Functions & Modular Code Practice

Python Course · Module 5 Practice

ElnurBDa

Practice 1 — Function Library

Create a project with the following structure:

```
practice1_functions/
├── main.py
└── math_utils.py
```

Requirements:

1. In `math_utils.py`, create these functions:
   - `calculate_area(length, width)` - returns the area of a rectangle
   - `calculate_perimeter(length, width)` - returns the perimeter of a rectangle
   - `is_even(number)` - returns `True` if number is even, `False` otherwise
   - `factorial(n)` - calculates and returns the factorial of n (n! = n × (n-1) × ... × 1)
   - `power(base, exponent)` - returns base raised to the power of exponent
2. In `main.py`:
   - Import all functions from `math_utils`
   - Ask the user for two numbers (length and width)
   - Call `calculate_area()` and `calculate_perimeter()` and display results
   - Ask the user for a number and check if it's even using `is_even()`
   - Ask the user for a number and calculate its factorial
   - Ask the user for base and exponent, then calculate power

Example interaction:

```
Enter length: 5
Enter width: 3
Area: 15
Perimeter: 16

Enter a number: 8
Is even: True

Enter a number for factorial: 5
Factorial of 5: 120

Enter base: 2
Enter exponent: 8
2^8 = 256
```

Practice 2 — Text Processing Functions

Create `text_processor.py` that contains:

1. **Functions to create:**
   - `count_words(text)` - returns the number of words in text
   - `count_vowels(text)` - returns the number of vowels (a, e, i, o, u, case-insensitive)
   - `reverse_text(text)` - returns the text reversed
   - `capitalize_words(text)` - returns text with each word capitalized (first letter uppercase)
   - `remove_spaces(text)` - returns text with all spaces removed
2. **Main program:**
   - Ask the user to enter a sentence
   - Call each function with the user's input
   - Display all results in a formatted report
3. **Bonus:** Create a function `analyze_text(text)` that returns a dictionary with all statistics:

```
{
    "word_count": 5,
    "vowel_count": 8,
    "reversed": "...",
    "capitalized": "...",
    "no_spaces": "..."
}
```

Example interaction:

```
Enter a sentence: hello world python
========================================
Text Analysis:
========================================
Word count: 3
Vowel count: 4
Reversed: nohtyp dlrow olleh
Capitalized: Hello World Python
No spaces: helloworldpython
```

Practice 3 – Student Management System (Functions)

Create `student_manager.py` that uses functions to manage student data:

1. **Create these functions:**
   - `add_student(students_dict, name, grades)` - adds a student with name and list of grades to dictionary
   - `calculate_average(grades)` - takes a list of grades and returns the average
   - `get_letter_grade(average)` - takes average and returns letter grade (A: 90+, B: 80-89, C: 70-79, D: 60-69, F: <60)
   - `find_top_student(students_dict)` - returns the name of student with highest average
   - `get_students_above_threshold(students_dict, threshold)` - returns list of student names with average above threshold
2. **Main program:**
   - Initialize an empty dictionary `students = {}`
   - Use a `while` loop with menu:
     - "1. Add student"
     - "2. View all students"
     - "3. Find top student"
     - "4. Find students above threshold"
     - "5. Exit"
   - For each menu option, call the appropriate functions

Example interaction:

```
Menu:
1. Add student
2. View all students
3. Find top student
4. Find students above threshold
5. Exit
Choice: 1
Student name: Alice
Enter grades (comma-separated): 85,90,88
Student added!

Choice: 2
All Students:
Alice: Average = 87.67, Grade = B

Choice: 3
Top student: Alice (Average: 87.67)

Choice: 4
Enter threshold: 85
Students above 85: ['Alice']
```

Note: For comma-separated grades input, you can use:

```
grades_str = input("Enter grades (comma-separated): ")
grades = [int(x.strip()) for x in grades_str.split(",")]
```

Practice 4 — Lambda Functions & Higher-Order Functions

Create `lambda_practice.py` that demonstrates lambda functions and higher-order operations:

