CSC 165
introduction
Danny Heap
heap@cs.toronto.edu
http://www.cdf.toronto.edu/~heap/165/W10/
what’s CSC165?

This is a course about expression (communication):

- with and through programs
- with developers
- knowing what you mean
- understanding what others mean
- analyzing arguments, programs

who to?
user
author
docstring
#
explain your code
other dev on your team
proof
my program is 2x as fast as yours.
why CSC165?

Do you:

- MEMORIZE math?
- HAVE TROUBLE explaining what you’re doing in technical work?
- HAVE TROUBLE understanding word problems?

DON’T YOU:

- ENJOY reading math books for new material
- LIKE talking about abstract \( x \) and \( y \) as much as particular examples of what \( x \) and \( y \) represent?

Shy away from
CS needs math:

- graphics → linear algebra, calculus, proj. geom, quaternions
- cryptography → number theory, field theory
- artificial intelligence → probability
- numerical analysis → multi-dim calc, linear alg
- networking
- databases → set theory (∩, ∪, −, ×)
Doing well has two aspects: being recognized as doing well by being awarded credit (grades), and being able to retain concepts and tools for use later on. Here’s how to do both:

- Read the course web page regularly. Understand the course information sheet.

- Spend enough time. We assume an average of 8 hours/week — three in lecture, two in tutorial, three reviewing or working on assignments.

- Ask questions. Make your own annotations.
When you use a natural language (English, Chinese) you can make it as precise or ambiguous as you need. For some purposes (jokes, gossip) rich ambiguity is essential. For other purposes (getting instructions on heart surgery) precision is essential. We’re all equipped to work in both modes. Work out the double meanings of these headlines:

- Prostitutes appeal to Pope
- Death may cause loneliness, feelings of isolation
- Two sisters reunite after 18 years at checkout counter
- Iraqi head seeks arms
- Police begin campaign to run down jaywalkers
We achieve precision by restricting our language. For certain jobs, in certain communities, we use some words or symbols with restricted meanings. Becoming part of the “club” involves learning the definitions of these meanings — my kids don’t mean the same thing as I do when they say something is “sick” or when they say “snap.” Some words and symbols used in special ways by mathematicians:

- continuous
- field
- group
- for all (each)
- there is (exists)
• computers are precise — they execute identical instructions identically

• humans are as precise as necessary, and different human audiences require different levels of precision

[el] especially

• The *really* difficult job is finding the right level of precision. Too much precision introduces unbearable tedium; too little introduces unfathomable ambiguity.

• Proofs are primarily works of literature: they communicate with humans, and the best proofs have suspense, pathos, humour and surprise. As a side-effect, proofs present a convincing argument for some fact.
Functions q1 through q4 each say something different about the relationship between lists L1 and L2. You may think of these lists as sets (although there’s an important difference between lists and sets)

def q1(L1, L2) :
    """Return whether ... Every element of L1 is not in L2."
    
    for x in L1 :
        if x in L2 : return False
    return True

def q2(L1, L2) :
    """Return whether ... All elements of L1 are in L2"
    
    for x in L1 :
        if not x in L2 : return False
    return True
def q3(L1, L2):
    """Return whether ...
    """
    for x in L1:
        if not x in L2: return True
    return False

def q4(L1, L2):
    """Return whether ...
    """
    for x in L1:
        if x in L2: return True
    return False

There is an element of L1 not in L2.
L1 = [1]
L2 = [2]

There is at least one element of L1 that is also in L2.
def q3(L1, L2): not q2(L1, L2)
Check your comments for q1–q4 in various ways (checking isn’t proving, but it increases our confidence or reveals flaws):

- Try out particular values for L1 and L2 to see whether the results are consistent with your comments. Check “corner” values, such as when one or both lists are empty. Try reversing the rôles of L1 and L2.

- Draw a venn diagram: interlocking circles representing L1 and L2, enclosed in a rectangle representing the “universe” from which list elements may be drawn. Try to make some of the functions q1–q4 false by having elements in some regions. Try to make some of the functions true in a similar way.

- Find lists that create patterns such as \([q1(L1,L2),q2(L1,L2),q3(L1,L2),q4(L1,L2)] = [True,True,False,False]\). Are some patterns impossible?
Functions quant1–quant4 do the same things as q1–q4, but the numbering doesn’t correspond. Use the same approach — examples, venn diagrams, etc., to verify them.

These functions use python list comprehension (don’t worry if you haven’t seen it before). Here’s an example that shows how comprehensions work:

```python
def quant1(L1, L2):
    '''Return whether ...'''
    return False in [x in L2 for x in L1]

def quant2(L1, L2):
    '''Return whether ...'''
    return True in [x in L2 for x in L1]
```

```python
>>> [x * 2 for x in [1, 2, 3]]
[2, 4, 6]
```
more quants...

def quant3(L1, L2):
    '''Return whether ...'''
    return False not in [x in L2 for x in L1]

def quant4(L1, L2):
    '''Return whether ...'''
    return True not in [x in L2 for x in L1]

    """ Solve """
You need both rigor and intuition to solve problems you haven’t seen a template for. In this course I’ll present open-ended problems, and recommend the following steps for getting started on them:

**Understand the problem**: Know what’s given, what’s required. Re-state the problem in your own words, perhaps draw some diagrams.

**Plan solution(s)**: If you’ve seen something similar, you may be able to use its result or its method. Work backwards: assume you’ve solved the problem and think about the next-to-last step. Try solving simpler, smaller versions of the problem. Have more than one plan before you attack the problem (!).

**Carry out your plan**: Does it lead somewhere? If not, repeat earlier steps. Articulate exactly why and how you’re stuck (if you are).

**Review**: Look back to savour breakthroughs and think about roadblocks. Verify your solution as much as possible. Convince a skeptical peer that you have a solution. Extend your solution to new problems…
Do the first two steps of problem-solving (understanding and planning):

A: Haven’t seen you in ages! How old are your three kids now?
B: The product of their ages (rounded down to nearest year) is 36.
A: That doesn’t really answer my question.
B: Well, the sum of their ages is — [fire engine goes by]
A: That still doesn’t tell me how old they are.
B: Well, the eldest plays piano.
A: Okay, I see: their ages are — [you have to get off the streetcar]