CSC148H Week 1

Ilir Dema, Michael Miljanovic

Summer 2021

Welcome!

- Welcome to CSC148
- Prerequisite: CSC108
- Goals
  - Designing programs using object-oriented programming principles
  - Writing and using linked structures (linked lists, trees)
  - Thinking recursively and writing recursive functions
  - Reasoning about efficiency of code
  - Reasoning about sorting algorithms

The Teaching Team

- The coordination is shared betwen both instructors, Michael and Ilir
- We do have 15 TAs

Finding Stuff

The course website is
`https://mcs.utm.utoronto.ca/~148`
(log-in with your utorid and password)
Start here! We'll use Quercus for a few things, but the course website is what you want to bookmark.

A Typical Week in CSC148

A typical week involves

- Doing prep reading and completing a prep exercise
- Attending our three lectures
- Working on a lab activity

Weekly Preps

- We ask you to read course material — this is where you'll see most of the course content
- Then, we ask you to complete two kinds of exercises
  - Short-answer questions (Quercus)
  - Programming exercises (MarkUs)

Lectures

- Lecture is designed to promote engagement with course content
- You will have opportunities to discuss solutions to problems and think about course content
- This is a far better use of time than just listening to me!
- You'll be working with a small group of your peers in breakout rooms throughout the term
- During the lecture portion, use chat to ask questions. In breakout rooms, you may use chat, audio, or video.

Labs

- Labs are not graded. However, they are REQUIRED work.
- We will post lab material each week that we hope you will do for practice
- Work on this material with your peers and ask the TA if you get stuck!

Evaluation

- Preps (15%)
- Two assignments (30% total, 18% the one you perform best, and 12% the other)
- Two term tests (20% total, 12% the one you perform best, and 8% the other)
- Final exam (35%)
  - 40% rule: students who earn less than 40% on the exam do not pass the course

Assignments

- The handouts will be on the course website
- Due at 22:00 on due date; submitted electronically using MarkUs
- Both assignments are completed individually

Assignments...

Late Policy

- For preps, late submissions are NOT accepted.
- For assignments, you have grace tokens—check the course information sheet for details

Academic Integrity

In brief:

- Never look at someone else's assignment work (not even a draft)
- Never show other students your assignment work (not even a draft)
- Don't copy code from any source
- Don't post anything online (e.g. `pastebin`, GitHub)
- Discuss how to solve an assignment only with the course TAs and instructor

Academic Integrity...

We often handle many academic offense cases in CSC148

- Waste of your time and ours
- Doesn't help you learn the course material
- Results in mark penalties and transcript annotations

Help!

- Instructor office hours
- TAs during labs
- Online discussion boards
- Anonymous feedback
- Form online study groups! There is a discord server.

A Sample Activity

Here's a group exercise for you.

- You've had some experience with online learning.
- Q: What're the pros and cons of online learning vs. in-person learning

Introduce yourself to your group members!

Checklist for This Week

- Bookmark the course website
- Read the course information sheet (syllabus)
- Log in to the online discussion board. Make sure your login works!
- If you plan on working on your own computer, install software listed on course website
- Work on prep 1 (not for credit) and prep 2 (for credit!)
- Drop by office hours and introduce yourself

Recap of Reading

- Object has unique ID, type, value
- Object of immutable type (e.g. integer, boolean, string) cannot change value
- Object of mutable type (e.g. list, dictionary) can change value
- A variable always stores a reference to an object
- An object has a type. A variable has no type.
- Several variables can refer to the same object

Worksheet 1

Worksheet 1
Let's practice these concepts on lists.

For Loop Gotcha

What's wrong here?

```
lst = [1, 2, 3]
for item in lst:
    item = item + 1
print(lst)
```

Function Calls

When a function is called:

- Each argument is evaluated to the id of an object. That id is assigned to the corresponding parameter
  - Argument passing is just like an assignment statement!
- Then the function body is executed

Function Calls and Parameters

```
def mess_about(n: int, s: str) -> None:
    """What does this function do?"""
    message = s * n
    s = message

if __name__ == ’__main__’:
    count = 13
    word = ’nonsense’
    mess_about(count, word)
```

(On your own) Practice more on worksheet 2!

Testing

How do we identify problems with our code?

