Assignment 2

COMP SCI 2ME3 and SFWR ENG 2AA4

February 9, 2019

1 Dates and Deadlines

Assigned: January 29, 2019

Part 1: February 11, 2019

Receive Partner Files: February 16, 2019

Part 2: February 18, 2019

Last Revised: February 9, 2019

All submissions are made through git, using your own repo located at:

`https://gitlab.cas.mcmaster.ca/se2aa4_cs2me3_assignments_2018/[macid].git`

where `[macid]` should be replaced with your actual macid. The time for all deadlines is 11:59 pm. If you notice problems in your Part 1 `*.py` files after the deadline, you should fix the problems and discuss them in your Part 2 report. However, the code files submitted for the Part 1 deadline will be the ones graded.

2 Introduction

The purpose of this software design exercise is to write a Python program that follows the given formal specification. The given specification covers a problem similar to A1 - the allocation of first year engineering students into their respective second year programs. As for the previous assignment, you will use doxygen, make, LaTeX and Python (version 3). In addition, this assignment will use pytest for unit testing and flake8 to verify that its pep8-inspired standard is enforced. This assignment also takes advantage of functional programming in Python.

All of your code, except for the testing files, should be documented using doxygen. Using doxygen on the testing files is optional. Your report should be written using LaTeX. Your code should follow the given specification exactly. In particular, you should not add public methods or procedures that are not specified and you should not change the number or order of parameters for methods or procedures. If you need private methods or procedures, please use the Python convention of naming the files with the double underscore (`__methodName__`) (dunders). **Please follow specified naming EXACTLY. You do not want to lose marks for a simple mistake.**

For the purpose of understandability, the specification provided at the end of the assignment uses notation that is slightly different from the Hoffman and Strooper notation. Specifically the types for real and natural numbers are represented by $\mathbb{R}$ and $\mathbb{N}$, respectively. (In this specification, the natural numbers are assumed to include 0.) Booleans are represented by $\mathbb{B}$. Also, subscripts are used for indexing a sequence. For instance, x_i means the same thing as $x[i]$. A subsection has also been added to the template for local types. The purpose of these local types is for specification; they are not exported.

2.1 Installing flake8

We will use flake8 to ensure your Python code meets the style conventions of the course. You will need to install two 'pip' packages. This is the standard way to install packages for Python.

First run the following command in your terminal:

```
pip --version
```

If the output includes 'Python 3,' then run the following instructions:

```
pip install flake8
pip install pep8-naming
```

If it does not, then use `pip3` for the following alternate instructions.

```
pip3 install flake8
pip3 install pep8-naming
```

2.2 Running flake8

Run the following command in your A2 directory. This will inform you of the location and types of style violations. You can find more information on what each error means here: https://lintlyci.github.io/Flake8Rules/

```
flake8
```

Part 1

Step 1

Write the modules `StdntAllocTypes.py`, `SeqADT.py`, `DCapALst.py`, `AALst.py`, `SALst.py` and `Read.py` following the specification given at the end of the assignment.

Step 2

Write a module (named `test_All.py`), using pytest, that tests the following modules: `SeqADT.py`, `DCapALst.py` and `SALst.py`. The given makefile `Makefile` has a rule `test` for running your tests. Each procedure/method should have at least one test case. Record your rationale for test case selection and the results of using this module to test the procedures in your modules. (You will submit your rationale with your report in Step 6.) Please make an effort to test normal cases, boundary cases, and exception cases. Your test program should compare the calculated output to the expected output and provide a summary of the number of test case that have passed or failed.

Step 3

Test the supplied `Makefile` rule for `doc`. This rule should compile your documentation into an html and LaTeX version. Along with the supplied `Makefile`, a doxygen configuration file is also given in your initial repo. You should not have to change these files.

