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

CPSC 103 - Systematic Program Design

Module 03 Day 2

Ian Mitchell, with thanks to Rik Blok and Giulia Toti

Reminders

- this Wed-Fri: Module 3 Tutorial Attendance
- Mon: Module 4: Pre-Lecture Assignment
- Mon: Module 3 (HtDD): Worksheet
- Wed: Module 3 (HtDD): Code Review
- Wed: Module 3 (HtDD): Tutorial Submission
- Wed: Module 1 (Intro): Tutorial Resubmission (optional)
- next Wed-Fri: Module 4 Tutorial Attendance

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

How to Design Data recipe (HtDD)

The HtDD recipe consists of the following steps:

1. **Definition:** the line that tells Python the name of the new type, it is like a signature from the HtDF.
2. **Interpretation:** describes what the new data type represents, it is like the purpose from the HtDF.
3. **Examples:** they show how to form data of this type, usually giving special cases.
4. **Template:** this is a one-parameter function that shows how a function acting on this data should operate.

Data types

Our data definitions will be composed from data types provided by Python (e.g., `int`, `float`, `str`, `bool`).

Now:

- Simple atomic data
- Interval
- Enumeration
- Optional

Later:

- Compound data (Module 4)
- Arbitrary-sized (Module 5)

Templates

Data template

This is a one-parameter function that shows how a function operating on this data should operate.

How to use with function template

When writing function with HtDF recipe, in Step 3 "Template", use template from data definition instead of writing your own:

- Comment out the body of the stub, as usual
- Then **copy the body of template from the data definition to the function**
- If there is no data definition, just copy all parameters (as we did in HtDF for atomic non-distinct data)

References: See the How to Design Data and Data Driven Templates pages on Canvas

---

"Standing" data type solution

Problem: Design a function that takes a student's standing (SD for "standing deferred", EX for "exempted", and W for "withdrew") and determines whether the student is still working on the course where they earned that standing.

Our HtDD solution from last class is copied into the cell below. Python needs to see this data type definition before you can use it in a function design, so you should execute that cell now.

Notes:

- Executing this cell will not produce any output because the final statement in the cell is a function definition for `fn_for_standing()` , and function definitions produce no output. In fact, none of the three statements in the cell (an `import` , an assignment, and a function definition) produce output, but Jupyter cells show only the output (if there is any) of the last statement.
- If we tried to call `fn_for_standing()` we would get an error because of the incomplete components of the function (shown as `...`). But we will never call this function – it is only there as a template for other functions which we will design.

In [3]:
```python
from enum import Enum

Standing = Enum('Standing', ['SD', 'EX', 'W'])
# interp. The standing of a student in a course at UBC.
# SD means standing deferred, EX means exempted, W means withdrawn.

# examples are redundant in an enumeration.

@typecheck
# template based on one of (3 cases) and atomic distinct (3 times)
def fn_for_standing(s: Standing) -> ...:
    '''
    purpose...
    '''
    if s == Standing.SD:
        return ...
    elif s == Standing.EX:
        return ...
    elif s == Standing.W:
        return ...
```

Using the new data type with our HtDF recipe

Now we can design – using the HtDF recipe – the function that takes a standing and "determines whether the student is still working on the course where they earned that standing."

Notice that the "Template" step in the HtDF recipe changes from **writing** a template to instead **copying** a template.

In [ ]:
```
@typecheck
def still_working(s: Standing) -> ...:
    """
    ...
    """
    return 0  # INCORRECT stub


start_testing()

expect(still_working(...), ...)

summary()
```

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

Re-using our "Standing" data definition

A single data definition is generally used for many different functions in a program, so we will design a second function that uses `Standing` here.

i Ratio of data definitions to functions in CPSC 103 In CPSC 103 we often only have time to design one example function for a given data definition in a lecture, tutorial, assignment, or exam; however, you should not interpret that 1:1 ratio as representative of most real problem solutions. In most real problems, we will design multiple functions to perform multiple operations on a single given data type.

Problem: Design a function that takes a standing (as above) and returns an English explanation of what the standing means.

We already have the data definition, which guides our function design. Indeed, the designed function is very similar to the previous one. Finding where it's *different* may tell you a lot about why examples and templates are useful!

In [ ]:
```
@typecheck
def describe_standing(s: Standing) -> ...:
    """
    returns an English description of a Standing s
    """
    return 0  # INCORRECT stub


start_testing()

# We've gone ahead and filled in the test cases
# already to help move us along a bit!
# The HtDD recipe tells us we should have one
# test for every value in the Standing enumeration!

expect(describe_standing(Standing.SD),
       "Standing Deferred: awaiting completion of some outstanding requirement")
expect(describe_standing(Standing.EX),
       "Exempted: does not need to take the course even if it is normally required"
expect(describe_standing(Standing.W),
       "Withdrew: withdrew from the course after the add/drop deadline")

summary()
```

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

Well done! Now, write a call to `describe_standing` :

In [ ]:
```
# Call describe_standing
```

Exercise: Data type for bank account balance

In the cell below, we have started designing a data definition for a bank account balance. Complete the definition by adding the correct template.

