CSC110 Tutorial 3: Function Correctness, Property-Based Testing, and Tabular Data

Print this handout

In this tutorial, you'll get more practice working with definitions, translating between English, predicate logic, and Python, writing proofs, and performing computations on tabular data. In the last part of this tutorial you'll apply what you've learned to analyze a real world dataset provided by the City of Toronto.

Exercise 1: Quick Review

1. Please answer the following questions.

   a. What is the most specific type annotation for `('hi', 'bye')`?

   b. What is the most specific type annotation for `[1, ['hi'], 2, ['hi', 'bye']]`?

   c. Suppose we have a function with the following header:

   ```
   def mystery(numbers: set[int], n: int) -> bool:
   ```

   Write a Python expression to represent the precondition "at least one number in `numbers` is greater than `n`".

   d. Suppose we define the following variables in the Python console:

   ```
   >>> data = [
   ...     ['a', 'b'],
   ...     [3, 4],
   ...     ['cat', 'mouse', 'elephant']
   ... ]
   >>> sublist = data[2]
   ```

   Write down what each of the following expressions evaluate to. *Do this by hand first, then check your work in the Python console.*

   ```
   >>> sublist




   >>> sublist[1]




   >>> data[0][0]




   >>> data[0][0] + data[1][1]




   >>> [row[0] for row in data]




   >>> len(data)




   >>> len(data[0])




   >>> len(data[1][0])
   ```

Exercise 2: Greatest common divisor and more proof practice

For your reference, here are the two definitions of *divisibility* and *prime* from lecture. (We've only included the symbolic forms here, to give you practice reading them!)

$$d \mid n : \!\!\Leftrightarrow \exists! k \in \mathbb{Z},\ n = dk^2 \quad \text{where } n, d \in \mathbb{Z}$$

$$IsPrime(p) : p > 1 \wedge (\forall d \in \mathbb{N},\ d \mid p \Rightarrow d = 1 \vee d = p), \qquad \text{where } p \in \mathbb{Z}$$

1. On a separate piece of paper, prove the following statement:

   $$\forall a, b, c \in \mathbb{Z},\ a \mid b \wedge b \mid c \Rightarrow a \mid c$$

Now, consider the following two definitions.

Definition 1.

Let $d, m, n \in \mathbb{Z}$. We say that d is a **common divisor** of m and n when it divides both m and n.

Definition 2.

Let $d, m, n \in \mathbb{Z}$. We say that d is the **greatest common divisor** of m and n when:

- d is the maximum common divisor of m and n, if at least one of m and n is non-zero.
- $d = 0$, if m and n are both 0.

We also can define the function $\gcd : \mathbb{Z} \times \mathbb{Z} \to \mathbb{N}$ to take in two integers and return their greatest common divisor. For example, we have $\gcd(10, 4) = 2$ and $\gcd(0, 0) = 0$.

2. On a separate piece of paper, prove the following statement:

   $$\forall m, p \in \mathbb{Z},\ IsPrime(p) \wedge p \mid m \Rightarrow \gcd(m, p) = 1$$

   You can use the statement we proved in lecture that $\forall m \in \mathbb{Z},\ 1 \mid m$.

Exercise 3: Property-based tests with gcd

In Python's `math` module, there is a function `gcd` that computes the greatest common divisor of two numbers. In this exercise, you'll practice writing some property-based tests for that function.

1. First, download the starter file `tutorial3_part3.py`. This file imports the `math` module, and sets up the structure for some property-based tests for this function.
2. For each of the following properties, implement the corresponding property-based test, and write an English description of the property in the docstring after the `"Test that"`.

   a. $\forall m, n \in \mathbb{Z},\ (m = 0 \wedge n = 0) \vee \gcd(m, n) \geq 1$
   b. $\forall n \in \mathbb{Z},\ \gcd(|n|, 0) = |n|$ (where $|n|$ denotes the absolute value of n)
   c. $\forall m, n \in \mathbb{Z},\ \gcd(2m, 2n) = 2 \cdot \gcd(m, n)$
   d. $\forall d, n \in \mathbb{Z},\ d \mid n \Rightarrow \gcd(d, n) = |d|$

   Tip: use `integers(min_value=-1000, max_value=1000)` rather than `integers()` to constrain the range of integers `hypothesis` generates. This will help speed up the tests.

   This is a good time to review Section 4.4 Testing Functions II: `hypothesis` to remind yourself how to write a property-based test.

Note: you might find it a bit weird that we're asking you to write tests for a function that comes with the built-in `math` module! Of course, you're doing this to practice writing property-based tests, but there's another reason: if you were to write *your own* implementation of a `gcd` function, it will be useful to have some tests already written to check your work!

Exercise 4: Data analysis on a csv data set

A significant source of frustration to the residents of Toronto are delays in public transit. Admittedly, adding time to your commute can take a negative toll on just about anyone who commutes. Some articles and books claim that a short commute time will improve your happiness. One article goes so far as linking the misery of additional commute time to a corresponding pay cut. In this exploration, you will analyze data on subway delays provided by the Toronto Transit Commission (TTC), the organization that runs Toronto public transit. You'll answer questions such as:

- When is the worst time/day/month to ride the subway?
- Are there any trends in the number of subway delays over time?
- What is the biggest contributor to delays on the subway (e.g., weather-related, signal problems, door problems)?
- Which stations are the most impacted by delays?

