

```
In [ ]: from cs103 import *
```

# CPSC 103 - Systematic Program Design

## Module 08 Day 2

Rik Blok, with thanks to Jessica Wong and Giulia Toti

---

### Reminders

- **Fri:** Module 6 (One Task Per Function): Tutorial Resubmission
- No tutorial resubmission for Module 7
- No tutorial for Module 8
- Starting **this** week: Tutorial sessions (Thu) will be open office hours, for project help
- Lecture time next Tue/Thu also open office hours

See your Canvas calendar (<https://canvas.ubc.ca/calendar>) for details.

---

### pyplot from Matplotlib library

In order to use `pyplot`, we first have to import it:

```
In [ ]: import matplotlib.pyplot as pyplot
```

## Exercise 1: Analysing VPD Crime Data

### Step 1: Planning - Highlights

#### Step 1a: Identify the info your program will read

- **TYPE:** The type of crime activities. One of
  - BNE Commercial
  - BNE Residential/Other
  - Theft of Vehicle
  - Theft of Bicycle
- **HOUR, MINUTE:** when the reported crime activity occurred
  - **HOUR:** A two-digit field that indicates the hour time (in 24 hours format)
  - **MINUTE:** A two-digit field that indicates the minute
  - **Note:** Some crimes may not contain time information.

#### Step 1b: Write a description of what your program will produce

- Given a type of crime, find the time of day (hour) with the highest frequency

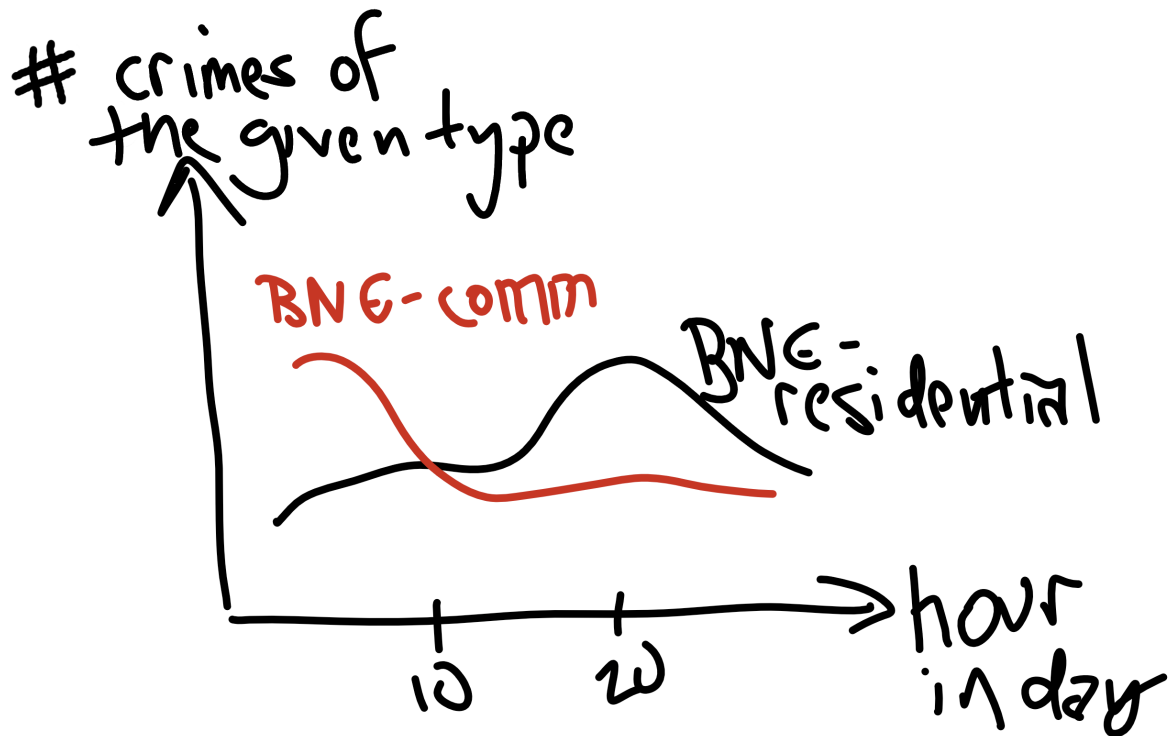
OR

- Draw a graph of crimes committed in each hour, maybe show different types overlaid in different colours

### Step 1c: Write or draw examples of what your program will produce

```
expect(main('crime_data_file.csv', CrimeType.BEC), 8)
```

OR



### Step 2a: Design data definitions

Document which information you will represent in your data definitions

We want to represent the type of crime and the hour it occurred.