Step 4

Submit (add, commit and push) all Python files using git. (Of course, you will be doing this throughout the development process. This step is to explicitly remind you that the version that will be graded is the one we see in the repo.) Please **do not change** the names and locations for the files already given in your git project repo. You should also push any input data files you created for testing purposes. For Part 1, the only files that you should modify are the Python files and the only "new" files you should create are the input data files. Changing other files could result in a serious grading penalty, since the TAs might not be able to run your code and documentation generation. You should NOT submit your generated documentation (html and latex folders). In general, files that can be regenerated are not put under version control.

Optionally, you can tag your final submission of Part 1 of the assignment with the label `A2Part1`.

Part 2

Your `SeqADT.py`, `DCapALst.py` and `SALst.py` files will automatically be pushed to your partner's repo and vice versa. **Including your name in your partner code files is optional.**

Step 5

After you have received your partner's files, replace your corresponding files with your partner's. Do not initially make any modifications to any of the code. Run your test module and record the results. Your evaluation for this step does not depend on the quality of your partner's code, but only on your discussion of the testing results. If the tests fail, for the purposes of understanding what happened, you are allowed to modify your partner's code.

Step 6

Write a report using LaTeX (`report.tex`) following the template given in your repo. Optionally, the final submission can have the tag `A2Part2`. The report should include the following:

1. Your name and macid.
2. Your updated Python files.
3. Your partner's files.

4. The results of testing your files (along with the rational for test case selection). The summary of the results should consist of the following: the number of passed and failed test cases, and brief details on any failed test cases.

5. The results of testing your files combined with your partner's files.

6. A discussion of the test results and what you learned doing the exercise. List any problems you found with (a) your program, (b) your partner's module, and

7. The specification of this assignment imposed design decisions on you. Please provide a critiques of the design. What did you like? What areas need improvement? How would you propose changing the design?

8. Answers to the following questions

   a) Contrast the natural language of A1 to the formal specifiation of A2. What are the advantages and disadvantages of each approach?

   b) The specification makes the assumption that the gpa will be between 0 and 12. How would you modify the specification to change this assumption to an exception? Would you need to modify the specification to replace a record type with a new ADT?

   c) If we ignore the functions `sort`, `average` and `allocate`, the two modules `SALst` and `DCapALst` are very similar. Ignoring the functions mentioned, how could the documentation be modified to take advantage of the similarities?

   d) A1 had a question about generality of an interface. In what ways is A2 more general than A1?

   e) The list of choices for each student is represented by a custom object, SeqADT, instead of a Python list. For this specific usage, what are the advantages of using SeqADT over a regular list?

   f) Many of the strings in A1 have been replaced by enums in A2. For these cases, what advantages do enums provide? Why weren't enums also introduced in the specification for macids?

The writing style for the report should be professional, but writing in the first person is fine. Some of your ideas can be summarized in lists, but most of the report should be written in full sentences. Spelling and grammar is important.

Commit and push `report.tex` and `report.pdf`. Although the pdf file is a generated file, for the purpose of helping the TAs, we'll make an exception to the general rule of avoiding version control for generated files. If you have made any changes to your Python files, you should also push those changes.

Notes

1. Your git repo will be organized with the following directories at the top level: `A1`, `A2`, `A3`, and `A4`.

2. Inside the `A2` folder you will start with initial stubs of the files and folders that you need to use. Please do not change the names or locations of any of these files or folders.

3. Please put your name and macid at the top of each of your source files, except for those that you share with a partner. Including your name and macid is optional for those files.

4. Your program must work on mills when compiled with its versions of Python (version 3), LaTeX, doxygen, make, pytest, pytest-cov, and flake8.