In [ ]:
```
AccountValue = float
# interpr. the value held by a bank account

AV0 = 0
AV_POS = 1500.55
AV_NEG = -300.10
```

▸ i Sample solution (don't peek before trying it yourself!)

Now, let's design a function that returns True if the account value is negative. We can call it `is_overdrawn` .

In [ ]:
```
# Design function is_overdrawn here,
# using the the AccountValue data definition
```

▸ i Sample solution (don't peek before trying it yourself!)

As always, once we have a function, we can use it! Write a function call to `is_overdrawn` here:

In [ ]:
```
# Call is_overdrawn
```

---

iClicker check-in

How are you doing? Any trouble keeping up?

A. 💪 Easy-peasy... you can go faster
B. 👍 Yup, I got this
C. 😕 I might have missed a bit here or there
D. 😟 Hmm, something's not working right
E. 😵 I have no idea what's going on

Exercise: Robotic Wheelchair Problem

Problem: A robotic wheelchair has a sensor that warns if it gets too close to an object. A reading from the sensor is either a distance in centimeters (that is, zero or greater) or an error code indicating that no data is presently available. Design a function to determine if a wheelchair is definitely safely out of range of any object (at least 50cm).

As before, we need to design a data definition before we can design the function!

---

iClicker question: Data definition

How do we represent the reading from the wheelchair sensor in a way that is understandable and meaningful in Python?

A. Simple atomic
B. Interval
C. Enumeration
D. Optional
E. Something else

▶ Next question

Data definition

```
In [ ]: Reading = ... # TODO!
```

▶ i Sample solution (don't peek before trying it yourself!)

Designing the function

Now on to designing the function. Here's the problem statement again:

Problem: Design a function to determine if a wheelchair is definitely safely out of range of any object (at least 50cm).

Spend a few minutes designing this function in the code cell below, with the HtDF recipe. Keep me up-to-date on your progress with the clicker question below.

iClicker question: Your HtDF progress?

Update your progress here as you complete each step of the HtDF recipe.

A. **S**tub done
B. **E**xamples done
C. **T**emplate done
D. **I**mplementation (coding) done
E. **T**esting done... my function is complete 👍

In [ ]:
```python
@typecheck
def is_safe(r: Reading) -> ...:
    """
    Returns True if a wheelchair with a sensor reading of r is known to be
    safely out of range of any object (at least 50cm away), otherwise False.
    """


start_testing()
#expect(is_safe(...), ...)
summary()
```

▸ i Sample solution (don't peek before trying it yourself!)

iClicker question: Code review 1

Is the code to the right a good solution to the problem? (Let's assume the purpose is complete.)

A. ✓ Yes
B. ✗ No

```python @typecheck def is_safe(r: Reading) -> bool: """ Returns ... """ # return True # stub # Template from Reading if r == None: return False else: if r < 50: return False elif r > 50: return True ```

▸ i Hint (for your review after class)

In [ ]:

iClicker question: Code review 2

Is the code to the right a good solution to the problem? (Let's assume the purpose is complete, sufficient examples are provided, and it passes all tests.)

A. ✓ Yes
B. ✗ No

```python @typecheck def is_safe(r: Reading) -> bool: """ Returns ... """ # return True # stub # Template from Reading if r == None: return False elif r < 50: return False else: return True ```

▸ i Hint (for your review after class)

Exercise: Age group problem

Problem: Given a person's age, write a function that identifies the age group the person belongs to. Here are the age groups we will use:

Age range (in years)	Age group
0 thru 2	Infant
3 thru 12	Child
13 thru 17	Teen
18 thru 64	Adult
65+	Senior

---

iClicker question: Data definitions

How many data definitions should we write for this problem?

A. Zero
B. 1
C. 2
D. 3
E. 4

▸ i Hint (for your review after class)

</div>

In [ ]:
```
# Data definition
```

▸ i Sample solution (don't peek before trying it yourself!)

Design our function

Now we're ready to design our function using our HtDF recipe. Recall the problem.

Problem: Given a person's age, write a function that identifies the age group the person belongs to.

iClicker question: HtDF template

In step 3 of our HtDF recipe we'll copy a template from a data definition. Which definition should we copy from?

A. `Age`
B. `AgeGroup`
C. Either, whichever we prefer
D. We should merge both templates

▶ i Hint (for your review after class)

</div>

In [ ]:
```
# Design your function here
```

▶ i Sample solution (don't peek before trying it yourself!)