In [ ]:
```
from cs103 import *


```

CPSC 103 - Systematic Program Design

Module 05a Day 2

Rik Blok, with thanks to Ian Mitchell and Giulia Toti

Reminders

- Wed: Module 5 Part 2: Pre-Lecture Assignment.
- Wed: Module 3 (HtDD): Tutorial Resubmission (optional).
- No tutorial due (but recommend you try to finish tutorial 5 before the midterm).
- No code review due (but you should still practice).
- No worksheet due (but you should still practice).
- **this Wed-Fri** (was Fri-Thu): Module 5 Week 2 Tutorial Attendance.
- **Next week Fri**: Midterm exam 6:30pm - 8:00pm.

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

---

iClicker check-in

Have you printed out a paper copy of the exam reference sheet and practiced some problems using it?

A. 😨 What is an "exam reference sheet"?
B. 😬 No, I've been just looking up HtDF, HtDD and Data Driven Templates online when I need them.
C. 🤔 Yes, but I'm still looking up stuff online too.
D. 😍 Yes, and I don't need to look up anything else about HtDF, HtDD or Data Driven Templates as long as I have the sheet.

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

Arbitrary-size data

What are some examples of information that would be well-represented as arbitrary-sized data?

Accumulator

Recall, an *accumulator* is used in the template for list data definitions:

- Stores useful data about our progress through the list
- No special meaning to Python, just another variable
- You can rename it from `acc` to something more meaningful

- You can create multiple accumulator variables, if necessary

---

From last time: `List[str]` Data Definition

In [ ]:
```
from typing import List

# List[str]
# interp. a list of strings
LOS0 = []
LOS1 = ["hello", "goodbye", "Beatles"]

@typecheck
# template based on arbitrary-sized
def fn_for_los(los: List[str]) -> ...:
    # description of the accumulator
    acc = ...   # type: ...

    for s in los:
        acc = ...(s, acc)

    return ...(acc)

```

Designing a data definition for a list of integers

Problem: design a data definition for a list of integers.

▸ Jump to...

Once we've learned how to design other data definitions and realize that a data definition for a particular problem needs to be a list, these tend to be fairly straightforward to complete. Let's practice one quickly this week.

Next week, we'll see how things change (for lists, optionals, and compounds!) when a data definition we create refers to another data definition we created!

✅ Exam reference sheet

We recommend you print out the Exam reference sheet and use it as you're solving problems. The same sheet will be provided with your exams.

In [ ]:
```
# TODO: Design a data definition for a list of integers

```

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

Exercise 1: Find the largest integer

Problem: Find the largest integer in a list.

▸ Jump to...

In [ ]:
```
# TODO: Follow the HtDF recipe to write a function that finds the largest integer in a l

```

▸ ⚠ Things to watch out for! (Read after completing your solution.)

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

Exercise 2: Find the even numbers

Problem: Find all the even numbers in a list.

First, let's brainstorm an outline to the solution.

Two subproblems

In solving this problem, you can expect to encounter two subproblems:

1. How do you check if a number is even?
2. How do you attach a new item to the end of a list?

Let's solve each of these subproblems now.

🛈 Tip 1: `%` (modulo) operator

From the Python Language Reference:

> The `%` (modulo) operator yields the remainder from the division of the first argument by the second.

Examples

Expression	Value	Reason	In other words
`4 % 2`	`0`	4 divided by 2 is 2 with remainder 0	`4 == 2*2 + 0`
`5 % 2`	`1`	5 divided by 2 is 2 with remainder 1	`5 == 2*2 + 1`
`6 % 2`	`0`	6 divided by 2 is 3 with remainder 0	`6 == 3*2 + 0`
`6.5 % 2`	`0.5`	6.5 divided by 2 is 3 with remainder 0.5	`6.5 == 3*2 + 0.5`

iClicker question: How to check if a number is even?

You want to determine if a number `i` is even. Which expression will return `True` if and only if `i` is even?

A. `2 % i != 0`
B. `2 % i == 0`
C. `i % 2 != 0`
D. `i % 2 == 0`

▸ 🛈 Hint (for your review after class)

Tip 2: Appending to a list

The Python Language Reference tells us how to append (add to the end) items to an existing list:

1. We can join two lists with the `+` operator, or
2. We can append a single item with the `.append` method.

Examples

```
los1 = ['one', 'two']
los2 = ['three', 'four']
item = 'three'

# join two lists with the `+` operator
los1 + los2  # evaluates to ['one', 'two', 'three', 'four']

# append a single item to `los1`
los1.append(item) # updates `los1` to be ['one', 'two', 'three']
```

Try them out for yourself in the cell below!

In [ ]:
```
los1 = ['one', 'two']
los2 = ['three', 'four']
item = 'three'

# Try them out here!

```

iClicker question: How to append an item to a list?

```
los1 = ['one', 'two']
item = 'three'
```

You want to update the list `los1` above to contain `['one', two', 'three']`. Which expression will perform the task? Select ALL that apply. [Set iClicker question type to "Multiple Answer".]

A. `los1 + item`
B. `los1 + [item]`
C. `los1 = los1 + item`
D. `los1 = los1 + [item]`
E. `los1 = los1.append(item)`

▸ Hints (for your review after class)

Back to Exercise 2: Find the even numbers

Problem: Find all the even numbers in a list.