Type of crime best represented by an enumeration (4 cases).

Hour can be represented by an interval, integer in the range [0,23].

We'll also check the minute, just to see if the data is reliable. But we don't need to store the minute.

#### Design data definitions

► Jump to...

⚠ The [magic command](#) %run -i ... below just loads the specified file into the current cell and runs it.

I've included it here to save space, since we've already seen this code a few times... most recently in Module 07b Day 1. You are **NOT** permitted to use magic commands in your project submission.

You can view the data definitions in a separate file here: [module07b\\_day1\\_step2a.py](#).

Run the following cell to load and run the file.

```
In [ ]: # load and run data definitions for
#      * CrimeType
#      * CrimeData
#      * List[CrimeData]
#      * List[str]
#      * List[int]

%run -i module07b_day1_step2a.py
```

## Step 2b: Design read function

► Jump to...

Design a function to read the information and store it as data in your program

You can view the `read` function and its helpers in a separate file here: [module07b\\_day1\\_step2b.py](#).

Run the following cell to load and run the file.

```
In [ ]: # load and run functions
#      * read
#      * and helpers
#      * parse_crime_type
#      * is_reliable
#      * crime_type_as_str (new)

%run -i module07b_day1_step2b.py
```

## Step 2c: Design analyze function

Design functions to analyze the data

### ✅ Continuing on Day 2

To make this manageable in class, we started Day 1 with the solution (including test data and helper functions) from Module 07b Day 1. We got as far as rewriting the composition steps in `analyze` :

```
# template based on composition
# Step 1: filter list for crime type
crimes_of_type = filter_for_crime_type(lccd, crime_type)
# Step 2: build a list of hours of the day
hours = hours_in_a_day()
# Step 3: make list of crime counts for all hours in the day
crime_count = count_crimes_by_hour(hours, crimes_of_type)
# Step 4: plot crimes per hour versus hour
plot_crimes_by_hour(hours, crime_count)
```

Let's jump down to the `count_crimes_by_hour` [helper](#) and design that helper there.

### Parsed test data

► Jump to...

Here are the test files parsed into `List[CrimeData]` . I've included this info here so we can quickly add it as needed to our examples. Let's skip down to the [main function](#) for now and we'll come back to it.

```

In [ ]: # from 'testfile_empty.csv'
TEST_EMPTY = []

# from 'testfile_all_missing.csv'
TEST_ALL_MISSING = [CrimeData(CrimeType.BEC, 0),
                    CrimeData(CrimeType.BER, 0),
                    CrimeData(CrimeType.TB, 0),
                    CrimeData(CrimeType.TV, 0)] # but none of these should be read

# from 'testfile_all_bec.csv'
TEST_ALL_BEC = [CrimeData(CrimeType.BEC, 6),
               CrimeData(CrimeType.BEC, 18)] # missing data removed

# from 'testfile_all_ber.csv'
TEST_ALL_BER = [CrimeData(CrimeType.BER, 21),
               CrimeData(CrimeType.BER, 17),
               CrimeData(CrimeType.BER, 0)]

# from 'testfile_all_tb.csv'
TEST_ALL_TB = [CrimeData(CrimeType.TB, 1),
               CrimeData(CrimeType.TB, 23),
               CrimeData(CrimeType.TB, 17)]

# from 'testfile_all_tv.csv'
TEST_ALL_TV = [CrimeData(CrimeType.TV, 23),
               CrimeData(CrimeType.TV, 14),
               CrimeData(CrimeType.TV, 21)]

# from 'testfile_all_types.csv'
TEST_ALL_TYPES = [CrimeData(CrimeType.BEC, 1),
                  CrimeData(CrimeType.BER, 2),
                  CrimeData(CrimeType.TB, 3),
                  CrimeData(CrimeType.TV, 4)]

# from 'testfile_all_bec_hour_6.csv'
TEST_ALL_BEC_HOUR_6 = [CrimeData(CrimeType.BEC, 6),
                       CrimeData(CrimeType.BEC, 6)] # missing data removed

# from 'testfile_all_ber_hour_0.csv'
TEST_ALL_BER_HOUR_0 = [CrimeData(CrimeType.BER, 0),
                       CrimeData(CrimeType.BER, 0),
                       CrimeData(CrimeType.BER, 0)]

