

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

CPSC 103 - Systematic Program Design

Module 06 Day 2

Rik Blok, with thanks to Giulia Toti

Reminders

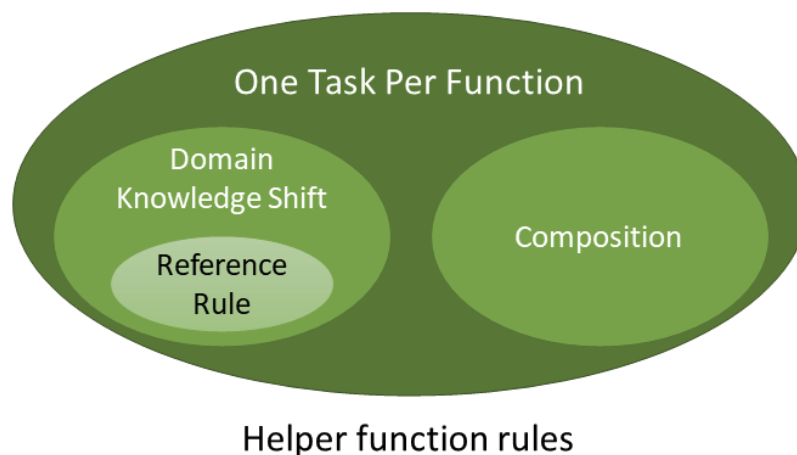
- this Wed-Fri: Module 6 Tutorial Attendance
- Wed: Module 7: Pre-Lecture Assignment
- Fri: Module 6 (One Task per Function): Worksheet
- Fri: Module 6 (One Task per Function): Code Review
- Fri: Module 6 (One Task per Function): Tutorial
- Mon-Wed: Midterm break

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

Helper rules

Now we're ready to start designing bigger, more complex problems. We use helper functions to make our programs readable and maintainable.

Each function should have one simple task that it does and it should hand off to other functions any additional tasks.



When is a helper needed?

Reference: Use a helper function when making *references* to other non-primitive data definitions (this will be in the template).

Knowledge Domain Shift: Use a helper function if a subtask involves *knowledge* about a type that is not taken as input by this function. (Won't use this one much, other than reference rule.)

Composition: When your solution is *composed* of multiple, separate operations, use a helper function for each distinct and complete operation that must be performed on the input data.



iClicker question: Calling functions

Consider the code segment to the right, which calls a sequence of functions *using variables* to store intermediate results.

```
first_result = first_func(parameter)
second_result = second_func(first_result)
third_result = third_func(second_result)
return third_result
```

Which of the following is equivalent to the code segment?

- A. `return first_result(second_result(third_result(parameter)))`
- B. `return third_result(second_result(first_result(parameter)))`
- C. `return first_func(second_func(third_func(parameter)))`
- D. `return third_func(second_func(first_func(parameter)))`
- E. None of the above

►  Hint (for your review after class)

Calling functions

Using variables (as in the question's code segment) or calling functions *directly* (as in the correct response) are both acceptable. I personally prefer *using variables* because it's more readable and easier to debug (for example, by stepping through lines in [PythonTutor](#)).

Q: How to choose the best template?

Composition provides an alternative to data-driven templating (choosing a template based on the type of input to a function). So, how do you decide which to use?

A: Step 1. Start with data-driven templating

- Choose the "protagonist" of your function, the input data that will be the focus
- When there are multiple parameters, it will usually be the most complex parameter
- *Data-driven templating:* Pick the template for the "protagonist" data type
- Don't forget to cite "...with additional parameter(s)..." if needed in template comment and apply reference rule where needed

A: Step 2. Consider whether to switch to composition

- After choosing a template, start coding your function
- If it becomes too complex, you can choose to switch to *composition* instead of the template you chose
- In this case the template will be "based on composition"
- Split into separate steps and **USE A HELPER FUNCTION** for each step
- Use data-driven templates to design helper functions

Exercise: Oldest University

Problem: Given some universities across the world, find the oldest one in a chosen country. Assume that there's at least one university in the chosen country.

