

```
In [ ]: 1 from cs103 import *
        2
        3
```

# CPSC 103 - Systematic Program Design

## Module 08 Day 2

Rik Blok, with thanks to 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 [ ]: 1 import matplotlib.pyplot as pyplot
        2
        3
```

## 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

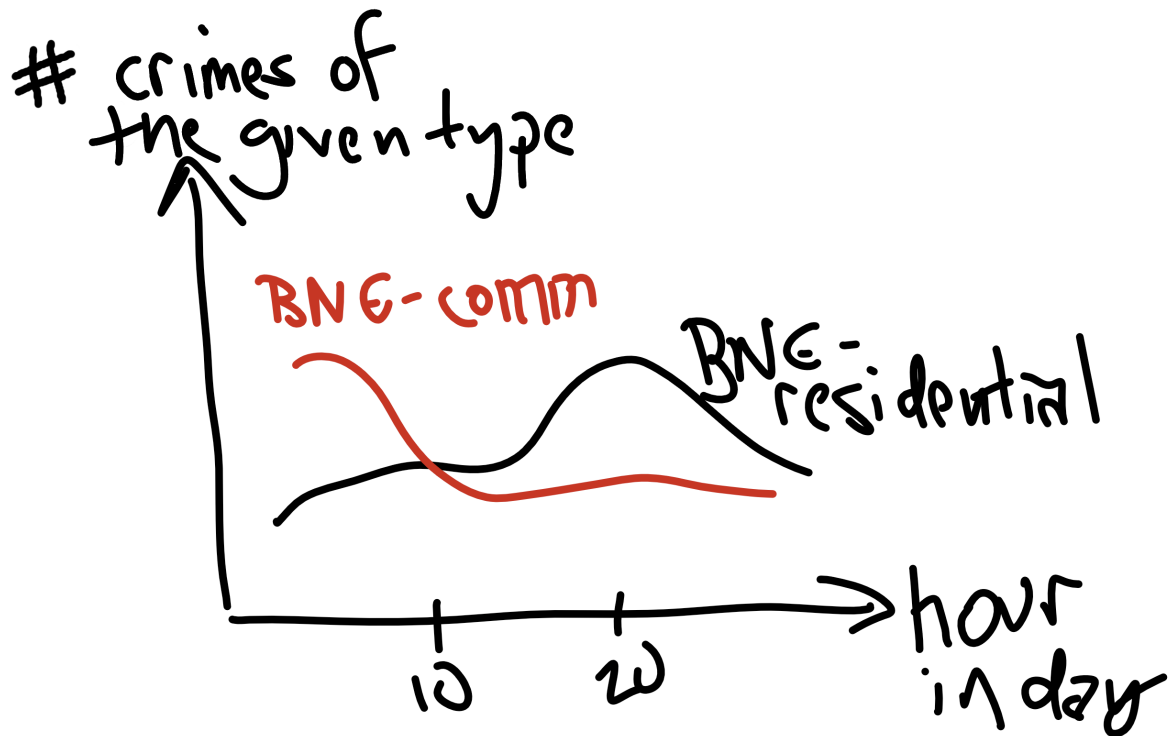
OR

- 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 [ ]: 1 # load and run data definitions for
        2 # * CrimeType
        3 # * CrimeData
        4 # * List[CrimeData]
        5 # * List[str]
        6 # * List[int]
        7
        8 %run -i module07b_day1_step2a.py
        9
       10
```

