Control Flow Practice

Python Course · Module 4 Practice

ElnurBDa

Practice 1 — Password Validator

Create password_validator.py that:

1. Asks the user to enter a password.
2. Validates the password using these rules:
   - At least 8 characters long
   - Contains at least one uppercase letter
   - Contains at least one lowercase letter
   - Contains at least one digit
   - Contains at least one special character (!@#$%^&*)
3. Uses loops and conditionals to check each requirement.
4. Prints clear feedback for each failed rule.
5. Only accepts the password if **all** rules pass.

Example interaction:

```
Enter password: abc123
Password too short (minimum 8 characters)
Missing uppercase letter
Missing special character
Password rejected!

Enter password: SecurePass123!
Password accepted!
```

Practice 2 — Shopping Cart Manager

Create shopping_cart.py that:

1. Uses a **dictionary** to store items and their prices:

```
cart = {"apple": 1.50, "bread": 2.00, "milk": 3.50}
```

2. Implements a menu system using a while loop:
   - "1. Add item"
   - "2. Remove item"
   - "3. View cart"
   - "4. Calculate total"
   - "5. Exit"
3. For "Add item": asks for item name and price, adds to dictionary.
4. For "Remove item": asks for item name, removes if exists.
5. For "View cart": displays all items with prices formatted to 2 decimals.
6. For "Calculate total": sums all prices and prints the total.
7. Uses break to exit the loop when user chooses "5".

Example interaction:

```
Menu:
1. Add item
2. Remove item
3. View cart
4. Calculate total
5. Exit
Choice: 1
Item name: orange
Price: 2.25
Item added!

Choice: 3
Cart:
apple: $1.50
bread: $2.00
milk: $3.50
orange: $2.25

Choice: 4
Total: $9.25
```

Practice 3 — Number Pattern Generator

Create pattern_generator.py that:

1. Asks the user for a number n (between 1 and 20).
2. Validates the input (must be between 1 and 20).
3. Uses nested for loops to generate patterns:
3. **Pattern A** (right-aligned triangle):

```
*
**
***
****
*****
```

Pattern B (number pyramid):

```
1
22
333
4444
55555
```

4. Prints both patterns one after another using nested loops.
5. Uses range() for loop control.

Practice 4 – Student Grade Analyzer

Create grade_analyzer.py that:

1. Uses a **dictionary** to store student names and their list of grades:

```
students = {
      "Alice": [85, 90, 78],
      "Bob": [92, 88, 95],
      "Charlie": [70, 75, 80]
}
```

2. Loops through each student in the dictionary.
3. For each student:
   - Calculates their average grade (sum of grades divided by count)
   - Determines their letter grade using if/elif/else:
     - "A" for 90+, "B" for 80-89, "C" for 70-79, "D" for 60-69, "F" for <60
   - Prints a formatted report line
4. Uses a loop to find and print which students have an average above 85.
5. Use list comprehensions where appropriate (e.g., filtering high scores).

Example output:

```
Student Report:
==============
Alice: Average = 84.33, Grade = B
Bob: Average = 91.67, Grade = A
Charlie: Average = 75.00, Grade = C

High Performers (Average > 85):
- Bob
```

Practice 5 – Text Statistics Tool

Create `text_stats.py` that:

1. Asks the user to enter a paragraph of text (multiple sentences).
2. Uses loops and conditionals to analyze the text:
   - Count words (split by spaces)
   - Count sentences (ends with `.`, `!`, `?`)
   - Count vowels (a, e, i, o, u, case-insensitive)
   - Find the longest word
   - Count characters (with and without spaces)
3. Uses loops to iterate through characters and words.
4. Uses conditionals to check for sentence endings and vowels.
5. Stores results in variables and prints a formatted statistics report.

Example:

```
Enter text: Hello world! How are you? I am fine.
========================================
Text Statistics:
========================================
Total words: 7
Total sentences: 3
Total vowels: 12
Longest word: world
Characters (with spaces): 33
Characters (without spaces): 30
```

Practice 6 — Set Operations & Data Filtering

Create `data_filter.py` that:

1. Creates two sets of user IDs:

```python
active_users = {101, 102, 103, 104, 105}
premium_users = {103, 104, 106, 107}
```

2. Uses set operations to find:
   - Users who are both active AND premium (intersection)
   - All unique users (union)
   - Active users who are NOT premium (difference)
   - Users who are in one set but not both (symmetric difference)
3. Creates a dictionary `users` with structure: `{user_id: {"name": str, "age": int}}`
4. Uses a loop with conditionals to filter users by age (between 20 and 28 inclusive).
5. Stores filtered user IDs in a list and prints formatted results for all operations.

Example:

```python
users = {
    101: {"name": "Alice", "age": 25},
    102: {"name": "Bob", "age": 30},
    103: {"name": "Charlie", "age": 22}
}
```

Example output:

```
Set Operations:
===============
Active AND Premium: {103, 104}
All Users: {101, 102, 103, 104, 105, 106, 107}
Active but NOT Premium: {101, 102, 105}
Symmetric Difference: {101, 102, 105, 106, 107}

Age Filter (20-28):
===================
Users aged 20-28: [101, 103]
```

Mini Project — Personal Task Manager

Create task_manager.py that combines multiple concepts:

1. Use a **list of dictionaries** to store tasks:

```
tasks = [
      {"id": 1, "title": "Learn Python", "priority": "high", "completed": False},
      {"id": 2, "title": "Exercise", "priority": "medium", "completed": True}
]
```

2. Implement a menu-driven system with a while loop:
   - "1. Add task"
   - "2. Mark task as complete"
   - "3. View all tasks"
   - "4. View incomplete tasks only"
   - "5. View tasks by priority"
   - "6. Delete task"
   - "7. Statistics"
   - "8. Exit"
3. For each menu option, use conditionals and loops to:
   - Add new tasks to the list
   - Mark tasks as complete by ID
   - Display tasks (all, incomplete only, or by priority)
   - Delete tasks by ID
   - Calculate and display statistics
4. Statistics should show:
   - Total tasks
   - Completed vs incomplete count (use loops to count)
   - Tasks by priority (use a dictionary and loops to count)
5. Use continue to skip invalid menu choices.
6. Use break to exit when user chooses "8".
7. Use conditionals to check if task ID exists before operations.