Data definitions for University and List[University]

```
In [ ]: 1 from typing import NamedTuple
2
3 University = NamedTuple('University',
4                           [('name', str),
5                             ('country', str),
6                             ('year_founded', int), # in range[0,...)
7                             ('students', int), # in range[0,...)
8                             ('local_tuition', int), # in range[0,...]
9                             ('non_local_tuition', int), # in range[0,...]
10                            ('public', bool)])
11 # interp. a university with its name, country, year founded,
12 # number of students, price of local tuition, price of non-local
13 # tuition, and if it is public or not
14
15 U_UBC = University('UBC', 'Canada', 1908, 66266, 400, 5050, True)
16 U_UNICAMP = University('UNICAMP', 'Brazil', 1962, 34616, 0, 0, True)
17 U_PUCSP = University('PUCSP', 'Brazil', 1908, 34616, 2000, 2000, False)
18 U_YALE = University('Yale', 'USA', 1718, 13609, 10000, 20000, False)
19 U_HARVARD = University('Harvard', 'USA', 1636, 20970, 5000, 10000, False)
20 U_SFU = University('SFU', 'Canada', 1965, 34990, 400, 5000, True)
21 U_WATERLOO = University('Uwaterloo', 'Canada', 1959, 41000, 12500, 3000, True)
22 U_SYD = University('USYD', 'Australia', 1850, 63602, 500, 3500, True)
23
24 # template based on Compound
25 @typecheck
26 def fn_for_university(u: University) -> ...:
27     return ... (u.name,
28                 u.country,
29                 u.year_founded,
30                 u.students,
31                 u.local_tuition,
32                 u.non_local_tuition,
33                 u.public)
34
35
```

```
In [ ]: 1 from typing import List
2
3 # List[University]
4 # interp. a list of universities
5 LOU0 = []
6 LOU1 = [U_UBC]
7 LOU2 = [U_UBC, U_UNICAMP]
8 LOU3 = [U_UBC, U_UNICAMP, U_PUCSP, U_YALE]
9 LOU4 = [U_UBC, U_UNICAMP, U_PUCSP, U_YALE, U_HARVARD, U_SFU, U_WATERLOO, U_SYD]
10
11 @typecheck
12 # template based on arbitrary-sized and reference rule
13 def fn_for_lou(lou: List[University]) -> ...:
14     # description of the accumulator
15     acc = ... # type: ...
16
17     for u in lou:
18         acc = ... (fn_for_university(u), acc)
19
20     return ... (acc)
21
22
```

iClicker question: Data-driven template



Problem: Given some universities across the world, find the oldest one in a chosen country.

Given the problem statement, what parameters will our main function have? Select ALL that apply. [Set question type to "Multiple Answer".]

- A. A list of countries
- B. A list of universities in a chosen country
- C. A country
- D. A list of universities
- E. A university

►  Follow-up question

Exercise: Oldest University

► Jump to...


Problem: Given some universities across the world, find the oldest one in a chosen country. Assume that there's at least one university in the chosen country.

Let's begin our top-down design with the [main function](#). We'll come back to these helpers later.

Space reserved for low-level helper 1

In []:


```
1  # TODO: Fill in this helper function's skeleton after designing
2  # the main function below
3
4  @typecheck
5  def ____
6      """
7
8      """
9      # return True # stub
10
11     # Template from University
12     return ...(u.name,
13                u.country,
14                u.year_founded,
15                u.students,
16                u.local_tuition,
17                u.non_local_tuition,
18                u.public)
19
20
21 start_testing()
22
23 expect(__(U_UBC, "Canada"), ...)
24 expect(__(U_UBC, "Brazil"), ...)
25
26 summary()
27
28
```

►  Solution (for your review after class)

Space reserved for low-level helper 2

[► Jump to...](#)

```
In [ ]: 1 # TODO: Fill in this helper function's skeleton after designing
        2 # the main function below
        3
        4 @typecheck
        5 def ____
        6     ....
        7     ....
        8     # return ... # stub
        9
       10     # Template from University
       11     return ... (u.name,
       12                u.country,
       13                u.year_founded,
       14                u.students,
       15                u.local_tuition,
       16                u.non_local_tuition,
       17                u.public)
       18
       19
       20
       21 start_testing()
       22
       23 # example values (to reduce scrolling):
       24 # U_UBC      = University('UBC',      'Canada', 1908, 66266,    400,    5050, True)
       25 # U_UNICAMP = University('UNICAMP',  'Brazil', 1962, 34616,    0,      0, True)
       26
       27 expect(__(U_UBC, U_UNICAMP), ...)
       28 expect(__(U_UNICAMP, U_UBC), ...)
       29 expect(__(U_UBC, U_UBC), ...)
       30
       31 summary()
       32
       33
```

