ITEC 1150 Chapter 7 Lab Projects

PATTERN MATCHING WITH REGULAR EXPRESSIONS

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: Full Name Validation

You do not need to meet MIPO requirements

- Create a program named **fullname_regex.py**. The program must:
  - Ask the user to enter a full name to search, in the format "first, middle, last"
  - Use a regular expression to check if the input could be a full name.
    - A full name contains three separate names, separated by spaces.
    - Each name consists of letters only (remember to think about upper and lower case!)
  - Print a message to the user telling them whether their input looks like it could be a full name.
  - Include the name (properly capitalized) in the message if it is valid.

Hint: Be sure to check the whole string from beginning to end (see slide 12 in the lecture notes).

Challenge: Include common punctuation in the name, such as apostrophes and dashes.

Lab Section 1: Examples

Main examples

```
Enter a name in the format 'first middle last': 
sally anne cavanaugh

Here is the name: Sally Anne Cavanaugh
```

```
Enter a name in the format 'first middle last': 
irwin fletcher

This does not look like a name.
```

```
Enter a name in the format 'first middle last': big 
red 1

This does not look like a name.
```

Challenge examples

```
Enter a name in the format 'first middle last': 
irwin m. fletcher

Here is the name: Irwin M. Fletcher
```

```
Enter a name in the format 'first middle last': 
d'angelo j barksdale

Here is the name: D'Angelo J Barksdale
```

```
Enter a name in the format 'first middle last': 
alonzo _ mosley

This does not look like a name.
```

Lab Section 2: Float Validation

You do not need to meet MIPO requirements

- We want user input like -12.34 to be recognized as a number! It's the long-awaited float validation!
- Create a program named float_validation.py. Your program must:
  - Prompt the user for a number with or without a decimal point or minus symbol.
  - Use a regular expression to check if the input is a valid number:
    - The minus sign is optional and must be at the beginning if present.
    - There must be one or more digits before the decimal point, if it is present.
    - The decimal point is optional and only one is allowed.
    - There must be one or more digits after the decimal point.
    - No characters other than digits (0-9), the decimal and a minus sign may be present.
  - Output a message to the user telling them whether their input is a valid number. Include the number in the message if it is valid.
  - See example output on the next slide.

Section 2: Examples

Valid Input

```
Enter a number. Negatives and decimals are allowed:
3.14159265

3.14159265 is a valid number!


Enter a number. Negatives and decimals are allowed:
-123.456

-123.456 is a valid number!


Enter a number. Negatives and decimals are allowed:
-0.99

-0.99 is a valid number!
```

Invalid Input

```
Enter a number. Negatives and decimals are allowed:
-.25

This does not look like a valid number.


Enter a number. Negatives and decimals are allowed:
1.0.1

This does not look like a valid number.


Enter a number. Negatives and decimals are allowed:
0xAE

This does not look like a valid number.
```

Submit & Do Reading for Chapter 8!

Remember to comment your code - for every program!

Save your programs for future reference.

Submit your 2 programs: fullname_regex.py and float_validation.py files to D2L before the deadline.

Questions or need help? Ask before the deadline!

Submit new/improved lab files & then read Chapter 8