In [ ]:
```
from cs103 import *


```

CPSC 103 - Systematic Program Design

Module 08 Day 1

Rik Blok, with thanks to Giulia Toti

Reminders

- Wed: Module 7 (HtDAP): Code Review
- Wed: Module 7 (HtDAP): Tutorial
- **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 will be open office hours, for project help
- Lecture time next Tue/Thu also open office hours

ℹ️ From the syllabus:

> To pass the course, you must achieve all of the following conditions:
>
> 1. earn a final course grade of at least 50%,
> 2. **pass the project**, and
> 3. pass the final exam.

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

Module learning goals

By the end of this module, you will be able to:

- Design functions that produce plots or graphs using `pyplot`
- Visualize data to communicate the results of your analysis
- Draw line charts, bar charts, scatter plots, pie charts, and histograms

---

`pyplot` from Matplotlib library

Visualizing data is a very effective way to communicate the results of your analysis. We will use the Matplotlib tool `pyplot` to draw our graphs. See the summary documentation for lots of information.

✅ We'll just have a quick overview of `pyplot` here. Check the provided links and the Module 8 Worked Examples on Jupyter for more details.

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

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


```

Line chart

We now have access to a variety of tools to plot data. Let's start by plotting a line based on the points in the table on the right:

x	y
0	0
1	1
2	4
3	9
4	16

> ⚠ Make sure your lists of x- and y-values have the same length! Otherwise, `pyplot.plot` will report an error.
>
> It's easy here but you'll need to be careful in your project, when you're deciding what should go on each axis. Part of your planning will be to decide what goes where and to make sure they make sense together.

In [ ]:
```
pyplot.plot([0,1,2,3,4], [0,1,4,9,16])
pyplot.show()


```

Each `pyplot...` function call makes changes to the current figure, then `pyplot.show` draws it.

> ℹ `pyplot.show()` can be used to display a graph in Python. You may notice plots still show without it. That's because Jupyter automatically calls `pyplot.show()` at the end of every cell. But it's good practice to include it anyway.

`pyplot.plot` help

`pyplot.plot` is a function. The first argument is the list of values for the x-axis, the second argument the values for the y-axis. You can call `help(pyplot.plot)` to learn more but you might find this Pyplot tutorial more friendly:

In [ ]:
```
help(pyplot.plot)


```

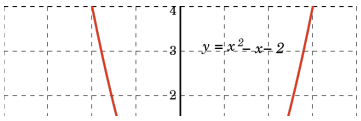
Using variables

We could have achieved the same result using list variables to store the x- and y-values:

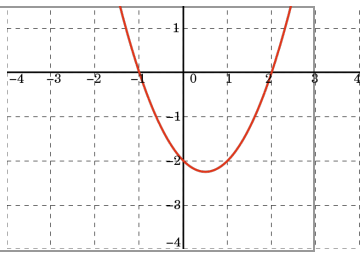
In [ ]:
```
x_vals = [0, 1, 2, 3, 4]
y_vals = [0, 1, 4, 9, 16]
pyplot.plot(x_vals, y_vals)
pyplot.show()


```

> ℹ **Dependent and independent variables**



> In single variable calculus, a function is typically graphed with the horizontal axis representing the independent variable and the vertical axis representing the dependent variable. In this function $[y = f(x) = x^2 - x - 2]$, y is the **dependent** variable and x is the **independent** variable.
>
> – Wikipedia



Title and axis labels

You can add text, such as a graph title and axis labels, to your plot. See the Matplotlib Text in Matplotlib Plots tutorial for details. Notice you can add some mathematical notation in your text.

The following will add a title and a label to each axis:

In [ ]:
```
pyplot.title('Graph of $y=x^2$ versus $x$') # use $...$ for math
pyplot.xlabel('X-axis')
pyplot.ylabel('Y-axis')
pyplot.plot(x_vals, y_vals)
pyplot.show()


```

Axis limits

You can modify the x- and y-axis limits with the pyplot.axis function.

> ⚠ Be careful if you choose to use fixed values for your axis limits. You might inadvertantly crop important data out of your visualization!

In [ ]:
```
pyplot.axis([-2.5, 2.5, -10, 10]) # [xmin, xmax, ymin, ymax]
pyplot.plot(x_vals, y_vals)
pyplot.show()


```

`pyplot.setp` to set properties

You can set various properties of your plot with the pyplot.setp function. For example,

In [ ]:
```
# assign the plot to a variable so you can refer to it in setp
line = pyplot.plot(x_vals, y_vals)
pyplot.setp(line, color='r', linewidth=2.0, marker='o',
            linestyle='--')
pyplot.show()


```

Keyword arguments