►  Solution (for your review after class)

Space reserved for high-level helper 1

[► Jump to...](#)

```

In [ ]: 1 # TODO: Fill in this helper function's skeleton after designing
        2 # the main function below
        3
        4 @typecheck
        5 def ____
        6     """
        7
        8     """
        9     # return [] # stub
       10
       11     # Template from List[University]
       12     # description of the accumulator
       13     acc = ... # type: ...
       14
       15     for u in lou:
       16         acc = ... (fn_for_university(u), acc)
       17
       18     return ... (acc)
       19
       20
       21 start_testing()
       22
       23 # example values (to reduce scrolling):
       24 # U_UBC = University('UBC', 'Canada', 1908, 66266, 400, 5050, True)
       25 # U_UNICAMP = University('UNICAMP', 'Brazil', 1962, 34616, 0, 0, True)
       26 # U_PUCSP = University('PUCSP', 'Brazil', 1908, 34616, 2000, 2000, False)
       27 # U_YALE = University('Yale', 'USA', 1718, 13609, 10000, 20000, False)
       28 # U_HARVARD = University('Harvard', 'USA', 1636, 20970, 5000, 10000, False)
       29 # U_SFU = University('SFU', 'Canada', 1965, 34990, 400, 5000, True)
       30 # U_WATERLOO = University('UWaterloo', 'Canada', 1959, 41000, 12500, 3000, True)
       31 # U_SYD = University('USYD', 'Australia', 1850, 63602, 500, 3500, True)
       32 # LOU1 is [U_UBC]
       33 # LOU2 is [U_UBC, U_UNICAMP]
       34 # LOU3 is [U_UBC, U_UNICAMP, U_PUCSP, U_YALE]
       35 # LOU4 is [U_UBC, U_UNICAMP, U_PUCSP, U_YALE, U_HARVARD, U_SFU, U_WATERLOO, U_SYD]
       36
       37 expect(____([], 'Canada'), ...)
       38 expect(____(LOU1, 'Canada'), ...)
       39 expect(____(LOU1, 'Brazil'), ...)
       40 expect(____(LOU2, 'Brazil'), ...)
       41 expect(____(LOU3, 'Brazil'), ...)
       42 expect(____(LOU4, 'Canada'), ...)
       43 expect(____(LOU4, 'Brazil'), ...)
       44 expect(____(LOU4, 'USA'), ...)
       45 expect(____(LOU4, 'Australia'), ...)
       46
       47 summary()
       48
       49 # Note how this function is perfectly ok with handling and
       50 # returning empty lists, unlike the top function.
       51 # It is ok for the top function to have stricter assumptions,
       52 # but they should not necessarily transfer to all the helpers.
       53 # Treat the helpers as independent functions.
       54
       55

```

►  Solution (for your review after class)

Space reserved for high-level helper 2

► Jump to...

