Python Programming Mock Exam

Comprehensive Assessment Based on Course Materials

Author: Manus AI
Date: December 2024
Duration: 120 minutes
Total Points: 100 points

Instructions

1. **Write all code in Python 3**
2. **Show your work clearly** - partial credit may be awarded
3. **Use proper Python syntax** and indentation
4. **Comment your code** where appropriate
5. **Test your solutions** mentally before finalizing
6. **Read each question carefully** before answering

Part I: Fundamentals and Control Structures (30 points)

Task 1: Basic Python Operations (8 points)

Subtask 1.1 (3 points)

Write a function `calculate_grade_statistics(scores)` that takes a list of exam scores and returns a dictionary containing: - `'average'`: the average score - `'highest'`: the highest score
- `'lowest'`: the lowest score - `'passing_count'`: number of scores >= 60

```
def calculate_grade_statistics(scores):
    # Your code here
    pass

# Example usage:
# scores = [85, 92, 78, 65, 88, 45, 90]
# result = calculate_grade_statistics(scores)
```

```
# Expected output: {'average': 77.57, 'highest': 92, 'lowest':
45, 'passing_count': 5}
```

Subtask 1.2 (3 points)

Create a function `validate_email(email)` that checks if an email address is valid according to these rules: - Must contain exactly one '@' symbol - Must have at least one character before and after '@' - Must end with a valid domain extension (.com, .org, .edu, .net) - Return `True` if valid, `False` otherwise

```
def validate_email(email):
    # Your code here
    pass
```

Subtask 1.3 (2 points)

Write a function that converts temperature between Celsius and Fahrenheit. The function should accept a temperature value and a scale ('C' or 'F') and return the converted temperature in the other scale.

```
def convert_temperature(temp, scale):
    # Your code here
    # Formula: F = (C * 9/5) + 32, C = (F - 32) * 5/9
    pass
```

Task 2: Control Structures and Loops (12 points)

Subtask 2.1 (4 points)

Write a function `print_pattern(n)` that prints a number pyramid pattern. For example, if n=4:

```
   1
  121
 12321
1234321
```

```
def print_pattern(n):
    # Your code here
    pass
```

Subtask 2.2 (4 points)

Create a function `find_prime_numbers(start, end)` that returns a list of all prime numbers between start and end (inclusive). A prime number is only divisible by 1 and itself.

```
def find_prime_numbers(start, end):
    # Your code here
    pass

# Example: find_prime_numbers(10, 30) should return [11, 13, 17,
19, 23, 29]
```

Subtask 2.3 (4 points)

Implement a function `guess_number_game()` that: 1. Generates a random number between 1 and 100 2. Asks the user to guess the number 3. Provides hints ("too high" or "too low") 4. Counts the number of attempts 5. Congratulates when correct and shows the number of attempts

```
import random

def guess_number_game():
    # Your code here
    pass
```

Task 3: String Processing (10 points)

Subtask 3.1 (4 points)

Write a function `analyze_text(text)` that analyzes a text string and returns a dictionary with: - `'word_count'`: number of words - `'char_count'`: number of characters (excluding spaces) - `'vowel_count'`: number of vowels (a, e, i, o, u) - `'most_common_char'`: the most frequently occurring character (excluding spaces)

```
def analyze_text(text):
    # Your code here
    pass
```

Subtask 3.2 (3 points)

Create a function `format_phone_number(phone)` that takes a 10-digit phone number string and formats it as "(XXX) XXX-XXXX". Handle various input formats (with/without spaces, dashes, parentheses).

```
def format_phone_number(phone):
    # Your code here
    pass

# Examples:
# format_phone_number("1234567890") → "(123) 456-7890"
# format_phone_number("123-456-7890") → "(123) 456-7890"
```

Subtask 3.3 (3 points)

Write a function `is_palindrome_sentence(sentence)` that checks if a sentence is a palindrome, ignoring spaces, punctuation, and case.

```
def is_palindrome_sentence(sentence):
    # Your code here
    pass

# Example: "A man a plan a canal Panama" should return True
```

Part II: Functions, OOP, and Advanced Concepts (40 points)