The call to `pyplot.setp(..., color='r', linewidth=2.0, marker='o', linestyle='--')` demonstrates Python's keyword argument feature. Some functions accept arguments that are identified by a name, rather than their position in the list of parameters. Only unnamed parameters need to be kept in a fixed position in the argument list.

> ℹ️ Keyword arguments are **NOT** a core concept of this course. For our purposes, you just need to be aware that some of the `pyplot` functions use keyword arguments.

---

iClicker Question: Keyword arguments

Which of the following sets the properties of `line` to be a **thick, red, dashed line with circle markers**, like the plot above? Select ALL that apply. [Set question type to "Multiple Answer".]

A. `pyplot.setp(line, linewidth=2.0, color='r', linestyle='--', marker='o')`
B. `pyplot.setp(line, color='r', linewidth=2.0, marker='o', linestyle='--')`
C. `pyplot.setp(line, marker='x', linewidth=1.0, color='g', linestyle='-')`
D. `pyplot.setp(line, linewidth=2.0, linestyle='--', marker='o', color='r')`
E. `pyplot.setp(linewidth=2.0, line, color='r', linestyle='--', marker='o')`

> ▶ ℹ️ Hints (for your review after class)

Multi-line chart

You can plot several lines on a single plot by plotting each before calling `pyplot.show`. You can include a legend by giving each line a `label` and calling the pyplot.legend function. For example,

In [ ]:
```
1 x_squared = [0, 1, 4, 9, 16]
2 x_cubed = [0, 1, 8, 27, 64]
3 pyplot.plot(x_vals, x_squared, label='$x^2$')
4 pyplot.plot(x_vals, x_cubed, label='$x^3$')
5 pyplot.legend()
6 pyplot.show()
7
8
```

Other types of plots

Line charts are far from the only type of plot available. Here are a few examples of other plots you can produce with this library. For more details, please check out the worked examples on Jupyter.

> ⚠️ This is the last module of the course, and it is very important for your **project**. For the **exam** however, we will only ask about line charts (not histograms, pie charts, bar charts, scatter plots, or any other type of chart).
>
> You should know how to use `pyplot` to draw a line chart with a title and axis labels.

Bar chart

Notice that with a bar chart, the x-values can be strings (categories) instead of numbers.

In [ ]:
```
x_vals = ['A','B','C','D','E']
y_vals = [0, 1, 4, 9, 16]
pyplot.bar(x_vals, y_vals)
pyplot.show()


```

Scatter plot

A scatter plot is similar to a line chart except each point is shown with a separate marker, without lines connecting them. Unlike a line chart, the order of the points in a scatter plot doesn't matter (as long as the x- and y-value lists are in the same order as each other).

In [ ]:
```
x_vals = [1, 0, 4, 2, 3]
y_vals = [1, 0,16, 4, 9]
pyplot.scatter(x_vals, y_vals)
pyplot.show()


```

Pie chart

A pie chart is shown here for completeness. However, they are not recommended because

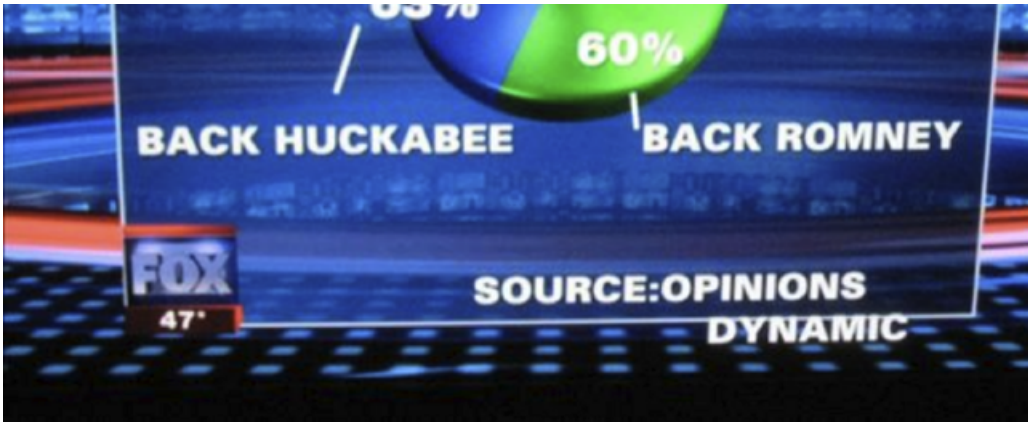
> ...research has shown it is difficult to compare different sections of a given pie chart, or to compare data across different pie charts.
>
> ...
>
> Statisticians generally regard pie charts as a poor method of displaying information, and they are uncommon in scientific literature. One reason is that it is more difficult for comparisons to be made between the size of items in a chart when area is used instead of length and when different items are shown as different shapes.
>
> Further, in research performed at AT&T Bell Laboratories, it was shown that comparison by angle was less accurate than comparison by length. Most subjects have difficulty ordering the slices in the pie chart by size; when an equivalent bar chart is used the comparison is much easier.
>
> – Wikipedia

In a pie chart, all values must be non-negative. The fractional area of each wedge is the same as the fraction of the value relative to the total.

In [ ]:
```
vals = [15, 25, 42, 18, 24]
pyplot.pie(vals, labels = ['A','B','C','D','E'])
pyplot.show()


```

A pie chart is seldom the best choice. And sometimes it's just plain **wrong**!



Histogram

A histogram is used for showing the distribution of a number of samples. I haven't included it here because:

1. It works slightly differently than the other charts and
2. It does some of the "heavy lifting" for you. That can make it dangerous to use in a course like this, where we want you to demonstrate you understand **how** to solve a problem.

But you are welcome to use a histogram **IF** you're sure your program is still doing *substantial computation* and your TA has vetted your proposed graph.

HtDAP with visualizations

Instead of returning a value, your `analyze` function can produce a plot.