In []:

```
1 # TODO: Fill in this helper function's skeleton after designing
2 # the main function below
3
4 @typecheck
5 def ____
6     """
7
8     """
9     # return U_UBC # stub
10
11     # Template from List[University]
12     # description of the accumulator
13     acc = ... # type: ...
14
15     for u in lou:
16         acc = ... (fn_for_university(u), acc)
17
18     return ... (acc)
19
20
21 start_testing()
22
23 # example values (to reduce scrolling):
24 # U_UBC = University('UBC', 'Canada', 1908, 66266, 400, 5050, True)
25 # U_UNICAMP = University('UNICAMP', 'Brazil', 1962, 34616, 0, 0, True)
26 # U_PUCSP = University('PUCSP', 'Brazil', 1908, 34616, 2000, 2000, False)
27 # U_YALE = University('Yale', 'USA', 1718, 13609, 10000, 20000, False)
28 # U_HARVARD = University('Harvard', 'USA', 1636, 20970, 5000, 10000, False)
29 # U_SFU = University('SFU', 'Canada', 1965, 34990, 400, 5000, True)
30 # U_WATERLOO = University('UWaterloo', 'Canada', 1959, 41000, 12500, 3000, True)
31 # U_SYD = University('USYD', 'Australia', 1850, 63602, 500, 3500, True)
32 # LOU3 is [U_UBC, U_UNICAMP, U_PUCSP, U_YALE]
33
34 expect(____(LOU3), ...)
35 expect(____([U_UBC]), ...)
36 expect(____([U_HARVARD, U_SFU, U_WATERLOO, U_SYD]), ...)
37 expect(____([U_WATERLOO, U_SFU, U_HARVARD, U_SYD]), ...)
38
39 # Tip: for finding, test lists where the variable of interest
40 # is at different positions (first, last, middle).
41 # This function has a different set of assumptions from the
42 # top one (it only assumes that the list is not empty).
43
44 summary()
45
46
```

►  Solution (for your review after class)

Main function

Problem: Given some universities across the world, find the oldest one in a chosen country.


The *stub* and *examples* have already been designed. We begin with templating...

► [Jump to...](#)

```

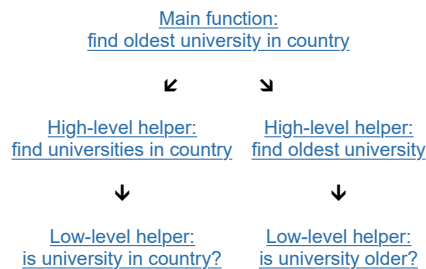
In [ ]: 1 @typecheck
2 def oldest_university_in_country(lou: List[University],
3                                 country: str) -> University:
4     """
5     Return the oldest university in a country.
6     Assume the list of universities is not empty,
7     there is no tie, and there is at least one university
8     in this country in the list.
9     """
10
11     return U_UBC # stub
12
13
14 start_testing()
15
16 expect(oldest_university_in_country([U_UBC], 'Canada'), U_UBC)
17 expect(oldest_university_in_country(LOU3, 'Brazil'), U_PUCSP)
18 expect(oldest_university_in_country(LOU3, 'Canada'), U_UBC)
19 expect(oldest_university_in_country(LOU3, 'USA'), U_YALE)
20 expect(oldest_university_in_country(LOU4, 'Australia'), U_SYD)
21 expect(oldest_university_in_country(LOU4, 'Canada'), U_UBC)
22 expect(oldest_university_in_country(LOU4, 'Brazil'), U_PUCSP)
23 expect(oldest_university_in_country(LOU4, 'USA'), U_HARVARD)
24
25 summary()
26
27

```

►  Solution (for your review after class)

Helper levels

Notice that our main function called two *high-level* helper functions. Each of those high-level helpers called their own *low-level* helper function.



You may design helper functions to any depth required by the problem.

Helper function reuse

Helper functions can often be adapted for other tasks beyond their original intended purpose.

When designing helpers, try to make them as flexible as possible to increase their usefulness and versatility.



iClicker questions: Helper function reuse

In addition to the helper functions we've already designed, let's say you have access to these additional helpers:

- `find_public_unis(lou)` - returns public universities in `lou`
- `local_tuition_at_most(lou, amt)` - returns universities with local tuition less than or equal to `amt`
- `nonlocal_tuition_at_most(lou, amt)` - returns universities with nonlocal tuition less than or equal to `amt`

Several tasks are shown below (A-E).

Given a list of universities `lou` and any other needed data, indicate which of the tasks would be performed by each of the following function bodies (direct function calls without using intermediate variables):

1. `return find_oldest_university(find_unis_in_country(lou, c1) + find_unis_in_country(lou, c2))`

► Next

- A. Find the oldest university from among two countries.
- B. Find all public universities with tuition (local or non-local) no more than a given amount.
- C. Find the oldest public university in a country.
- D. Find all public universities in a country with nonlocal tuition no more than a given amount.
- E. Something else

►  Hints (for your review after class)

☒ If you design your helper functions to be versatile, they can be reused to carry out many more tasks!