5. Python specifics:

   - The exceptions in the specification are intentionally selected to use the names of existing Python exceptions. There is no need to define new exceptions.
   - Although efficient use of computing resources is always a good goal, your implementation will be judged on correctness and not on performance.
   - For the Python implementation of an abstract module, use `@staticmethod`. Your access programs should be called via the module name followed by a dot and then the access program name. For instance, for a module named `Data`, the access program is accessed using `Data.accessProg`, not simply `accessProg` or `Data_accessProg`. The call `Data.Data_accessProg` is also incorrect.
   - For types that are defined as a set of potential values (like GenT), you should use an enumerated type (https://docs.python.org/3/library/enum.html).
   - For types that are simply records, use the `typing.NamedTuple` class from https://docs.python.org/3/library/typing.html#typing.NamedTuple.
   - Since the specification is silent on this point, for methods that return an object, or use objects in their state, you can decide to either use references or construct new objects. The implementation will be easier if you just work in terms of references to objects.
   - A sample program (`A2Examples.py`) that uses the modules in this specification is available in the repo. You can use this to do an initial test that your interface matches the specification.
   - Sample files (`StdntData.txt` and `DeptCap.txt`) are available in the repo.

- Marking scheme will be available in the course repo. The marking scheme will include some grades based on git usage, make, unit testing, and flake8.

6. **Your grade will be based to a significant extent on the ability of your code to compile and its correctness. If your code does not compile, then your grade will be significantly reduced.**

7. **Any changes to the assignment specification will be announced in class. It is your responsibility to be aware of these changes. Please monitor all pushes to the course git repo.**

Student Allocation Types Module

Module

StdntAllocTypes

Uses

SeqADT(T)

Syntax

Exported Constants

None

Exported Types

GenT = {male, female}
DeptT = {civil, chemical, electrical, mechanical, software, materials, engphys}

SInfoT = tuple of (fname: string, lname: string, gender: GenT, gpa: $\mathbb{R}$, choices: SeqADT(DeptT), freechoice: $\mathbb{B}$)

Exported Access Programs

None

Semantics

State Variables

None

State Invariant

None

Assumptions

For SInfoT the gpa will always lie between 0 and 12.0.

Sequence ADT Module

Generic Template Module

SeqADT(T)

Uses

None

Syntax

Exported Constants

None

Exported Types

SeqADT = ?

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
new SeqADT	sequence of T	SeqADT	
start			
next		T	StopIteration
end		$\mathbb{B}$	

Semantics

State Variables

s: sequence of T
i: integer

State Invariant

$i \in [0..|s|]$

Assumptions

The constructor for SeqADT will not be called with an empty sequence as input.

Access Routine Semantics

new SeqADT(x):

- transition: $s, i := x, 0$
- output: $out :=$ self
- exception: none

start():

- transition: $i := 0$
- exception: none

next():

- transition-output: $i, out := i + 1, s[i]$
- exception: $(i \geq (|s|) \Rightarrow$ StopIteration)

end():

- output: $out := i \geq |s|$
- exception: None

Department Capacity Association List Module

Module

DCapALst

Uses

StdntAllocTypes

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine name	In	Out	Exceptions
init			
add	DeptT, $\mathbb{N}$		KeyError
remove	DeptT		KeyError
elm	DeptT	$\mathbb{B}$	
capacity	DeptT	$\mathbb{N}$	KeyError

Semantics

State Variables

s: set of tuple of (dept: DeptT, cap: $\mathbb{N}$)

State Invariant

None

Assumptions

DCapALst.init() is called before any other access program.

Access Routine Semantics

init():

- transition: $s := \{\}$
- exception: none

add(d, n):

- transition: $s := s \cup \{\langle d, n \rangle\}$
- exception: $(\langle d, ? \rangle \in s \Rightarrow \text{KeyError})$ (? can be any natural number)

remove(d):

- transition: $s := s - \{\langle d, n \rangle\}$ where $\langle d, n \rangle \in s$
- exception: $(\langle d, n \rangle \notin s \Rightarrow \text{KeyError})$

elm(d):

- output: $\mathit{out} := \langle d, n \rangle \in s$
- exception: none

capacity(d):

- output: $\mathit{out} := n$ where $\langle d, n \rangle \in s$
- exception: $(\langle d, n \rangle \notin s \Rightarrow \text{KeyError})$

Allocation Association List Module

Module

AALst

Uses

StdntAllocTypes

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine name	In	Out	Exceptions
init			
add_stdnt	DeptT, string		
lst_alloc	DeptT	sequence of string	
num_alloc	DeptT	$\mathbb{N}$	

Semantics

State Variables

s: set of AllocAssocListT

State Invariant

None

Assumptions

AALst.init() is called before any other access program.