Task 4: Advanced Functions and Recursion (15 points)

Subtask 4.1 (5 points)

Implement a recursive function `fibonacci_sequence(n)` that returns the first n numbers in the Fibonacci sequence as a list. Also implement an iterative version for comparison.

```
def fibonacci_sequence_recursive(n):
    # Your recursive implementation here
    pass

def fibonacci_sequence_iterative(n):
```

```
    # Your iterative implementation here
    pass
```

Subtask 4.2 (5 points)

Create a function `process_data(*args, **kwargs)` that: - Accepts any number of positional arguments (numbers) - Accepts keyword arguments for operation type and formatting - Supports operations: 'sum', 'product', 'average', 'max', 'min' - Returns formatted result based on 'format' parameter ('int', 'float', 'string')

```
def process_data(*args, **kwargs):
    # Your code here
    pass

# Examples:
# process_data(1, 2, 3, 4, operation='sum', format='string') →
"Sum: 10"
# process_data(2, 4, 6, operation='average', format='float') →
4.0
```

Subtask 4.3 (5 points)

Write a decorator function `timing_decorator` that measures and prints the execution time of any function it decorates.

```
import time

def timing_decorator(func):
    # Your code here
    pass

@timing_decorator
def slow_function():
    time.sleep(1)
    return "Done"

# When called, should print something like: "slow_function took
1.0023 seconds"
```

Task 5: Object-Oriented Programming (25 points)

Subtask 5.1 (10 points)

Create a `Library` class with the following specifications:

Class variables: - `total_books`: tracks total number of books across all libraries - `library_count`: tracks number of library instances

Instance variables: - `name`: library name - `books`: list of books (initially empty) - `library_id`: unique ID for each library

Methods: - `add_book(title, author, isbn)`: adds a book dictionary to the library - `remove_book(isbn)`: removes a book by ISBN - `find_books_by_author(author)`: returns list of books by given author - `get_library_info()`: returns formatted string with library details

```
class Library:
    # Your implementation here
    pass
```

Subtask 5.2 (8 points)

Create a `BankAccount` class with the following features:

Instance variables: - `account_number`: unique account identifier - `account_holder`: name of account holder - `_balance`: private balance (use property for access)

Methods: - `deposit(amount)`: add money to account - `withdraw(amount)`: remove money (check sufficient funds) - `transfer(amount, target_account)`: transfer money to another account - Property `balance`: getter for balance (read-only)

Special methods: - `__str__`: return formatted account information - `__eq__`: compare accounts by account number

```
class BankAccount:
    # Your implementation here
    pass
```

Subtask 5.3 (7 points)

Create an inheritance hierarchy with a base `Vehicle` class and derived `Car` and `Motorcycle` classes:

Vehicle class: - Attributes: `make`, `model`, `year`, `fuel_level` - Methods: `start_engine()`, `stop_engine()`, `refuel(amount)`

Car class (inherits from Vehicle): - Additional attribute: `num_doors` - Override `start_engine()` to include door check

Motorcycle class (inherits from Vehicle): - Additional attribute: `has_sidecar` - Override `start_engine()` to include safety check

```
class Vehicle:
    # Your implementation here
    pass

class Car(Vehicle):
    # Your implementation here
    pass

class Motorcycle(Vehicle):
    # Your implementation here
    pass
```

Part III: Data Structures and Advanced Topics (30 points)

Task 6: List Comprehensions and Lambda Functions (12 points)

Subtask 6.1 (4 points)

Convert the following for-loop code into list comprehensions:

```
# Original code:
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
result1 = []
for num in numbers:
    if num % 2 == 0:
        result1.append(num ** 2)

# Your list comprehension:
result1 = # Your code here

# Original code:
words = ["hello", "world", "python", "programming"]
result2 = []
for word in words:
    if len(word) > 5:
        result2.append(word.upper())
```

```
# Your list comprehension:
result2 = # Your code here
```

Subtask 6.2 (4 points)