```

**Helper function:** filter\_for\_crime\_type

► [Jump to...](#)

Here's a helper function (and it's lower-level helper) that we'll use later. Let's skip down to the [main function](#) for now and we'll come back to it.

```

In [ ]: @typecheck
def filter_for_crime_type(locd: List[CrimeData], ct: CrimeType) -> List[CrimeData]:
    """
    Returns only items in locd that have crime type ct.
    """
    # return [] # stub

    # template from List[CrimeData]

    # description of the accumulator
    matches = [] # type: List[CrimeData]

    for cd in locd:
        if is_crime_type(cd, ct):
            matches.append(cd)

    return matches

@typecheck
def is_crime_type(cd: CrimeData, ct: CrimeType) -> bool:
    """
    Returns True if cd has crime type `ct`, otherwise returns False.
    """
    # return False # stub

    # template from CrimeData with additional parameter ct
    return cd.type == ct

# Examples and tests for is_crime_type
start_testing()

# Test 1: does match
# Test 2: doesn't match
expect(is_crime_type(CrimeData(CrimeType.BEC, 0), CrimeType.BEC), True) # Test 1
expect(is_crime_type(CrimeData(CrimeType.BER, 1), CrimeType.BER), True) # Test 1
expect(is_crime_type(CrimeData(CrimeType.TB, 0), CrimeType.TV), False) # Test 2
expect(is_crime_type(CrimeData(CrimeType.TB, 2), CrimeType.TV), False) # Test 2

summary()

# Examples and tests for filter_for_crime_type
start_testing()

# Test 1: empty list
# Test 2: crime type doesn't match any
# Test 3: crime type matches some
expect(filter_for_crime_type([], CrimeType.BEC), []) # Test 1
expect(filter_for_crime_type(TEST_ALL_BEC, CrimeType.TV), []) # Test 2
expect(filter_for_crime_type(TEST_ALL_TB, CrimeType.BER), []) # Test 2
expect(filter_for_crime_type(TEST_ALL_BEC+TEST_ALL_BER, CrimeType.BEC), TEST_ALL_BEC) # Test 3
expect(filter_for_crime_type(TEST_ALL_BEC+TEST_ALL_BER, CrimeType.BER), TEST_ALL_BER) # Test 3

summary()

```

**Helper function:** `hours_in_a_day`

► [Jump to...](#)

Here's a helper function that we'll use later. Let's skip down to the [main function](#) for now and we'll come back to it.

```
In [ ]: @typecheck
def hours_in_a_day() -> List[int]:
    """
    Returns a list of the hours in a day: [0,23].
    """
    # return [] # stub

    # no template
    hours = []

    for h in range(24): # range is like a list, but subtly different
        hours.append(h)
    # or just hours = list(range(24))

    return hours

# Examples and tests for hours_in_a_day
start_testing()

expect(hours_in_a_day(), [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
summary()
```

**Another helper function for** analyze

► [Jump to...](#)

