Lab 4: Exceptions and MVC

Due: February 6, 9 p.m.

1 Introduction

This week we introduce Unified Modelling Language (UML) to describe the properties and methods of classes of objects as well as their relationships. This language an be used to illustrated common software designs, like designs for Model-View-Controller (MVC) systems. This week's lab is loosely structured according to an MVC design; its purpose is to allow you to play with visualization of data.

The data we'll be looking at an inventory of the city's water main breaks. Water main breaks are likely when pipes are old and the climate fluctuates rapidly, and they can be both costly and inconvenient. Engineers are actively exploring techniques to predict water main failures in order to better respond to potential problems (see this link for an example). Our data is drawn from the Toronto City Open Data Catalogue, and it details every water main break between 1990 and 2016:

https://open.toronto.ca/dataset/watermain-breaks/

Your goal will be to parse valid water main break data from a file and to populate *WatermainBreak* objects using the data. Given valid data, you will visualize locations of the water main breaks and allow users to filter for breaks that occurred during specific years. However, if you encounter invalid data, you will be raising *BreakException* objects.

As ever, you will find Lab 4 starter files on eClass and you can submit your completed code to this PrarieLearn URL:

https://us.prairielearn.com/pl/course_instance/200825/assessment/2625073

The files in this week's starter package are as follows:

user_code.py, *york.png*, and *york_breaks.csv*

Each method you must implement is described below and some unit tests (e.g. doctests) have been provided for you.

2 Programming Task

You will implement:

- A $WatermainBreak$ constructor, i.e.: $def\_\_init\_\_(self, when : date, year : int, lon : float, lat : float) \rightarrow None$. This code will be similar to what you have written in prior labs, save that we want you to raise a $BreakException$ with the message *"No data before 1990 or after 2016"* if the year that is provided to the constructor is less than 1990 or greater than 2016. You should also raise a $BreakException$ (with messages you can design) if the longitude or latitude are not floats, if the year is not an integer, or the date is not a *date* object.

- The $BreakFilter$ constructor, i.e.: $def\_\_init\_\_(self, filter\_years : list[int], breaks : list[WatermainBreak]) \rightarrow None$. This creates an object designed to identify subsets of $WatermainBreak$ objects from an incoming list of $WatermainBreak$ objects (*breaks*). The filter will select only those $WatermainBreak$ objects in *breaks* with a *year* attribute that is contained in the input list, $filter\_years$.

- The method $read\_breaks\_file(filename : str) \rightarrow list[WatermainBreak]$. This wwill read a comma separated file ($filename$) and return a list containing $WatermainBreak$ objects to represent each water main break in the file. Each row in $filename$ (after the first) represents data about a single water main break and is in this format:

  date, year, longitude, latitude

  The date will be a string in the format *YYYY/MM/DD*.

Once you are able to pass all of the doctests, you will be able to visualize subsets of water main breaks on a map, if you like! To do this, change line 19 in your starter code to read as follows, and then run your code:

$$draw = True$$

3 Testing and Submitting

It's always a good idea to write some doctests! What happens when years are negative or in the future? Make sure you can pass all of the tests we have provided as well as those you create before you submit.

To submit, you will cut and past the code you write in *user_code.py* into the Lab 4 instance of PrarieLearn. The score you get on PrarieLearn is the score you get for the lab!