```
@typecheck
def analyze(loc: List[Consumed]) -> None:
    """
    Plots the ...
    """
    return None
```

If the `analyze` function draws a visualization, then we always make three changes from the template:

1. The function return type is `None` instead of `Produced`.
2. The purpose describes the plot that will be generated.
3. The function **explicitly** returns `None` (first in the stub and then at the last line of the implementation).

> ℹ️ Since `analyze` returns `None` and `main` just returns the result of `analyze`, it returns `None`, too.
>
> ```
> @typecheck
> def main(filename: str) -> None:
>     ...
>     return analyze(read(filename))
> ```

Examples and tests for graphs

How do we test that a chart is correct? We test our `analyze` function (or whatever we've renamed it to) in two ways:

1. Check the return value

We expect the return value to be `None`, so we should make sure:

```
start_testing()

# Examples and tests for analyze
expect(analyze(...), None)

summary()
```

But the test only confirms that the function doesn't return a value. It won't be able to tell you that the graph is correct. For that, you need to

2. Do a visual inspection

Draw the user a sample of what the graph should look like with the example data.

ASCII art

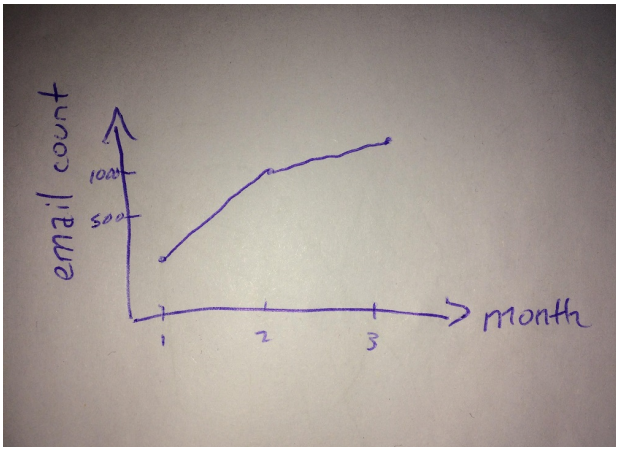
You can type it into the code it with ASCII art like this:

```
start_testing()