Let's apply the HtDF recipe to design a new `count_crimes_by_hour` function. Later we'll skip down to the [main function](#) to test our program. We may also want to review the [parsed test data](#).

```

In [ ]: # BOOKMARK: On Day 2 we'll redesign our old `find_hour_with_most_crimes`
        # function into a new `count_crimes_by_hour` which will serve
        # as a helper for `analyze` to generate our plot data.

        @typecheck
        def find_hour_with_most_crimes(hours: List[int], lccd: List[CrimeData]) -> int:
            """
            Returns hour in `hours` for which lccd has the most occurrences.

            In case of a tie, returns earliest hour.

            Assumes `hours` is not empty.
            """
            # return -1 # stub

            # template from List[int] with extra parameter lccd

            # maximum number of crimes by hour in list so far
            max_crimes = 0 # type: int

            # hour of maximum crimes in list so far
            hour_of_max_crimes = hours[0] # type: int

            for h in hours:
                crimes_in_hour = count_crimes_in_hour(lccd, h)
                if crimes_in_hour > max_crimes:
                    max_crimes = crimes_in_hour
                    hour_of_max_crimes = h

            return hour_of_max_crimes

        @typecheck
        def count_crimes_in_hour(lccd: List[CrimeData], hour: int) -> int:
            """
            Returns the number of crimes from `lccd` that occur at a particular `hour`.
            """
            # return -1 # stub

            # template from List[CrimeData] with extra parameter hour

            # count of crimes in given hour in list so far
            count = 0 # type: int
            for cd in lccd:
                if is_crime_in_hour(cd, hour):
                    count = count + 1
            return count

        @typecheck
        def is_crime_in_hour(cd: CrimeData, hour: int) -> bool:
            """
            Returns True if crime `cd` occurred during `hour`, otherwise False.
            """
            # return False # stub

            # template from CrimeData with extra parameter hour
            return cd.hour == hour

        # Examples and tests for is_crime_in_hour
        start_testing()

        # Test 1: Crime is not in hour
        # Test 2: Crime is in hour
        expect(is_crime_in_hour(CrimeData(CrimeType.BEC, 7), 0), False) # Test 1
        expect(is_crime_in_hour(CrimeData(CrimeType.BEC, 7), 7), True) # Test 2
        expect(is_crime_in_hour(CrimeData(CrimeType.TV, 0), 0), True) # Test 2

```

```
summary()

# Examples and tests for count_crimes_in_hour
start_testing()


# Test 1: Empty crime data list
# Test 2: Not empty but no crimes in given hour
# Test 3: Crimes in given hour, of various types
expect(count_crimes_in_hour([], 1), 0) # Test 1
expect(count_crimes_in_hour(TEST_ALL_BER, 1), 0) # Test 2
expect(count_crimes_in_hour(TEST_ALL_BER, 17), 1) # Test 3
expect(count_crimes_in_hour(TEST_ALL_TB+TEST_ALL_TV, 23), 2) # Test 3
expect(count_crimes_in_hour(TEST_ALL_TB+TEST_ALL_TV, 17), 1) # Test 3
expect(count_crimes_in_hour(TEST_ALL_TV+TEST_ALL_TB, 23), 2) # Test 3 (crimes shuffled)

summary()

# Examples and tests for find_hour_with_most_crimes
start_testing()

# Test 1: Empty crime data list
# Test 2: Not empty but no crimes in given hours
# Test 3: Crimes in given hours, of various types
expect(find_hour_with_most_crimes([1], []), 1) # Test 1
expect(find_hour_with_most_crimes([5, 6, 7, 8], TEST_ALL_TYPES), 5) # Test 2
expect(find_hour_with_most_crimes([8, 7, 6, 5], TEST_ALL_TYPES), 8) # Test 2 (hours reversed)
expect(find_hour_with_most_crimes([3, 4, 5, 6], TEST_ALL_TYPES), 3) # Test 3
expect(find_hour_with_most_crimes([6, 5, 4, 3], TEST_ALL_TYPES), 4) # Test 3 (hours reversed)
expect(find_hour_with_most_crimes([1, 14, 17, 21, 23], TEST_ALL_TB+TEST_ALL_TV), 23) # Test 3
expect(find_hour_with_most_crimes([1, 14, 17, 21, 23], TEST_ALL_TV+TEST_ALL_TB), 23) # Test 3

summary()
```

►  Sample solution (For later. Don't peek if you want to learn 😊)

**Helper function:** `plot_crimes_by_hour`

► Jump to...

To draw a graph, we'll build a helper function `plot_crimes_by_hour` here by copying code from a relevant [Worked Example](#).

Let's skip down to the [main function](#) for now and we'll come back to it.

In [ ]:

**main and analyze functions**

► Jump to...

Here are our `main` and `analyze` functions from Day 1.