0. The data set

Download this data set into this week's tutorial folder. This file contains a record of all TTC delays in the time period from January 1, 2014 to October 31, 2019, courtesy of the City of Toronto. That's a lot of delays! The data is stored using the **comma-separated values (csv) file format**, which is a common way of storing tabular data in plain text. The csv format uses commas to separate entries in a row, and lines to separate rows in the table. For example, in our sample data the first four lines look like this:

```
Date,Time,Day,Station,Code,Min Delay,Min Gap,Bound,Line,Vehicle
01/01/2014,00:21,Wednesday,VICTORIA PARK STATION,MUPR1,55,60,W,BD,5111
01/01/2014,02:06,Wednesday,HIGH PARK STATION,SUDP,3,7,W,BD,5901
01/01/2014,02:40,Wednesday,SHEPPARD STATION,MUNCA,0,0,,YU,0
```

and they represent the following tabular data:

Date	Time	Day	Station	Code	Min Delay	Min Gap	Bound	Line	Vehicle
01/01/2014	0:21	Wednesday	VICTORIA PARK STATION	MUPR1	55	60	W	BD	5111
01/01/2014	2:06	Wednesday	HIGH PARK STATION	SUDP	3	7	W	BD	5901
01/01/2014	2:40	Wednesday	SHEPPARD STATION	MUNCA	0	0		YU	0

Here is a description and expected Python data types of the columns in this data set.

Column name	Description	Python data type
Date	The date of the delay	`datetime.date`
Time	The time of the delay	`datetime.time`
Day	The day of the week on which the delay occurred.	`str`
Station	The name of the subway station where the delay occurred.	`str`
Code	The TTC delay code, which usually describes the cause of the delay. You can find a table showing the codes and descriptions in `ttc-subway-delay-codes.csv`.	`str`
Min Delay	The length of the subway delay (in minutes).	`int`
Min Gap	The length of time between subway trains (in minutes).	`int`
Bound	The direction in which the train was travelling. This is dependent on the line the train was on.	`str`
Line	The abbreviated name of the subway line where the delay occurred.	`str`
Vehicle	The id number of the train on which the delay occurred.	`int`

1. Reading the file

Our first task is to take the `ttc-subway-delays.csv` and load the data in Python. In lecture, the data set we worked with was already written as Python code, so this is your first time working with "raw" data files in this course!

To get started, download `tutorial3_part4.py` into this week's tutorial folder. Then open this file, and follow these instructions:

1. The code we've provided does most of the work for reading a csv file using the `csv` Python module. Review the `read_file` function, and read the comments to learn about what it's doing. (We don't expect you to be able to write this code yourself yet, but you should be able to read it and understand what's going on with some guidance from your TA.)
2. Run this file in the Python console, and call the `read_csv_file` with the filename `'ttc-subway-delays.csv'`, and store the result:

   ```
   >>> result = read_csv_file('ttc-subway-delays.csv')
   >>> headers = result[0]
   >>> data = result[1:]
   ```

   There's a lot of data stored in this file, and just displaying the value of `data` is pretty overwhelming! Instead, use the Python console and these variables to answer the following questions:

   a. What headers are contained in the data?
   b. How long (i.e., number of entries) is each row of data?
   c. How many rows of data are there in total?
   d. What is the first row of data?
   e. What is the last row of data?

3. You'll notice that `read_csv_file` turns every row into a list of strings. However, in order to do useful computations on this data, we'll need to convert many of these entries into other Python data types, like `int` and `datetime.date`.

   Implement the function `process_row`, which processes a single row of data to convert the entries into their appropriate data types (specified in a table in the previous section).

4. Once you've tested your `process_row` function, modify `read_csv_file` so that you call `process_row` on each row inside the list comprehension where `data` is defined.

   This means that you should change the return type to `tuple[list[str], list[list]]`, since in the data the inner lists will have elements of different types (not just strings).

   Run your file in the Python console now, and check the first few rows of data. Make sure you see elements of the correct types before moving on!

2. Operating on the data

Now that we have this csv data stored as a nested list in Python, we can can do some analysis on it! Complete the functions below to answer some questions about this data. Don't worry if you don't get through all of these functions today. The important part is that you're getting practice on working with tabular data, and will be able to take these functions (and others that you come up with) as additional practice later!

1. What was the longest subway delay? (`longest_delay`)
2. On average, how long do the subway delays last? (`average_delay`)
3. How many subway delays were there in July 2018? (`num_delays_by_month`)
4. How many delays were caused by each cause? (`delays_by_cause`) (This will give you codes, but you'll need to look them up yourself in `ttc-subway-delay-codes.csv`.)
5. Which causes most frequently caused delays? (`sorted_delays_by_cause`)

Further exploration

Are there other questions you have about the data? Maybe you want to know when the best time to travel on the subway is. Maybe you want to know which stations have the most and least delays. Maybe you want to know which subway lines have the least delays. If you have time, feel free to explore the questions that most interest you, by writing functions to analyze the data!