# Examples and tests for analyze
expect(analyze(...), None)
# Should produce a graph that looks a bit like:
#
#   email
#   count
#       |          ----*
# 1000 +         *
#       |       /
#  500 +     /
#       |   *
#       +---+----+----+----  month
#           1    2    3

summary()
```

Or attach a hand-drawn example



Similar to the sample graph you drew for your project proposal. But now, the values and shape shouldn't be made up but should come from the example data for the test.

Insert the images just below the cell that runs the tests (drag and drop into a cell you're editing or select the menu item `Edit > Insert Image`). One image per cell.

When running the tests, you will compare the actual output to the expected graphs.

Template based on visualization

We don't have a generic template for visualizations. Instead, we recommend you start your `analyze` function – or appropriate helper – by copying code from a relevant Worked Example. When you do, include the comment

```
# Template based on visualization
...
```

at the top.

Next, modify the copied code in your HtDF "implementation" step.

Implementing `analyze`

After taking the template from a worked example, our `analyze` function – or appropriate helper – will look something like the following:

In [ ]:

```
from typing import List
Consumed = None # just for this example

@typecheck
def analyze(loc: List[Consumed]) -> None:
    """
    Plots the ...
    """
    # return None # stub
    # Template based on visualization

    # set the x-axis label, y-axis label, and plot title
    pyplot.xlabel('hours')
    pyplot.ylabel('fish caught')
    pyplot.title('Fish caught over time')

    # range for the axes
    # [x-min, x-max, y-min, y-max]
    pyplot.axis([0,4,0,12])

    # plot our data
    line = pyplot.plot(hours, fish_caught)

    # set some properties for the line (color to red, line width to 2, and marker to a s
    pyplot.setp(line, color='r', linewidth=2.0, marker="o")

    # show the plot
    pyplot.show()

    return None


```

It remains for us to adapt the calls to `pyplot...` to perform the purpose of `analyze`.

That may involve adding or removing `pyplot...` function calls.

We'll also need to generate the lists of x- and y-values so we can pass them to our plotting function. We'll design helper functions as needed.

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

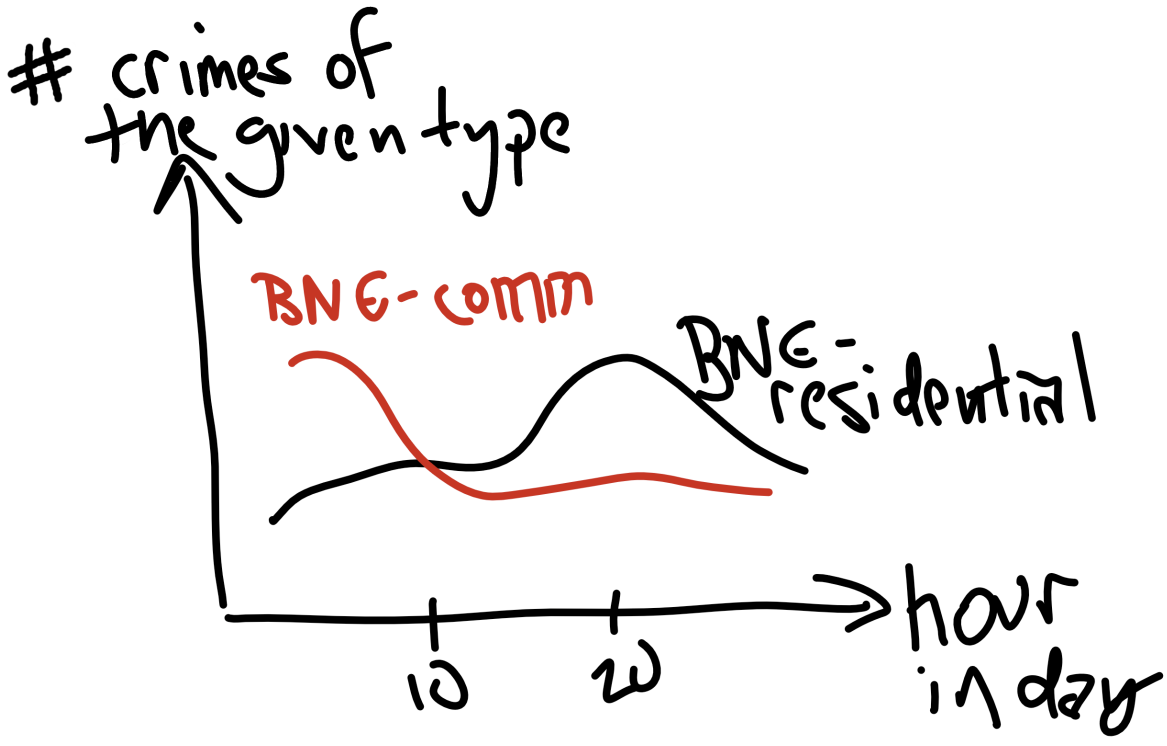
OR

- **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

```

▸ Jump to...

