Python Programming Mock Exam

Comprehensive Assessment - Pen and Paper Format

Name: _____ **Student ID:** ___

Date: _____ **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. **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):
    _____

    _____

    _____

    _____

    _____
```

```
    _____

    _____

    _____
```

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):
    _____

    _____

    _____

    _____

    _____

    _____
```

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):
    # Formula: F = (C * 9/5) + 32, C = (F - 32) * 5/9
    _____

    _____

    _____

    _____
```

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):
    _____
    _____
    _____
    _____
    _____
    _____
    _____
    _____
```

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):
    _____
    _____
    _____
    _____
    _____
```

```
    _____

    _____

    _____

    _____

    _____
```

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():
    _____

    _____

    _____

    _____

    _____

    _____

    _____

    _____

    _____

    _____
```

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):
    _____
    _____
    _____
    _____
    _____
    _____
    _____
    _____
    _____
    _____
```

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):
    _____
    _____
    _____
    _____
    _____
    _____
```

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):
    _____
    _____
    _____
    _____
    _____
```

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):
    _____
    _____
    _____
    _____
    _____
    _____
def fibonacci_sequence_iterative(n):
    _____
    _____
    _____
    _____
    _____
```

```
    ____________________________________________________________________
```

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):
    ____________________________________________________________________
    ____________________________________________________________________
    ____________________________________________________________________
    ____________________________________________________________________
    ____________________________________________________________________
    ____________________________________________________________________
    ____________________________________________________________________
    ____________________________________________________________________
    ____________________________________________________________________
    ____________________________________________________________________
```

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):
    ____________________________________________________________________
    ____________________________________________________________________
    ____________________________________________________________________
    ____________________________________________________________________
```

```
    _____

    _____

    _____

    _____
```

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:
    _____

    _____

    def __init__(self, name):

_____

_____

_____

_____

    def add_book(self, title, author, isbn):

_____
```

```
_____

_____

_____
    def remove_book(self, isbn):
_____

_____

_____

_____

_____
    def find_books_by_author(self, author):
_____

_____

_____

_____
    def get_library_info(self):
_____

_____
```

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:
    def __init__(self, account_number, account_holder,
initial_balance=0):

_____

_____

_____

    @property
    def balance(self):

_____

    def deposit(self, amount):

_____

_____

_____

    def withdraw(self, amount):

_____

_____

_____

_____

    def transfer(self, amount, target_account):

_____
```

```
_____

_____

    def __str__(self):

_____

    def __eq__(self, other):

_____

_____
```

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:
    def __init__(self, make, model, year):

_____

_____

_____

_____

    def start_engine(self):
```

```
    def stop_engine(self):
```

```
    def refuel(self, amount):
```

```
class Car(Vehicle):
    def __init__(self, make, model, year, num_doors):
```

```
    def start_engine(self):
```

```
class Motorcycle(Vehicle):
    def __init__(self, make, model, year, has_sidecar=False):
```

```
    def start_engine(self):
```

```
_____


_____
```

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 =
_____

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

# Your list comprehension:
result2 =
_____
```

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 =
_____

# 2. Filter students with grade >= 90 using filter and lambda
high_achievers =
_____

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

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 =
_____

_____
```

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):
    _____

    _____

    _____

    _____
```

```
    _____

    _____

    _____

    _____

    _____

    _____
```

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):
    # Assume CSV format: name,grade1,grade2,grade3
    _____

    _____

    _____

    _____

    _____

    _____

    _____

    _____

    _____

    _____
```

Task 8: Modules and Type Annotations (8 points)

Subtask 8.1 (4 points)

Add appropriate type annotations to this function signature:

```
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
    """
    # Write the function signature with type annotations:
    _____________________________________________________________


    _____________________________________________________________
```

Subtask 8.2 (4 points)

Create 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 =
_____________________________________________________________

# Check if 17 is prime
is_17_prime =
_____________________________________________________________

# Import only the factorial function with an alias
_____________________________________________________________

# Use the aliased function
result =
_____________________________________________________________
```

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:
    def __init__(self, operation_name):

_____

_____

    def __enter__(self):

_____

_____

_____

_____

    def __exit__(self, exc_type, exc_value, traceback):

_____

_____

_____

_____

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

Scratch Work Space

Use this space for any calculations, planning, or rough work:

End of Exam

Total Points: 105 (100 + 5 bonus)

Good luck!