Access Routine Semantics

init():

- transition: $s := \{d : \text{DeptT} | d \in \text{DeptT} : \langle d, []\rangle\}$
- exception: none

add_stdnt(dep, m):

- transition: $s := \{\langle d, L\rangle : \text{AllocAssocListT} | \langle d, L\rangle \in s : (d = dep \Rightarrow \langle d, L || [m]\rangle | \text{True} \Rightarrow \langle d, L\rangle)\}$
- exception: none

lst_alloc(d):

- output: $out := L$ where $\langle d, L\rangle \in s$
- exception: none

num_alloc(d):

- output: $out := |L|$ where $\langle d, L\rangle \in s$
- exception: none

Local Types

AllocAssocListT = tuple of (dept: DeptT, sequence of string)

Student Association List Module

Module

SALst

Uses

StdntAllocTypes
AALst
DCapALst

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine name	In	Out	Exceptions
init			
add	string, SInfoT		KeyError
remove	string		KeyError
elm	string	$\mathbb{B}$	
info	string	SInfoT	KeyError
sort	SInfoT $\rightarrow \mathbb{B}$	sequence of string	
average	SInfoT $\rightarrow \mathbb{B}$	$\mathbb{R}$	ValueError
allocate			RuntimeError

Semantics

State Variables

s: set of StudentT

State Invariant

None

Assumptions

SALst.init() is called before any other access program. DCapALst has been fully populated for all departments before running allocate. The following assumptions apply to the data:

- The free choice students will never number so many that they will fill a department.
- The case will never arise where the next student to be added to a department will exceed the capacity of the department, but have the exact same gpa as the last student allocated to that department,

Access Routine Semantics

init():

- transition: $s := \{\}$
- exception: none

add(m, i):

- transition: $s := s \cup \{\langle m, i \rangle\}$
- exception: $(\langle m, ? \rangle \in s \Rightarrow \text{KeyError})$

remove(m):

- transition: $s := s - \{\langle m, i \rangle\}$ where $\langle m, i \rangle \in s$
- exception: $(\langle m, i \rangle \notin s \Rightarrow \text{KeyError})$

elm(m):

- output: $\mathit{out} := \langle m, i \rangle \in s$
- exception: none

info(m):

- output: $\mathit{out} := i$ where $\langle m, i \rangle \in s$

- exception: $(\langle m, i\rangle \notin s \Rightarrow \text{KeyError})$

sort(f):

- output: $out := L$: sequence of string, such that $(\forall \langle m, i\rangle : \text{StudentT} | \langle m, i\rangle \in s \wedge f(i) : (\exists j : \mathbb{N} | j \in [0..|s|-1] : L_j = m)) \wedge (\forall k : \mathbb{N} | k \in [0..|L|-2] : \text{get\_gpa}(L_k, s) \geq \text{get\_gpa}(L_{k+1}, s))$

- exception: none

average(f):

- output:

$$out := \frac{(+i : \text{SInfoT} | i \in \mathit{fset} : i.\text{gpa})}{|\mathit{fset}|} \text{ where } \mathit{fset} = \{\langle m, i\rangle : \text{StudentT} | \langle m, i\rangle \in s \wedge f(i) : i\}$$

- exception: $(\{\langle m, i\rangle : \text{StudentT} | \langle m, i\rangle \in s \wedge f(i) : i\} = \emptyset \Rightarrow \text{ValueError})$

allocate():