Step 2b: Design read function

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

✅ To make this manageable in class, we will start with the solution (including test data and helper functions) from

Module 07b Day 1. Recall, the problem there was a little different:

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

Our new goal is to

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

▸ Jump to...

Parsed test data

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)]


```

▸ Jump to...

Helper function: `filter_for_crime_type`

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 [ ]:

```python
@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) #
expect(filter_for_crime_type(TEST_ALL_BEC+TEST_ALL_BER, CrimeType.BER), TEST_ALL_BER) #

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,

summary()

```

▸ Jump to...

Another helper function for `analyze`

Here's a helper function (and it's lower-level helper) that we'll replace later. Let's skip down to the `main` function for now and we'll come back to it. We'll also want to review the parsed test data.

In [ ]:

```
@typecheck
def find_hour_with_most_crimes(hours: List[int], locd: List[CrimeData]) -> int:
    """
    Returns hour in `hours` for which locd 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 locd

    # 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(locd, 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(locd: List[CrimeData], hour: int) -> int:
    """
    Returns the number of crimes from `locd` 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 locd:
        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 rev
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 rev
expect(find_hour_with_most_crimes([1, 14, 17, 21, 23], TEST_ALL_TB+TEST_ALL_TV), 23) #
expect(find_hour_with_most_crimes([1, 14, 17, 21, 23], TEST_ALL_TV+TEST_ALL_TB), 23) #

summary()
```

▶ ℹ️ Sample solution (For later. Don't peek if you want to learn 🙂)

▶ Jump to...

`main` and `analyze` functions

Here are our `main` and `analyze` functions from Module 07b Day 1. Let's start here.

Later we'll look at the parsed test data and these helpers, from above:

- `filter_for_crime_type`
- `hours_in_a_day`
- Another helper function for `analyze`

Recall, our purpose is:

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

To draw a graph, we'll build a helper function for `analyze` by copying code from a relevant Worked Example.

In [ ]:

```
###########
# Functions

@typecheck
def main(filename: str,
         crime_type: CrimeType) -> int:
    """
    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(locd: List[CrimeData],
            crime_type: CrimeType) -> int:
    """
    Returns the hour at which crimes of type
    crime_type were most common in the list
    locd.

    Returns earliest hour in the case of a tie.
    """

    # return -1 # stub

    # template based on composition
    # Step 1: filter list for crime type
    crimes_of_type = filter_for_crime_type(locd, crime_type)
    # Step 2: build a list of hours of the day
    hours = hours_in_a_day()
    # Step 3: find hour of the day most common in list
    worst_hour = find_hour_with_most_crimes(hours, crimes_of_type)
    # Step 4: return that hour
    return worst_hour

# 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), 0) # Test 1
expect(analyze(TEST_ALL_BEC, CrimeType.BER), 0) # Test 2
expect(analyze(TEST_ALL_BEC, CrimeType.TB), 0) # Test 2
expect(analyze(TEST_ALL_BEC, CrimeType.TV), 0) # Test 2
expect(analyze(TEST_ALL_BEC, CrimeType.BEC), 6) # Test 3
expect(analyze(TEST_ALL_BER_HOUR_0, CrimeType.BER), 0) # Test 3
expect(analyze(TEST_ALL_BEC+TEST_ALL_BEC_HOUR_6, CrimeType.BEC), 6) # Test 3
expect(analyze(TEST_ALL_BEC_HOUR_6+TEST_ALL_BEC, CrimeType.BEC), 6) # Test 3

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), 0) # Test 1
expect(main('testfile_all_missing.csv', CrimeType.BER), 0) # Test 2
expect(main('testfile_all_bec.csv', CrimeType.BER), 0) # Test 3
expect(main('testfile_all_bec.csv', CrimeType.TB), 0) # Test 3
expect(main('testfile_all_bec.csv', CrimeType.TV), 0) # Test 3
expect(main('testfile_all_bec.csv', CrimeType.BEC), 6) # Test 4
expect(main('testfile_all_ber_hour_0.csv', CrimeType.BER), 0) # Test 4

```

```
summary()
```

▶ ℹ 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
```

In [ ]:
```
```