One way

- Call your function at the Python shell
- Look at what it returns
- Judge whether your function is doing the right thing
- Repeat

What are the disadvantages of this?

Doctests

Doctests are useful for showing users what the function does.

```
def insert_after(lst: List[int], n1: int, n2: int) -> None:
    """After each occurrence of <n1> in <lst>, insert <n2>.

    >>> lst = [5, 1, 2, 1, 6]
    >>> insert_after(lst, 1, 99)
    >>> lst
    [5, 1, 99, 2, 1, 99, 6]
    """
    ...

if __name__ == ’__main__’:
    import doctest
    doctest.testmod()
```

Unit Tests

Unit tests are useful for more thorough testing of code.

- Thorough doctests would make docstrings too long
- “Unit” is usually one function
- Unit tests are typically written in a separate file
  - We can have as many tests as we like without impacting the readability of our code

Unit Tests...

- We'll use the Python pytest module to do our unit testing
- Each test case should be in a function whose name starts with `test`
- Use `assert` to state what should be true

```
def test_simple() -> None:
    input_list = [5, 1, 2, 1, 6]
    insert_after(input_list, 1, 99)
    expected = [5, 1, 99, 2, 1, 99, 6]
    assert input_list == expected
```

Pytest

- You may have used unittest before. Compared to unittest
  - You have to install pytest (doesn't come with Python)
  - Easier to write small tests (less boilerplate code)

Choosing Test Cases

- There are lots of testing frameworks out there
- We expect you to learn how to use pytest
  - Use documentation
  - Practice examples
- However, the most important (and challenging) skill is choosing (good!) test cases
- We will focus on this next

Choosing Test Cases...

Suppose we're testing a function that returns the maximum value in a list.

List	*Expected*
[3, 6, 4, 42, 9]	42
[22, 32, 59, 17, 18, 1]	59
[1, 88, 17, 59, 33, 22]	88
[1, 3, 5, 7, 9, 1, 3, 5, 7]	9
[7, 5, 3, 1, 9, 7, 5, 3, 1]	9
[561, 1024, 13, 79, 97, 4]	1024
[9, 6, 7, 11, 5]	11

Are you convinced that the function is correct?

Test Case Properties

- There are zillions of test cases. We can't test them all!
- Instead, we focus on *properties* of the test cases
- But which properties?
  - Based on what a function or method does
  - If we know how a function works, we can use that to find more properties

Worksheet 3

```
def insert_after(lst: List[int], n1: int, n2: int) -> None:
    """After each occurrence of <n1> in <lst>, insert <n2>.

    >>> lst = [5, 1, 2, 1, 6]
    >>> insert_after(lst, 1, 99)
    >>> lst
    [5, 1, 99, 2, 1, 99, 6]
    """
```

One important property is the position of `n1` in `lst` (front, middle, back)
Worksheet 3!

Property Tests: Describing Behaviour

- Generating random inputs is easy, but it's time-consuming to check the expected output
- Instead, we can describe properties of the desired function behaviour and check these properties on a huge number of random inputs
- We do this in Python using the hypothesis module

Property Tests...

Input

- In a standard test, we specify a specific input, like `[3, 6, 4, 42, 9]`
- In a hypothesis test, we specify a *property* of inputs, like “list of integers”

Output

- In a standard test, we specify the output, like 6
- In a hypothesis test, we specify a *property* of outputs, like “an integer in the list”

Thoughts on testing

- Designing test cases before writing code is a best practice in industry
- It is part of test-driven development
- When you test code, you must try to break it!

Fixing a bug

- When your testing reveals a bug, what to do?
- Beginners often:
  - Try some “typical” changes, e.g., change > to >=
  - Add `print` calls
- A rarely done but better strategy:
  - Trace the code on paper
  - Why is this better?
- A professional strategy:
  - Use the debugger to trace it for you
  - Use what you learn to hypothesize a fix

Checking your fix

- A *test suite* is a thorough set of tests
- The benefit of a test suite is that you can easily test your code again after you fix a bug!

Professionalism

- We have seen two practices that are expected of any professional
  - Test-driven development
  - Using a debugger to find and fix bugs
- You will hone these skills throughout the course
- Professionalism is a theme we will revisit