verify results manually.

ADDITIONAL PRACTICE QUESTIONS

Question 7: Data Type Operations (6 marks)

Given a mixed dataset containing student information:

```python
student_data = "Alice,85,Math;Bob,92,Physics;Carol,78,Chemistry;David,88,Biology"
```

Write a complete solution to:

1. Parse this string into a structured format
2. Convert grades to appropriate numeric types
3. Group students by grade ranges (A: 90+, B: 80-89, C: 70-79)
4. Calculate statistics for each grade group

Question 8: Control Flow and Error Handling (7 marks)

Implement a robust grade calculator that:

- Handles various input formats (strings, integers, floats)
- Validates that grades are between 0 and 100
- Provides appropriate error messages for invalid inputs
- Uses exception handling to manage conversion errors
- Implements a retry mechanism for invalid inputs

Question 9: Advanced String Processing (5 marks)

Process a dataset of course descriptions to extract key information:

```python
courses = [
    "CSCI316: Big Data Mining - 3 credits - Prerequisites: CSCI201, MATH180",
    "PHYS201: Classical Mechanics - 4 credits - Prerequisites: MATH150",
    "CHEM301: Organic Chemistry - 3 credits - Prerequisites: CHEM101, CHEM102"
]
```

Extract and structure: course codes, titles, credit hours, and prerequisites.

MARKING RUBRIC GUIDELINES

Code Quality (Applied to all programming questions):

- **Excellent (90-100%):** Clean, efficient code with proper variable names and comments
- **Good (80-89%):** Functional code with minor style issues
- **Satisfactory (70-79%):** Working code with some inefficiencies
- **Needs Improvement (<70%):** Incomplete or non-functional code

Problem-Solving Approach:

- **Excellent:** Clear understanding of problem, optimal solution approach
- **Good:** Correct solution with minor logical gaps
- **Satisfactory:** Basic solution that meets requirements
- **Needs Improvement:** Incomplete understanding or incorrect approach

Manual Calculations:

- **Full marks:** All steps shown with correct arithmetic
- **Partial marks:** Correct method but minor calculation errors
- **Minimal marks:** Understanding shown but significant errors

---

EXAM PREPARATION TIPS

1. **Practice Manual Calculations:** Be prepared to trace through code execution step by step
2. **Understand Time Complexity:** Know the performance characteristics of different data structures
3. **Master Data Structure Selection:** Understand when to use lists vs tuples vs sets vs dictionaries
4. **NumPy Operations:** Practice matrix operations and array manipulations without calculators
5. **Code Implementation:** Be able to write functions from scratch without library dependencies
6. **Real-world Applications:** Practice with realistic datasets and scenarios
7. **Error Handling:** Include appropriate validation and error checking in your code

Time Management: Allocate approximately 30 minutes per question, leaving time for review and manual calculation verification.