CSC110 Lecture 10: Data Classes

Hisbaan Noorani

October 05, 2020

Contents

1 Ex 1: Reviewing data classes

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.

   (a) Number one

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

   Problem:

   - The header should be `class` not `dataclass`.
   - No `@dataclass` header.
   - English description of variables instead of Python.

   (b) Number two

```
@dataclass
   class food_container:
       """A container that can store different foods."""
       label: str
       contents: List[str
```

   Problem:

   - `@dataclass` should use `CamelCase` and not `snake_case`

   (c) Number three

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

Problem:

- No `@dataclass` header.
- No description of variables.

(d) Number four

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


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

Problem:

- You should pass an empty list instead of nothing when initializing a dataclass.
- The thing itself looks good.

2. Suppose we have the following data class definition:

```
from dataclasses import dataclass
from typing import List

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

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

```
lunch-box = FoodContainer('Mario\'s lunch box', ['sushi', 'chocolate'])
```

3. Assume we have the same data class definition as in the previous question, and we instantiate the following object:

```
>>> mc = FoodContainer('Secret snacks', ['mystery A', 'mystery B', 'mystery C'])
```

(a) Write an expression that accesses the second item in the container's contents:

```
>>> mc.contents[1]
'mystery B'
```

(b) Write an expression that would cause the following error: `AttributeError: 'FoodContainer' object has no attribute 'Contents'`.

```
>>> mc.Contents
```

(c) What does part (b) imply about writing names for accessing attributes?
They are case sensitive, and so are all things in Python.

2 Ex 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.given_name != ''
     - self.family_name != ''
     - self.year_of_study >= 1
     - len(self.utorid) == 8
    """
    given_name: str
    family_name: str
    year_of_study: int
    utorid: str
```

2. The following data class represents data for the Computer Science Student Union. Again, review the attributes (we've also provided descriptions in the class docstring), and then translate the following representation invariants into Python expressions:

   - every clothing item's price is ≥ 0
   - every executive role is a non-empty string containing only alphabetic letters (use `str.isalpha` to check for this)
   - no student can hold more than one executive role (you can use $==$ to compare Students)

```
@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:
     - all({self.merch[item] >= 0 for item in self.merch})
     - all({str.isalpha(role) for role in self.execs})

     - len({self.execs}) == len({self.execs[role] for role in self.execs})

    OR

     - len({(role1, role2) for role1 in self.execs for role 2 in self.execs if role1 != role2 and se

    OR
```

```
    - not any({self.execs[role1] == self.exec[role2] for role1 in self.execs for role2 in self.exec
    """
    execs: Dict[str, Student]
    merch: Dict[str, float]
```

3 Ex 3: Marriage Licences 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 dataclass:

```
@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

    Representation Invariants:
        - self.civic_centre in {'TO', 'ET', 'NY', 'SC'}
        - self.num_licenses >= 0
        - self.month.day == 1

    >>> row_1657 = MarriageData(1657, 'ET', 80, datetime.date(2011, 1, 1))
    """
    id: int
    civic_centre: str
    num_licenses: int
    month: datetime.date
```

In this exercise, you'll apply what you've learned about data classes to redo some of the computations from Lecture 9's 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.

```
>>> {mc for mc in marriage_data if mc.id == 1662}
```

2. Write representation invariants for this data class to represent each of the following:

   - Civic centres must be one of 'TO', 'ET', 'NY', or 'SC'.
   - The number of marriage licenses is greater than or equal to 0.
   - The month value has a day value of 1.
     Done in the above dataclass.

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.
    """
    return {row.marriage_center for row 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 list comprehension with a filter.
    """
    filtered_data = [row for row in data if row.civic_center = civic_center]
    return all({row.civic_center >= num for row in filtered data})


def issued_licenses_in_range(data: List[MarriageData], start: datetime.date, end: datetime.date) ->
    """Return the number of marriage licenses issued between start and end,
    inclusive.

    Preconditions:
        - end > start

    HINT: You can use <, <=, >, and >= to compare date values chronologically.
    """
    return sum([row.num_licenses for row in data if start <= row.month <= end])
```