Use `map()`, `filter()`, and lambda functions to solve these problems:

```
# Given list of dictionaries
students = [
    {"name": "Alice", "grade": 85, "age": 20},
    {"name": "Bob", "grade": 92, "age": 19},
    {"name": "Charlie", "grade": 78, "age": 21},
    {"name": "Diana", "grade": 96, "age": 20}
]

# 1. Extract all names using map and lambda
names = # Your code here

# 2. Filter students with grade >= 90 using filter and lambda
high_achievers = # Your code here

# 3. Create list of formatted strings "Name: Grade" using map
and lambda
formatted_grades = # Your code here
```

Subtask 6.3 (4 points)

Create a nested list comprehension that generates a 5x5 multiplication table as a list of lists:

```
# Expected output: [[1, 2, 3, 4, 5], [2, 4, 6, 8, 10], [3, 6, 9,
12, 15], ...]
multiplication_table = # Your code here
```

Task 7: File Operations and Exception Handling (10 points)

Subtask 7.1 (5 points)

Write a function `safe_file_operations(filename, data)` that: 1. Safely writes data to a file 2. Handles potential exceptions (FileNotFoundError, PermissionError, etc.) 3. Returns a tuple (success: bool, message: str) 4. Uses proper file handling with context managers

```
def safe_file_operations(filename, data):
    # Your code here
    pass
```

Subtask 7.2 (5 points)

Create a function `process_csv_data(filename)` that: 1. Reads a CSV file with student data (name, grade1, grade2, grade3) 2. Calculates average grade for each student 3. Returns a dictionary with student names as keys and averages as values 4. Handles file errors gracefully

```
def process_csv_data(filename):
    # Your code here
    # Assume CSV format: name,grade1,grade2,grade3
    pass
```

Task 8: Modules and Type Annotations (8 points)

Subtask 8.1 (4 points)

Add appropriate type annotations to this function:

```
def calculate_statistics(data, operation):
    """
    Calculate statistics on a list of numbers.

    Args:
        data: List of numbers
        operation: String indicating operation ('mean',
'median', 'mode')

    Returns:
        Calculated statistic value or None if invalid operation
    """
    # Add type annotations to the function signature
    if operation == 'mean':
        return sum(data) / len(data)
    elif operation == 'median':
        sorted_data = sorted(data)
        n = len(sorted_data)
        if n % 2 == 0:
            return (sorted_data[n//2-1] + sorted_data[n//2]) / 2
        else:
            return sorted_data[n//2]
    elif operation == 'mode':
        from collections import Counter
```

```
        counts = Counter(data)
        return counts.most_common(1)[0][0]
    else:
        return None
```

Subtask 8.2 (4 points)

Create a simple module structure. Write the import statements and function calls to use a hypothetical `math_utils` module that contains functions `factorial(n)` and `is_prime(n)`:

```
# Your import statements here

# Calculate factorial of 5
fact_5 = # Your code here

# Check if 17 is prime
is_17_prime = # Your code here

# Import only the factorial function with an alias
# Your import statement here

# Use the aliased function
result = # Your code here
```

Bonus Question (5 points)

Create a context manager class `TimedOperation` that can be used with the `with` statement to measure the execution time of a code block. The context manager should print the elapsed time when the block completes.

```
class TimedOperation:
    # Your implementation here
    pass

# Usage example:
# with TimedOperation("Database query"):
#     time.sleep(2)  # Simulated operation
# Should print: "Database query completed in 2.00 seconds"
```

Answer Key Summary

Part I (30 points): - Task 1: Basic operations and validation functions - Task 2: Control structures, loops, and algorithms
- Task 3: String processing and analysis

Part II (40 points): - Task 4: Advanced functions, recursion, decorators - Task 5: Object-oriented programming with inheritance

Part III (30 points): - Task 6: List comprehensions and functional programming - Task 7: File operations and exception handling - Task 8: Modules and type annotations

Bonus (5 points): - Context manager implementation

Total: 105 points (100 + 5 bonus)

Grading Rubric

- **Syntax and Logic (60%)**: Correct Python syntax and logical implementation
- **Code Quality (20%)**: Proper naming, comments, and structure
- **Error Handling (10%)**: Appropriate exception handling where required
- **Efficiency (10%)**: Reasonable algorithmic approach and performance

Good luck with your exam!