Final Project

Due 11:59 p.m., Friday, 30 November

We agreed in class on 14 November for Assignment #4 to be used in place of the second examination. Thus I renamed this assignment the Final Project.

Correction (2018-11-16): The example exam was incorrect. I had mislabelled the Feedback on the choices as Hints and left off the Hints from questions.

General Instructions

All homework and programming exercises must be prepared in accordance with the instructions given in the Syllabus. Each assignment must be submitted to your instructor by its stated deadline.

Citations: In accordance with expected scholarly and academic standards, if you reference outside textbooks, reference books, articles, websites, etc., or discuss an assignment with individuals inside or outside the class, you must document these by including appropriate citations or comments at prominent places in your submission such as in the header of the primary source file.

Identification: Put your name, course name, and assignment number as comments in each file you submit.
Goal

The goal for this assignment is to practice applying the modular and object-oriented design and Python 3 implementation techniques we have studied this semester. In particular, we practice the language processing techniques.

Assignment Description

1. This is an individual assignment (unless otherwise approved by the instructor).

2. For the Exam DSL Project Project given below:
   • Develop a grammar and a set of Python 3 modules to solve the problem.
   • Use the design and programming practices we have discussed this semester.
     You may use a mix of modular and object-oriented programming techniques as appropriate. Follow good design and programming practices.

3. Design appropriate tests for your program and test the program thoroughly.

4. Document your program appropriately.

5. Submit the source code and documentation for your program and test driver, any needed instructions on how to run the program, and appropriate test output to Blackboard. Be sure to identify yourself in the materials turned in.

Exam DSL Parsing Project

Introduction

Few computer science graduates will design and implement a general-purpose programming language during their careers. However, many graduates will design and implement—and all likely will use—special-purpose languages in their work.

These special-purpose languages are often called *domain-specific languages* (or DSLs). For more discussion of DSL concepts and terminology, see the accompanying notes on Domain-Specific Languages.

In this project, we design and implement a simple *external DSL*. This DSL consists of a simple textual language defined with a grammar. Our task is to develop a program that parses the language input and generates an abstract
Syntax tree (AST) defined using Python 3 objects—preferably objects defined using Python 3.7 frozen data classes.

Note: I assigned a related project as CSci 450/503 Assignment #4 this semester. However, that project involved a bit different structure for the exam, processing ASTs implemented as Haskell algebraic data types, and generating HTML output. You may find the discussion of that assignment useful in this work.

Building Exam DSL

Suppose Professor Harold Pedantic decides to create a DSL to encode his (allegedly vicious) multiple choice examinations. Because some of his courses use Python, he wishes to implement the language processor in Python. Professor Pedantic is too busy to do the task himself. He is also cheap, so he assigns us, the students in his class, the task of developing a prototype.

First, let’s focus on multiple-choice questions. For this prototype, we can assume a question has the following components:

- the text of the question
- an optional hint for the question (perhaps when used with an interactive quiz program)
- a group of several choices for the answer to the question, exactly one of which should be the correct answer to the question
- each choice has an optional feedback response (perhaps when used with an interactive quiz program after getting an answer)
- a group of zero or more tags identifying topics covered by the question (perhaps used in selecting questions for the exam)

For example, we might have the following DSL input to describe a question.

```bash
Question ["curriculum", "course"] "Which of the following is a required course?"
Hint "It is a 400-level course."
Choice "CSci 323" Feedback "Systems Programming is elective."
Choice correct "CSci 450" Feedback "Programming Languages is core."
Choice "CSci 525" Feedback "Compiler Construction is elective."
```

Now, let’s consider the examination as a whole. It consists of an optional title and a list of questions. We thus define the data type Exam as follows.

We can encode an exam with three questions as follows.

```bash
Exam "Curriculum Test"

Question ["curriculum", "course"] "Which of the following is a required course?"
Hint "It is a 400-level course."
```
Choice "CSci 323" Feedback "Systems Programming is elective."
Choice correct "CSci 450" Feedback "Programming Languages is core."
Choice "CSci 525" Feedback "Compiler Construction is elective."

Question ["language","course"]
"What one of the following languages is used in CSci 556?"
Choice "Lua"
Choice "Elixir"
Choice "Scala"
Choice "Haskell"
Choice correct "Python 3"
Choice "Rust"

Question "Are ready for Thanksgiving Break?"
Hint "Is this a valid question?"
Choice correct "Yes"
Choice correct "No"
Choice "Maybe"

Exercises

1. Define a grammar for the Exam DSL as described above. Document the grammar using BNF or a similarly precise notation.

2. Develop a Python 3 module to represent the ASTs for exams as described above. (See the Calculator expression example calc_ast.py.)

3. Develop a Python 3 parser that accepts inputs that follow the grammar and creates the corresponding ASTs. (See the Calculator expression example calc_parser.py.)

   You may use Parsita (as I did in my examples), some other appropriate Python 3 package (e.g. Parya, Argpeggio/textX), or a hand-coded recursive descent parser.

4. Challenge: Develop a Python 3 module (or modules) to output an exam as HTML. (See CSci 450 Assignment #4.)

   You may use an appropriate Python 3 HTML template package or develop your own module similar to my Haskell module SimpleHTML.hs.