1. **Create a list of numbers:**

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

2. **Using lambda functions, create:**
   - A lambda to square a number
   - A lambda to check if a number is even
   - A lambda to check if a number is greater than 5
3. **Using these lambdas with list comprehensions or loops:**
   - Create a list of squared numbers
   - Filter even numbers
   - Filter numbers greater than 5
   - Create a list of numbers that are both even AND greater than 5
4. **Create regular functions that do the same:**
   - `square_number(x)` - returns x squared
   - `is_even_num(x)` - returns True if x is even
   - `is_greater_than_five(x)` - returns True if x > 5
5. **Compare the results** - show that both lambda and regular functions produce the same results

Example output:

```
Original numbers: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Using Lambda Functions:
Squared: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
Even numbers: [2, 4, 6, 8, 10]
Greater than 5: [6, 7, 8, 9, 10]
Even AND > 5: [6, 8, 10]

Using Regular Functions:
Squared: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
Even numbers: [2, 4, 6, 8, 10]
Greater than 5: [6, 7, 8, 9, 10]
Even AND > 5: [6, 8, 10]
```

Practice 5 — Bank Account Class

Create `bank_account.py` with a `BankAccount` class:

1. **Class Requirements:**
   - `__init__(self, account_holder, initial_balance=0)` - initialize account with holder name and balance
   - `deposit(self, amount)` - adds amount to balance, returns new balance
   - `withdraw(self, amount)` - subtracts amount from balance if sufficient funds, returns new balance or None if insufficient
   - `get_balance(self)` - returns current balance
   - `display_info(self)` - prints formatted account information
2. **Main program:**
   - Create at least 2 `BankAccount` objects with different initial balances
   - Perform multiple deposits and withdrawals
   - Display account information after each operation
   - Show that each account maintains its own balance
3. **Bonus:** Add a class attribute `total_accounts` that counts how many accounts have been created

Example interaction:

```
Creating account for Alice with $100...
Account created!

Depositing $50 to Alice's account...
New balance: $150.00

Withdrawing $75 from Alice's account...
New balance: $75.00

Attempting to withdraw $100 from Alice's account...
Insufficient funds! Current balance: $75.00

Account Information:
====================
Account Holder: Alice
Balance: $75.00

Creating account for Bob with $200...
Account created!

Bob's balance: $200.00
Alice's balance: $75.00
(Each account is independent!)
```

Mini Project – Library Management System

Create a project with the following structure:

```
library_system/
├── main.py
├── book.py
└── library.py
```

Requirements:

1. **In** `book.py`**, create a** `Book` **class:**
   - Attributes: `title`, `author`, `isbn`, `available` (boolean)
   - `__init__(self, title, author, isbn)` - initialize book
   - `checkout(self)` - marks book as unavailable if available, returns True/False
   - `return_book(self)` - marks book as available
   - `get_info(self)` - returns formatted string with book details
2. **In** `library.py`**, create a** `Library` **class:**
   - Attributes: `books` (list of Book objects), `name`
   - `__init__(self, name)` - initialize library with name
   - `add_book(self, book)` - adds a Book object to the library
   - `find_book(self, title)` - returns Book object if found, None otherwise
   - `list_available_books(self)` - returns list of available books
   - `checkout_book(self, title)` - finds book and checks it out, returns success message
   - `return_book(self, title)` - finds book and returns it, returns success message
   - `display_all_books(self)` - prints all books with their status
3. **In** `main.py`**:**
   - Create a Library instance
   - Create several Book instances
   - Add books to library
   - Implement a menu system:
     - "1. Add book"
     - "2. List all books"
     - "3. List available books"
     - "4. Checkout book"
     - "5. Return book"
     - "6. Exit"
   - Use functions to handle menu operations (e.g., `handle_add_book(library)`, `handle_checkout(library)`)

Example interaction:

```
Welcome to Python Library System!

Menu:
1. Add book
2. List all books
3. List available books
4. Checkout book
5. Return book
6. Exit
```

Thanks!

By ElnurBDa