## 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 [ ]: 1 # load and run functions
        2 # * read
        3 # and helpers
        4 # * parse_crime_type
        5 # * is_reliable
        6 # * crime_type_as_str (new)
        7
        8 %run -i module07b_day1_step2b.py
        9
       10
```

## 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 [ ]: 1 # from 'testfile_empty.csv'
2 TEST_EMPTY = []
3
4 # from 'testfile_all_missing.csv'
5 TEST_ALL_MISSING = [CrimeData(CrimeType.BEC, 0),
6                     CrimeData(CrimeType.BER, 0),
7                     CrimeData(CrimeType.TB, 0),
8                     CrimeData(CrimeType.TV, 0)] # but none of these should be read
9
10 # from 'testfile_all_bec.csv'
11 TEST_ALL_BEC = [CrimeData(CrimeType.BEC, 6),
12                CrimeData(CrimeType.BEC, 18)] # missing data removed
13
14 # from 'testfile_all_ber.csv'
15 TEST_ALL_BER = [CrimeData(CrimeType.BER, 21),
16                CrimeData(CrimeType.BER, 17),
17                CrimeData(CrimeType.BER, 0)]
18
19 # from 'testfile_all_tb.csv'
20 TEST_ALL_TB = [CrimeData(CrimeType.TB, 1),
21               CrimeData(CrimeType.TB, 23),
22               CrimeData(CrimeType.TB, 17)]
23
24 # from 'testfile_all_tv.csv'
25 TEST_ALL_TV = [CrimeData(CrimeType.TV, 23),
26               CrimeData(CrimeType.TV, 14),
27               CrimeData(CrimeType.TV, 21)]
28
29 # from 'testfile_all_types.csv'
30 TEST_ALL_TYPES = [CrimeData(CrimeType.BEC, 1),
31                  CrimeData(CrimeType.BER, 2),
32                  CrimeData(CrimeType.TB, 3),
33                  CrimeData(CrimeType.TV, 4)]
34
35 # from 'testfile_all_bec_hour_6.csv'
36 TEST_ALL_BEC_HOUR_6 = [CrimeData(CrimeType.BEC, 6),
37                        CrimeData(CrimeType.BEC, 6)] # missing data removed
38
39 # from 'testfile_all_ber_hour_0.csv'
40 TEST_ALL_BER_HOUR_0 = [CrimeData(CrimeType.BER, 0),
41                        CrimeData(CrimeType.BER, 0),
42                        CrimeData(CrimeType.BER, 0)]
43
44

```

**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 [ ]: 1 @typecheck
2 def filter_for_crime_type(locd: List[CrimeData], ct: CrimeType) -> List[CrimeData]:
3     """
4     Returns only items in locd that have crime type ct.
5     """
6     # return [] # stub
7
8     # template from List[CrimeData]
9
10    # description of the accumulator
11    matches = [] # type: List[CrimeData]
12
13    for cd in locd:
14        if is_crime_type(cd, ct):
15            matches.append(cd)
16
17    return matches
18
19
20 @typecheck
21 def is_crime_type(cd: CrimeData, ct: CrimeType) -> bool:
22     """
23     Returns True if cd has crime type `ct`, otherwise returns False.
24     """
25     # return False # stub
26
27     # template from CrimeData with additional parameter ct
28     return cd.type == ct
29
30
31 # Examples and tests for is_crime_type
32 start_testing()
33
34 # Test 1: does match
35 # Test 2: doesn't match
36 expect(is_crime_type(CrimeData(CrimeType.BEC, 0), CrimeType.BEC), True) # Test 1
37 expect(is_crime_type(CrimeData(CrimeType.BER, 1), CrimeType.BER), True) # Test 1
38 expect(is_crime_type(CrimeData(CrimeType.TB, 0), CrimeType.TV), False) # Test 2
39 expect(is_crime_type(CrimeData(CrimeType.TB, 2), CrimeType.TV), False) # Test 2
40
41 summary()
42
43
44 # Examples and tests for filter_for_crime_type
45 start_testing()
46
47 # Test 1: empty list
48 # Test 2: crime type doesn't match any
49 # Test 3: crime type matches some
50 expect(filter_for_crime_type([], CrimeType.BEC), []) # Test 1
51 expect(filter_for_crime_type(TEST_ALL_BEC, CrimeType.TV), []) # Test 2
52 expect(filter_for_crime_type(TEST_ALL_TB, CrimeType.BER), []) # Test 2
53 expect(filter_for_crime_type(TEST_ALL_BEC+TEST_ALL_BER, CrimeType.BEC), TEST_ALL_BEC) #
54 expect(filter_for_crime_type(TEST_ALL_BEC+TEST_ALL_BER, CrimeType.BER), TEST_ALL_BER) #
55
56 summary()
57
58

```

**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 [ ]: 1 @typecheck
2 def hours_in_a_day() -> List[int]:
3     """
4     Returns a list of the hours in a day: [0,23].
5     """
6     # return [] # stub
7
8     # no template
9     hours = []
10
11     for h in range(24): # range is like a list, but subtly different
12         hours.append(h)
13     # or just hours = list(range(24))
14
15     return hours
16
17
18 # Examples and tests for hours_in_a_day
19 start_testing()
20
21 expect(hours_in_a_day(), [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
22
23 summary()
24
25
```