Recall, our purpose is:

Draw a graph of crimes committed in each hour, maybe show different types overlaid in different colours



```

In [ ]: #####
# Functions

@typecheck
def main(filename: str,
          crime_type: CrimeType) -> None:
    """
    Reads the file from given filename, analyzes the data, returns the result
    """
    # Template from HtDAP, based on function composition
    return analyze(read(filename), crime_type)

@typecheck
def analyze(lcld: List[CrimeData],
            crime_type: CrimeType) -> None:
    """
    Plots the number of crimes of a particular
    type for each hour of the day from the list
    lcld.
    """

    # return None # stub

    # template based on composition
    # Step 1: filter list for crime type
    crimes_of_type = filter_for_crime_type(lcld, crime_type)
    # Step 2: build a list of hours of the day
    hours = hours_in_a_day()
    # Step 3: make list of crime counts for all hours in the day
    crime_count = count_crimes_by_hour(hours, crimes_of_type)
    # Step 4: plot crimes per hour versus hour
    plot_crimes_by_hour(hours, crime_count)

    return None

# Examples and tests for analyze
start_testing()

# Test 1: empty list
# Test 2: no matching crime type
# Test 3: some matching crime types
expect(analyze([], CrimeType.BEC), None) # Test 1
# We expect an empty plot like this:
#
# number
# of crimes
# |
# 1 |
# |
# 0 +----- hour
#   0    6    12   18
expect(analyze(TEST_ALL_BEC, CrimeType.BER), None) # Test 2
# We expect an empty plot like this:
#
# number
# of crimes
# |
# 1 |
# |
# 0 +----- hour
#   0    6    12   18
expect(analyze(TEST_ALL_BEC, CrimeType.TB), None) # Test 2
# We expect an empty plot like this:
#
# number
# of crimes
# |
# 1 |

```

```

# /
# 0 +----- hour
# 0 6 12 18
expect(analyze(TEST_ALL_BEC, CrimeType.TV), None) # Test 2
# We expect an empty plot like this:
#
# number
# of crimes
# /
# 1 |
# /
# 0 +----- hour
# 0 6 12 18
expect(analyze(TEST_ALL_BEC, CrimeType.BEC), None) # Test 3
# We expect a plot similar to this:
#
# number
# of crimes
# /
# 1 | *
# / |
# 0 +---|-----|----- hour
# 0 6 12 18
expect(analyze(TEST_ALL_BER_HOUR_0, CrimeType.BER), None) # Test 3
# We expect a plot similar to this:
#
# number
# of crimes
# /
# 3 *
# / \
# 2 ||
# / \
# 1 ||
# / \
# 0 +---|-----|----- hour
# 0 6 12 18
expect(analyze(TEST_ALL_BEC+TEST_ALL_BEC_HOUR_6, CrimeType.BEC), None) # Test 3
# We expect a plot similar to this:
#
# number
# of crimes
# /
# 3 | *
# / |
# 2 | |
# / |
# 1 | |
# / |
# 0 +---|-----|----- hour
# 0 6 12 18
expect(analyze(TEST_ALL_BEC_HOUR_6+TEST_ALL_BEC, CrimeType.BEC), None) # Test 3
# We expect a plot similar to this:
#
# number
# of crimes
# /
# 3 | *
# / |
# 2 | |
# / |
# 1 | |
# / |
# 0 +---|-----|----- hour
# 0 6 12 18

summary()

# Examples and tests for main

```

```

start_testing()

# Test 1: empty file
# Test 2: invalid data
# Test 3: no matching crime types
# Test 4: some matching crime types
expect(main('testfile_empty.csv', CrimeType.BEC), None) # Test 1
# We expect an empty plot like this:
#
# number
# of crimes
# |
# 1 |
# |
# 0 +----- hour
#   0     6    12    18
expect(main('testfile_all_missing.csv', CrimeType.BER), None) # Test 2
# We expect an empty plot like this:
#
# number
# of crimes
# |
# 1 |
# |
# 0 +----- hour
#   0     6    12    18
expect(main('testfile_all_bec.csv', CrimeType.BER), None) # Test 3
# We expect an empty plot like this:
#
# number
# of crimes
# |
# 1 |
# |
# 0 +----- hour
#   0     6    12    18
expect(main('testfile_all_bec.csv', CrimeType.TB), None) # Test 3
# We expect an empty plot like this:
#
# number
# of crimes
# |
# 1 |
# |
# 0 +----- hour
#   0     6    12    18
expect(main('testfile_all_bec.csv', CrimeType.TV), None) # Test 3
# We expect an empty plot like this:
#
# number
# of crimes
# |
# 1 |
# |
# 0 +----- hour
#   0     6    12    18
expect(main('testfile_all_bec.csv', CrimeType.BEC), None) # Test 4
# We expect a plot similar to this:
#
# number
# of crimes
# |
# 1 |      *      *
#   |      |      |
# 0 +---+-----+----- hour
#   0     6    12    18
expect(main('testfile_all_ber_hour_0.csv', CrimeType.BER), None) # Test 4
# We expect a plot similar to this:
#
# number

