CSC110 Lecture 11: Data Classes

Print this handout

Exercise 1: Reviewing data classes

For all exercises on this worksheet, please assume that the `dataclass` decorator has been successfully imported:

```
from dataclasses import dataclass
```

(If you are working on your own computer, you should add this line to the top of the file.)

1. Each code snippet below attempts to define and/or use a data class to represent a food container. Unfortunately, each has some kind of problem (syntax, logical, style, etc.) Underneath each one, identify the problem.

   Assume all the necessary imports are included—this is not the error.

```
dataclass FoodContainer:
    """A container that can store different foods."""
    label: The name of this container
    contents: The contents of this container
```

   Problem:

   - @dataclass
     class FoodContainer:

   improper Python data types

```
class FoodContainer:
    """A container that can store different foods."""
    label
    contents
```

@dataclass

Problem:

· attribute types not given

```
@dataclass
class FoodContainer:
    """A container that can store different foods."""
    label: str
    contents: list[str]


# In Python console
>>> mc = FoodContainer('Nothing in here...')
```

Problem:

· expecting 2 values but given 1

2. Suppose we have the following data class definition:

```
@dataclass
class FoodContainer:
    """A container that can store different foods."""
    label: str
    contents: list[str]
```

Write an expression to represent a food container labelled `'Mario lunch box'` containing items `'sushi'` and `'chocolate'`.

label=

```
>>> mlb = FoodContainer('Mario lunch box',
                        contents=['sushi', 'chocolate'])
```

3. Implement the function below:

```

```

```
def num_contents(food_containers: list[FoodContainer]) -> int:
    """Return the total number of items contained in the given
        food_containers.

    >>> container1 = FoodContainer('David', ['ham', 'cheese', 'chocolate'])
    >>> container2 = FoodContainer('Tom', ['sushi', 'chips'])
    >>> num_contents([container1, container2])
    5
    """
```

3 + 2

return sum([len(container.contents)

for container in food_containers])

avoid using indexing when not required

sum([len (food_containers[i].contents for i in range (0, len(f_c))])

Exercise 2: Representation invariants

1. We have defined following data class to represent a student at the University of Toronto. Review the attributes of this data class, and then brainstorm some representation invariants for this data class. Write each one as a Python expression (using `self.<attribute>` to refer to instance attributes), and then for practice write English translations for each one.

```
@dataclass
class Student:
    """A student at the University of Toronto.

    Representation Invariants:
        - self.year_of_study >= 0
        - self.year_of_study < 5
        - len(self.utorid) > 0
        - str.isalnum(self.utorid)


    """
    given_name: str
    family_name: str
    year_of_study: int
    utorid: str
```

- names len > 0? != ''

2. The following data class represents data for the Computer Science Student Union.

```
@dataclass
class Cssu:
    """The Computer Science Student Union at the University of Toronto.

    Instance Attributes:
        - president: The Student who is the president
        - vice_president: The Student who is vice-president
        - events: The names of events that the CSSU holds throughout the
        year

    Representation Invariants:

        - self.president.year_of_study >= 3
        - self.president.given_name !=
              self.vice_president.given_name
        - self.president != self.vice_president

    """
    president: Student
    vice_president: Student
    events: list[str]
```

Think about relationship between these two.

Complete the data class docstring by translating the following representation invariants into Python expressions:

- the president must be in at least 3rd year
- the president and vice-president cannot have the same given name (*not realistic, but just for practice*)
- the president and vice-president are not the same student
    - *Hint*: you can use ==/!= to compare `Students`

- every event is a non-empty string containing only alphabetic characters
  - *Hint*: use `str.isalpha` to check a single string for this property

Exercise 3: Marriage licenses revisited

In our last lecture we used a nested list to represent a table of marriage license data:

ID	Civic Centre	Marriage Licenses Issued	Time Period
1657	ET	80	January 2011
1658	NY	136	January 2011
1659	SC	159	January 2011
1660	TO	367	January 2011
1661	ET	109	February 2011
1662	NY	150	February 2011
1663	SC	154	February 2011
1664	TO	383	February 2011

In this lecture, we saw how to define this as a data class:

```
@dataclass
class MarriageData:
    """A record of the number of marriage licenses issued in a civic centre
    in a given month.

    Instance Attributes:
      - id: a unique identifier for the record
      - civic_centre: the name of the civic centre
      - num_licenses: the number of licenses issued
      - month: the month these licenses were issued
    """
    id: int
    civic_centre: str
    num_licenses: int
    month: datetime.date  # Make sure to "import datetime"
```

In this exercise, you'll apply what you've learned about data classes to redo some of the computations fro
Lecture 10's worksheet (../10-tabular-data/worksheet/) using data classes instead of lists.

1. (warm-up) Using the above data class definition, write an expression to represent the row with id `1662` in the above table.

MarriageData(1662, 'NY', 150, datetime.date(2011, 2, 1))

2. Write representation invariants for this data class to represent each of the following contraints:
   - Civic centres must be one of `'TO'`, `'ET'`, `'NY'`, or `'SC'`.

see posted .py file for solutions to remainder of this worksheet

   - The number of marriage licenses is greater than or equal to 0.

3. Implement each of the following functions, which are equivalent to the ones from the previous worksheet, except they now take in a `list[MarriageData]` rather than a nested list.

```
def civic_centres(data: list[MarriageData]) -> set[str]:
    """Return a set of all the civic centres found in data.
    """




def civic_centre_meets_threshold(data: list[MarriageData], civic_centre:
        str,
                                 num: int) -> bool:
    """Return whether civic_centre issued at least num marriage licences
        every
    month.

    You only need to worry about the rows that appear in data; don't worry
        about
    "missing" months.

    Preconditions:
        - civic_centre in {'TO', 'NY', 'ET', 'SC'}

    HINT: you'll need to use a filtering comprehension.
    """









def summarize_licences_by_centre(data: list[MarriageData]) -> dict[str,
        int]:
```

```
    """Return the total number of licences issued by each civic centre in
        <data>.

    Returns a dictionary where keys are civic centre names and values are
        the
    total number of licences issued by that civic centre.

    HINT: you will find it useful to write a function that calculates the
        total
    number of licences issued for a given civic centre as a parameter,
    e.g. total_licenses_for_centre(data, civic_centre).
    """
```

Additional exercises

1. Consider the following alternate version of the `Cssu` data class from this worksheet:

```
@dataclass
class Cssu:
    """The Computer Science Student Union at the University of Toronto.

    Instance Attributes:
        - execs: A mapping from executive role (president, treasurer, etc.)
                 to Student.
        - merch: A mapping from clothing item (t-shirt, hoodie, etc.)
                 to price.

    Representation Invariants:









    """
    execs: dict[str, Student]
    merch: dict[str, float]
```

Complete the data class docstring by translating the following representation invariants into Python expressions:

- `'president'` is an executive role
- every executive role is a non-empty string containing only alphabetic letters

- every clothing item's *price* is ≥ 0

Some of these representation invariants will require using a comprehension to range over a `dict`. The key thing to know about doing so is that *the comprehension variable ranges over the **keys** of the dictionary*. Example:

```
>>> my_dict = {'a': 1, 'b': 2, 'c': 3}
>>> {key for key in my_dict}
{'a', 'b', 'c'}
>>> {key + '!' for key in my_dict}
{'a!', 'b!', 'c!'}
```

If you want to access the *values*, you can use key lookup:

```
>>> {my_dict[key] for key in my_dict}
{1, 2, 3}
```