**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 [ ]:

```
1  # BOOKMARK: On Day 2 we'll redesign our old `find_hour_with_most_crimes`
2  # function into a new `count_crimes_by_hour` which will serve
3  # as a helper for `analyze` to generate our plot data.
4
5  @typecheck
6  def find_hour_with_most_crimes(hours: List[int], locd: List[CrimeData]) -> int:
7      """
8      Returns hour in `hours` for which locd has the most occurrences.
9
10     In case of a tie, returns earliest hour.
11
12     Assumes `hours` is not empty.
13     """
14     # return -1 # stub
15
16     # template from List[int] with extra parameter locd
17
18     # maximum number of crimes by hour in list so far
19     max_crimes = 0 # type: int
20
21     # hour of maximum crimes in list so far
22     hour_of_max_crimes = hours[0] # type: int
23
24     for h in hours:
25         crimes_in_hour = count_crimes_in_hour(locd, h)
26         if crimes_in_hour > max_crimes:
27             max_crimes = crimes_in_hour
28             hour_of_max_crimes = h
29
30     return hour_of_max_crimes
31
32
33  @typecheck
34  def count_crimes_in_hour(locd: List[CrimeData], hour: int) -> int:
35      """
36      Returns the number of crimes from `locd` that occur at a particular `hour`.
37      """
38     # return -1 # stub
39
40     # template from List[CrimeData] with extra parameter hour
41
42     # count of crimes in given hour in list so far
43     count = 0 # type: int
44     for cd in locd:
45         if is_crime_in_hour(cd, hour):
46             count = count + 1
47     return count
48
49
50  @typecheck
51  def is_crime_in_hour(cd: CrimeData, hour: int) -> bool:
52      """
53      Returns True if crime `cd` occurred during `hour`, otherwise False.
54      """
55     # return False # stub
56
57     # template from CrimeData with extra parameter hour
58     return cd.hour == hour
59
60
61  # Examples and tests for is_crime_in_hour
62  start_testing()
63
64  # Test 1: Crime is not in hour
65  # Test 2: Crime is in hour
66  expect(is_crime_in_hour(CrimeData(CrimeType.BEC, 7), 0), False) # Test 1
67  expect(is_crime_in_hour(CrimeData(CrimeType.BEC, 7), 7), True) # Test 2
68  expect(is_crime_in_hour(CrimeData(CrimeType.TV, 0), 0), True) # Test 2
69
```

```

70 summary()
71
72
73 # Examples and tests for count_crimes_in_hour
74 start_testing()
75
76 # Test 1: Empty crime data list
77 # Test 2: Not empty but no crimes in given hour
78 # Test 3: Crimes in given hour, of various types
79 expect(count_crimes_in_hour([], 1), 0) # Test 1
80 expect(count_crimes_in_hour(TEST_ALL_BER, 1), 0) # Test 2
81 expect(count_crimes_in_hour(TEST_ALL_BER, 17), 1) # Test 3
82 expect(count_crimes_in_hour(TEST_ALL_TB+TEST_ALL_TV, 23), 2) # Test 3
83 expect(count_crimes_in_hour(TEST_ALL_TB+TEST_ALL_TV, 17), 1) # Test 3
84 expect(count_crimes_in_hour(TEST_ALL_TV+TEST_ALL_TB, 23), 2) # Test 3 (crimes shuffled)
85
86 summary()
87
88
89 # Examples and tests for find_hour_with_most_crimes
90 start_testing()
91
92 # Test 1: Empty crime data list
93 # Test 2: Not empty but no crimes in given hours
94 # Test 3: Crimes in given hours, of various types
95 expect(find_hour_with_most_crimes([1], []), 1) # Test 1
96 expect(find_hour_with_most_crimes([5, 6, 7, 8], TEST_ALL_TYPES), 5) # Test 2
97 expect(find_hour_with_most_crimes([8, 7, 6, 5], TEST_ALL_TYPES), 8) # Test 2 (hours rev
98 expect(find_hour_with_most_crimes([3, 4, 5, 6], TEST_ALL_TYPES), 3) # Test 3
99 expect(find_hour_with_most_crimes([6, 5, 4, 3], TEST_ALL_TYPES), 4) # Test 3 (hours rev
100 expect(find_hour_with_most_crimes([1, 14, 17, 21, 23], TEST_ALL_TB+TEST_ALL_TV), 23) #
101 expect(find_hour_with_most_crimes([1, 14, 17, 21, 23], TEST_ALL_TV+TEST_ALL_TB), 23) #
102
103 summary()
104
105

