ITEC 1150 Chapter 6 Lab Projects

STRING OPERATIONS

Program Development Plan (PDP)

This is a step-by-step process for successfully constructing an application. Follow these steps and repeat until you have successfully completed the program.

This is a reminder to use the PDP process.

You do not need to turn in your PDP document, but going through the process will help you design your programs.

PDP template -

1. Problem definition
2. Analysis of variables & functions
3. Steps in algorithm
4. Code (separate .py file with added comments)
5. Screen snips from testing
6. Keep track of any wanted or needed improvements for a future version

General Requirements

All assignments must meet the following requirements:

The program must start with header at top and include appropriate comments throughout. Header example:

```
"""
Author: Erik Granse
Date: 2024-09-02
Description: Calculate and display student's
average grades
"""
```

- Ensure the output is *information*; it needs to be a statement which explains the value being displayed (for example, "The average grade is 12.34"). Simply outputting "12.34" is not sufficient.

General Requirements (cont.)

All assignments must meet the following requirements:

- The data in the program must be stored in variables.
- The output **must** come from variables in the program
  - Do not simply hard code the output value in the `print()` statement.
  - Some data will be given to you, and some will be user input—any calculations that need to happen must be in your program. Don't calculate somewhere else and enter the value into your program.

General Requirements (cont.)

All assignments must meet the following requirements:

- All input must be validated to ensure the string from `input()` can be turned into a number without crashing.
- All input must be validated to ensure it meets the requirements of the lab (for example, ensuring an age is >= 0 or a quiz score is between 0 and 10).
- If input is not valid, you must give a message to the user and allow them to try again until the input is valid.
- Exemptions to the above will be called out in the lab sections. **If not exempted, validation is required!**

General Requirements, continued

- MIPO:
  - Main
  - Inputs
  - Processing
  - Outputs
- This is the basic structure all our programs will now follow.
- Add additional functions as necessary, but the MIPO functions must exist and be used.
- Generic exception handling must be used to ensure input errors do not cause a crash.
- Programs must offer restart to the user when they are done.

Lab Section 1: Title Validator

- Create a new program named **text_validator.py**.
- Your program must:
  - Define **main()**, **inputs()**, and **outputs()** functions (**the processing function is not needed**).
  - ask the user for a book title
  - validate that the title is not blank
  - use **.split()** on the string entered to turn it into a list of words
  - use a loop to correct capitalization and spacing for each word. In book titles:
    - *the, a, an,* and *of* are always lowercase unless they are the first or last word, in which case they are capitalized
    - All other words are capitalized
  - Add each corrected word back to a new string, and return, then display the formatted title in **outputs()**.

See hints and output samples on the next slide.

Lab Section 1: Title Validator (cont.)

Example output:

```
1. Welcome to the title validation program.
2. Please enter a title for validation & correction:
3. Please try again to enter a title: the history of the world
4. The corrected title is The History of the World
5. Do you want to validate another title? (y/n) n
6. Thanks for using the program.
```

A blank value entered on line 2, which re-prompts the user

First word is capitalized, even though it is "the"

"of" and "the" are not capitalized in the middle of the title

Lab Section 2: Feedback Collector

- Create a new program named **feedback_collector.py**. It collects phrases that will be used in an online learning system when students get answers right.
- Your program must:
  - Use all the MIPO program features (exception handling, restart, functions, etc.)
  - Inputs function:
    - Ask the user to enter multiple feedback phrases into a single input prompt. Each phrase must end in an exclamation point.
    - Validate that the input is not blank and has at least 1 exclamation point.
    - Fix common errors, like taking out extra spaces. Return the corrected input string.

Lab Section 2: Feedback Collector (cont.)

- Processing function:
  - Split the input phrases into a list, based on the ! character.
  - Using a loop, capitalize each phrase, remove any spaces from the beginning and end of the phrase, and add the exclamation point back to the end of the phrase.
  - Collect all phrases into a list and return.
- Outputs function:
  - Take in the corrected phrase list and use loop printing to display each phrase as shown in the next slide.

Lab Section 2: Feedback Collector Example

```
1. Welcome to the feedback generator.
2. Please enter multiple feedback phrases, each ending in an exclamation point.
3. Enter as many as you like. You don't have to capitalize: you're awesome! you rock!
   you got this!
4. Here are your feedback phrases:
5. 1: You're awesome!
6. 2: You rock!
7. 3: You got this!
8. Thanks for helping us build our feedback library.
9. Would you like to try again? Enter y or n:
```

Final Steps

Remember, both programs must include multiple interactive functions and validation. The feedback_collector program must also include exception handling and restart.

Submit your text_validator.py and feedback_collector.py programs to D2L by the deadline.

Questions or need help? Ask before the close date.

Read Chapter 7 of the book on Regular Expressions.