```

```
# of crimes
# |
# 3 *
# /\
# 2 //
# //
# 1 //
# //
# 0 +----/-----/----- hour
# 0 6 12 18
```

summary()

►  Sample solution (For later. Don't peek if you want to learn 😊)

## Final Graph/Chart

Now that everything is working, you **must** call `main` on the intended information source in order to display the final graph/chart:

### BNE Commercial

```
In [ ]: main('crimedata_subset_bne_theft_of_bike_veh_2018.csv',
             CrimeType.BEC) # BNE Commercial
```

### BNE Residential/Other

```
In [ ]: main('crimedata_subset_bne_theft_of_bike_veh_2018.csv',
             CrimeType.BER) # BNE Residential/Other
```

### Theft of Bicycle

```
In [ ]: main('crimedata_subset_bne_theft_of_bike_veh_2018.csv',
             CrimeType.TB) # Theft of Bicycle
```

### Theft of Vehicle

```
In [ ]: main('crimedata_subset_bne_theft_of_bike_veh_2018.csv',
             CrimeType.TV) # Theft of Vehicle
```

## Exercise 2: Multi-line chart

Recall, our goal was to

Draw a graph of crimes committed in each hour, **maybe show different types overlaid in different colours**

To overlay different lines we need to create a multi-line chart.

```
In [ ]: x_vals = [0, 1, 2, 3, 4]
x_squared = [0, 1, 4, 9, 16]
x_cubed = [0, 1, 8, 27, 64]
pyplot.plot(x_vals, x_squared, label='$x^2$')
pyplot.plot(x_vals, x_cubed, label='$x^3$')
pyplot.legend()
pyplot.show()
```


► Jump to...

## Multiline chart of crime data

Let's start by copying our `main`, `analyze`, and `plot_crimes_by_hour` functions from the cell above. Then we'll revise them to draw a multi-line chart with all four crime types.

```
In [ ]: # TODO: Copy `plot_crimes_by_hour` from above.
```

```
In [ ]: # TODO: Copy `main` and `analyze` from above.
```

►  Sample solution (For later. Don't peek if you want to learn 😊)

### ✓ Re-use functions when possible!

For example, we didn't need to write four different plot functions for the different crime types. We designed **one** function and **called it** multiple times.

Here we come to the end! Let's see the final result:

```
In [ ]: main('crimedata_subset_bne_theft_of_bike_veh_2018.csv')
```

## That's all, folks!

We've spent the last few weeks building up the knowledge we need to design an analysis program and visualize information. These are skills that will be useful for your project.

But beyond that, you are now able to:

- Write small programs that solve a problem in an academic discipline of your choice and are readable, well-organized, well-documented, and well-tested.
- Write small programs for a reasonably complex task, where the ability to use the one task - one function rule can be demonstrated.
- Design the data representation for a reasonably complex problem.
- Describe the information encoded in given data.

## After the course...

- You will have access to Syzygy as long as you are a registered UBC student.
- After the term ends, access Syzygy by going to (<https://ubc.syzygy.ca/>).
- If you want to download a copy of your files, see [here](#) for instructions.

## Interested in More Python?

- [CPSC 203](#) - Use Python to learn about data structures other than lists and solve some really interesting problems!
- [CPSC 107](#) - Important if you want to apply for a CS major! Uses a subset of the Racket language.
- [CPSC 110](#) - Either CPSC 103+107 or CPSC 110 is required for all CS majors.

## Interested in Computer Science?

- Consider majoring in CS! (There are lots of combined major/honours programs like MICB/CS, Business/CS, Physics/CS, etc.)
- About to graduate? Consider the BCS program. Some of our TAs are in the BCS program so feel free to ask them for their thoughts on the program!

---

## Thanks!

Thanks for your great participation all term! You've been a pleasure to teach! I hope to see you in CPSC 107 next term!

▶  My cat Tom would also like to say... 😊)

Remember, next week we'll hold office hours in lectures and tutorials.

---