```

►  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 [ ]:

```

1
2
3

```

**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 [ ]:

```
1 #####
2 # Functions
3
4 @typecheck
5 def main(filename: str,
6           crime_type: CrimeType) -> None:
7     """
8     Reads the file from given filename, analyzes the data, returns the result
9     """
10    # Template from HtDAP, based on function composition
11    return analyze(read(filename), crime_type)
12
13
14 @typecheck
15 def analyze(locd: List[CrimeData],
16            crime_type: CrimeType) -> None:
17     """
18     Plots the number of crimes of a particular
19     type for each hour of the day from the list
20     locd.
21     """
22
23     # return None # stub
24
25     # template based on composition
26     # Step 1: filter list for crime type
27     crimes_of_type = filter_for_crime_type(locd, crime_type)
28     # Step 2: build a list of hours of the day
29     hours = hours_in_a_day()
30     # Step 3: make list of crime counts for all hours in the day
31     crime_count = count_crimes_by_hour(hours, crimes_of_type)
32     # Step 4: plot crimes per hour versus hour
33     plot_crimes_by_hour(hours, crime_count)
34
35     return None
36
37 # Examples and tests for analyze
38 start_testing()
39
40 # Test 1: empty list
41 # Test 2: no matching crime type
42 # Test 3: some matching crime types
43 expect(analyze([], CrimeType.BEC), None) # Test 1
44 # We expect an empty plot like this:
45 #
46 # number
47 # of crimes
48 # |
49 # 1 |
50 # |
51 # 0 +----- hour
52 # 0 6 12 18
53 expect(analyze(TEST_ALL_BEC, CrimeType.BER), None) # Test 2
54 # We expect an empty plot like this:
55 #
56 # number
57 # of crimes
58 # |
59 # 1 |
60 # |
61 # 0 +----- hour
62 # 0 6 12 18
63 expect(analyze(TEST_ALL_BEC, CrimeType.TB), None) # Test 2
64 # We expect an empty plot like this:
65 #
66 # number
67 # of crimes
68 # |
69 # 1 |
```

```

70 # |
71 # 0 +----- hour
72 # 0 6 12 18
73 expect(analyze(TEST_ALL_BEC, CrimeType.TV), None) # Test 2
74 # We expect an empty plot like this:
75 #
76 # number
77 # of crimes
78 # |
79 # 1 |
80 # |
81 # 0 +----- hour
82 # 0 6 12 18
83 expect(analyze(TEST_ALL_BEC, CrimeType.BEC), None) # Test 3
84 # We expect a plot similar to this:
85 #
86 # number
87 # of crimes
88 # |
89 # 1 | *
90 # | |
91 # 0 +---|-----|----- hour
92 # 0 6 12 18
93 expect(analyze(TEST_ALL_BER_HOUR_0, CrimeType.BER), None) # Test 3
94 # We expect a plot similar to this:
95 #
96 # number
97 # of crimes
98 # |
99 # 3 *
100 # |\
101 # 2 ||
102 # ||
103 # 1 ||
104 # ||
105 # 0 +---|-----|----- hour
106 # 0 6 12 18
107 expect(analyze(TEST_ALL_BEC+TEST_ALL_BEC_HOUR_6, CrimeType.BEC), None) # Test 3
108 # We expect a plot similar to this:
109 #
110 # number
111 # of crimes
112 # |
113 # 3 | *
114 # | |
115 # 2 | |
116 # | |
117 # 1 | | *
118 # | | |
119 # 0 +---|-----|----- hour
120 # 0 6 12 18
121 expect(analyze(TEST_ALL_BEC_HOUR_6+TEST_ALL_BEC, CrimeType.BEC), None) # Test 3
122 # We expect a plot similar to this:
123 #
124 # number
125 # of crimes
126 # |
127 # 3 | *
128 # | |
129 # 2 | |
130 # | |
131 # 1 | | *
132 # | | |
133 # 0 +---|-----|----- hour
134 # 0 6 12 18
135
136 summary()
137
138
139 # Examples and tests for main