▸ Jump to...

Let's complete Steps 1–3 (**S**tub, **E**xamples, and **T**emplate) of the HtDF recipe together. Then you'll finish the remaining steps on your own.

In [ ]:
```
# TODO: Your program goes here


```

▶ ℹ Sample solution of HtDF Steps 1–3 (For later. Don't peek if you want to learn 🙂)

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

---

iClicker question: Target a bug

The following program was designed to calculate the product of all numbers in a list of integers... but it contains a few "bugs" (errors). Tap on one of the bugs you find. (Examples are hidden and can be considered correct.) [Set iClicker question type to "Target".]

In [ ]:

```
1  @typecheck
2  def product(loi: List[int]) -> int:
3      """
4      Return the product of all integers in loi
5      """
6      # return 0 # stub
7
8      # template from List[str]
9      prod = 0 # type: int
10
11     for i in loi:
12         prod = prod * loi
13
14     return prod
15
16
17 # examples are hidden and can be considered correct
18
19
```

▶ ℹ Hint (for your review after class)

▶ ℹ Corrected program (For later. Don't peek if you want to learn 🙂)

Exercise 3: Fun with a Large-ish Information Set

The data below represents the length of episodes from a TV show, in minutes. We have examples for only one episode of a show (Friends), a full season of a show (Game of Thrones), and a whole show (The Good Place).

ℹ Things to notice

- We've chosen to name our data type `EpisodeDurations` because it's more meaningful than just `List[float] # in range [0, ...)`
- Likewise, we've given our loop variable a descriptive name, `duration`
- In Python a loop variable (e.g., `duration`) persists after the loop finishes. That's not the case for all languages

In [ ]:
```python
from typing import List

EpisodeDurations = List[float] # in range [0, ...)
# interp. the duration of episodes in minutes for some number of
# episodes of a TV Show

ED0 = []
ED_FRIENDS_S01E01 = [22.8]
ED_GAME_OF_THRONES_S01 = [61.62, 55.28, 57.23, 55.62, 54.27, 52.6,
                          57.79, 58.13, 56.27, 52.62]
ED_GOOD_PLACE = [
    26.27, 21.50, 24.90, 22.55, 26.30, 26.35, 24.23, 25.23, 24.88,
    23.78, 26.62, 21.53, 26.88, 42.68, 21.60, 23.92, 25.37, 24.65,
    23.28, 23.72, 21.60, 24.78, 22.77, 23.47, 24.33, 21.60, 21.55,
    21.60, 21.60, 21.60, 21.53, 21.55, 21.53, 21.53, 22.53, 21.53,
    21.53, 21.53, 22.42, 21.40, 21.42, 21.43, 21.43, 21.42, 21.42,
    21.40, 21.42, 21.42, 21.45, 21.43, 52.48
]

# template based on arbitrary-sized
@typecheck
def fn_for_episode_durations(ed: EpisodeDurations) -> ...:
    # description of the accumulator
    acc = ...   # type: ...

    for duration in ed:
        acc = ...(duration, acc)

    return ...(acc)


```

Now, let's design a function that finds the average duration (in minutes) of all episodes in an `EpisodeDurations` list.

We're going to end up using multiple accumulators in this design. Template functions in data definitions give you code that *may* be useful to you, but you might decide not to use some pieces, to add or duplicate pieces, or to insert code that isn't suggested by the template at all.

For the list template, the accumulator is a suggestion: you may not need one, you may need one, or in cases where you're tracking multiple different evolving pieces of information through the loop, you may need multiple!

In [ ]:
```python
# We've completed signature, purpose, stub, and examples.
# Let's continue from there!

@typecheck
def avg_episode_duration(ed: EpisodeDurations) -> float:
    """
    Return the average duration (in minutes) of the episodes in ed.

    (The average duration of zero episodes is returned as 0.)
    """
    return 0.0  #stub

start_testing()

expect(avg_episode_duration([]), 0.0)
expect(avg_episode_duration([1, 1, 1, 1, 1]), 1)
expect(avg_episode_duration([100.12, 12.1]), (100.12+12.1)/2)
expect(avg_episode_duration(ED_GAME_OF_THRONES_S01),
       (61.62+55.28+57.23+55.62+54.27+52.6+57.79+58.13+56.27+52.62)/10
      )
expect(avg_episode_duration(ED_FRIENDS_S01E01), 22.8)

summary()


```

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

In [ ]:
```
# Now, what we all came here for; the average episode length of The Good Place:
"The Good Place is the smartest, dumbest " + str(avg_episode_duration(ED_GOOD_PLACE)) +


```

iClicker question: Do you still have questions?

Here are some common questions that came up in this week's pre-class assignment. Which questions are you still unsure about? Select as many as you like. [Set iClicker question type to "Multiple Answer".]

A. What are scenarios that you would use arbitrary-sized data but wouldn't need an accumulator?
B. When do we need more than one accumulator?
C. Can a function take in a list as a parameter without creating a data definition?
D. What are loops? How do we use them? And when do we need them?
E. Nah, I'm pretty sure I can answer these questions. 🙂

▶ ℹ️ Hints (for your review after class)