- transition: # *procedural specification*

```
AALst.init()
$F$ = SALst.sort($\lambda t \to t$.freechoice $\wedge$ $t$.gpa $\geq 4.0$)
for all $m$ in $F$
    $ch$ = SALst.info($m$).choices
    AALst.add_stdnt($ch$.next(), $m$)

$S$ = SALst.sort($\lambda t \to \neg t$.freechoice $\wedge$ $t$.gpa $\geq 4.0$)
for all $m$ in $S$
    $ch$ = SALst.info($m$).choices
    $alloc$ = False
    while $\neg alloc \wedge \neg ch$.end()
        $d = ch$.next()
        if AALst.num_alloc($d$) < DCapALst.capacity($d$)
            AALst.add_stdnt($d, m$)
            $alloc$ = True
    if $\neg alloc$ raise(RuntimeError)
```

- exception: none

Local Types

StudentT = tuple of (macid: string, info: SInfoT)

Local Functions

get_gpa: string × set of StudentT

get_gpa$(m,\, s) \equiv i.$gpa for $\langle m, i\rangle \in s$

Read Module

Module

Read

Uses

StdntAllocTypes, DCapALst, SALst

Syntax

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
load_stdnt_data	s : string		
load_dcap_data	s : string		

Semantics

Environment Variables

stdnt_data: File listing student data
dept_capacity: File listing department capacities

State Variables

None

State Invariant

None

Assumptions

The input file will match the given specification.

Access Routine Semantics

load_stdnt_data(s)

- transition: read data from the file stdnt_data associated with the string s. Use this data to update the state of the SALst module. Load will first initialize SALst (SALst.init()) before populating SALst with student data that follows the types in StdntAllocTypes.

  The text file has the following format, where id_i, fn_i, ln_i, g_i, gpa_i, $[ch_i^0, ch_i^1, ..., ch_i^{n-1}]$ and fc_i stand for strings that represent the ith student's macid, first name, last name, gender, grade point average, list of choices and free choice, respectively. The gender is represented by either the string "male" or "female." The list of choices comes from strings following the department names in the type DeptT. The list of choices has length n. fc_i is either the string "True" or the string "False." All data values in a row are separated by commas. Rows are separated by a new line. The data shown below is for a total of m students.

$$\begin{array}{ccccccc}
id_0, & fn_0, & ln_0, & g_0, & gpa_0, & [ch_0^0, ch_0^1, ..., ch_0^{n-1}], & fc_0 \\
id_1, & fn_1, & ln_1, & g_1, & gpa_1, & [ch_1^0, ch_1^1, ..., ch_1^{n-1}], & fc_1 \\
id_2, & fn_2, & ln_2, & g_2, & gpa_2, & [ch_2^0, ch_2^1, ..., ch_2^{n-1}], & fc_2 \\
..., & ..., & ..., & ..., & ..., & [..., ...,], & ... \\
id_{m-1}, & fn_{m-1}, & ln_{m-1}, & g_{m-1}, & gpa_{m-1}, & [ch_{m-1}^0, ch_{m-1}^1, ..., ch_{m-1}^{n-1}], & fc_{m-1}
\end{array} \quad (1)$$

- exception: none

load_dcap_data (s)

- transition: read data from the file dept_capacity associated with the string s. Use this data to update the state of the DCapALst module. Load will first initialize DCapALst (DCapALst.init()) before populating DCapALst with department capacity data.

  The text file has the following format. Each department is identified by a string with the department name, and then a string for the natural number that represents the department's capacity. All data values in a row are separated by commas. Rows are separated by a new line.

civil,	n_{civil}
chemical,	n_{chemical}
electrical,	$n_{\text{electrical}}$
mechanical,	$n_{\text{mechanical}}$
software,	n_{software}
materials,	$n_{\text{materials}}$
engphys,	n_{engphys}

- exception: none