```

```

140 start_testing()
141
142 # Test 1: empty file
143 # Test 2: invalid data
144 # Test 3: no matching crime types
145 # Test 4: some matching crime types
146 expect(main('testfile_empty.csv', CrimeType.BEC), None) # Test 1
147 # We expect an empty plot like this:
148 #
149 # number
150 # of crimes
151 # |
152 # 1 |
153 # |
154 # 0 +----- hour
155 # 0 6 12 18
156 expect(main('testfile_all_missing.csv', CrimeType.BER), None) # Test 2
157 # We expect an empty plot like this:
158 #
159 # number
160 # of crimes
161 # |
162 # 1 |
163 # |
164 # 0 +----- hour
165 # 0 6 12 18
166 expect(main('testfile_all_bec.csv', CrimeType.BER), None) # Test 3
167 # We expect an empty plot like this:
168 #
169 # number
170 # of crimes
171 # |
172 # 1 |
173 # |
174 # 0 +----- hour
175 # 0 6 12 18
176 expect(main('testfile_all_bec.csv', CrimeType.TB), None) # Test 3
177 # We expect an empty plot like this:
178 #
179 # number
180 # of crimes
181 # |
182 # 1 |
183 # |
184 # 0 +----- hour
185 # 0 6 12 18
186 expect(main('testfile_all_bec.csv', CrimeType.TV), None) # Test 3
187 # We expect an empty plot like this:
188 #
189 # number
190 # of crimes
191 # |
192 # 1 |
193 # |
194 # 0 +----- hour
195 # 0 6 12 18
196 expect(main('testfile_all_bec.csv', CrimeType.BEC), None) # Test 4
197 # We expect a plot similar to this:
198 #
199 # number
200 # of crimes
201 # |
202 # 1 | * *
203 # | | |
204 # 0 +---|-----|----- hour
205 # 0 6 12 18
206 expect(main('testfile_all_ber_hour_0.csv', CrimeType.BER), None) # Test 4
207 # We expect a plot similar to this:
208 #
209 # number

```

```

210 # of crimes
211 # /
212 # 3 *
213 # /\
214 # 2 //
215 # //
216 # 1 //
217 # //
218 # 0 +---/-----/----- hour
219 # 0 6 12 18
220
221 summary()
222
223

```

►  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 [ ]: 1 main('crimedata_subset_bne_theft_of_bike_veh_2018.csv',
2           CrimeType.BEC) # BNE Commercial
3
4

```

### BNE Residential/Other

```

In [ ]: 1 main('crimedata_subset_bne_theft_of_bike_veh_2018.csv',
2           CrimeType.BER) # BNE Residential/Other
3
4

```

### Theft of Bicycle

```

In [ ]: 1 main('crimedata_subset_bne_theft_of_bike_veh_2018.csv',
2           CrimeType.TB) # Theft of Bicycle
3
4

```

### Theft of Vehicle

```

In [ ]: 1 main('crimedata_subset_bne_theft_of_bike_veh_2018.csv',
2           CrimeType.TV) # Theft of Vehicle
3
4

```

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


► 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 [ ]: 1 # TODO: Copy `plot_crimes_by_hour` from above.
        2
        3
```

```
In [ ]: 1 # TODO: Copy `main` and `analyze` from above.
        2
        3
```

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

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

```
In [ ]: 1 main('crimedata_subset_bne_theft_of_bike_veh_2018.csv')
        2
        3
```

## 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.

## 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.

