

mystery functions 64 ✓ 76 ✗

CSC165 winter 2013

Mathematical expression

Danny Heap

heap@cs.toronto.edu

BA4270 (behind elevators)

Course web page 416-978-5899

Using Course notes, chapter 1, 2



Outline

Introduction

Set properties

Notes



what's CSC165?

a course about expression (communication):

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



CS needs math:

- ▶ graphics
- ▶ cryptography
- ▶ artificial intelligence
- ▶ numerical analysis
- ▶ networking
- ▶ databases



doing well in CSC165

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, and emails, 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.



ambiguity

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

precision

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) \forall
- ▶ there is (exists) \exists



comment these!

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 ...  
    '''  
    for x in L1 :  
        if x in L2 : return False  
    return True
```



more comments

```
def q2(L1, L2) :  
    '''Return whether ...  
    '''  
    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
```



yet more comments

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



verify

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; see whether the results are consistent with your comments. Check “corner” values, e.g. when one or both lists are empty. Try reversing rôles of L1, 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
 $[q_1(L_1, L_2), q_2(L_1, L_2), q_3(L_1, L_2), q_4(L_1, L_2)] =$
 $[True, True, False, False]$. Are some patterns impossible?



rigor without mortis

You need both rigor and intuition to solve problems you haven't seen a template for. In this course I'll present some 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 (!).



...mortis, continued

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. . .



streetcar drama

1. What's given, required.
2. Come up with 1 (or more) plans.

Do the first two recommended steps of problem-solving for the following puzzle:

A: Haven't seen you in a long time! 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]

info. → focus on eldest, not actually -

3 kids
product
36
eldest
plays
piano

sum,
product,
aren't
enough



variation?

Functions quant1–quant4 (below) 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:

```
>>> [x * 2 (for) x in [1, 2, 3]]  
[2, 4, 6]
```

some quant...

```
def quant1(L1, L2) :  
    '''Return whether ... Some element of L1 is  
    ''' not in L2.  
    return False in [x in L2 for x in L1]
```

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



more quant...

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

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



quantifiers

Quantifiers connect properties of elements to properties of sets. The statement “Every employee earns less than 70,000” isn’t about one or even several employees — it’s about an entire set of employees. How do you check whether it’s true? What if 70,000 is replaced by 40,000?

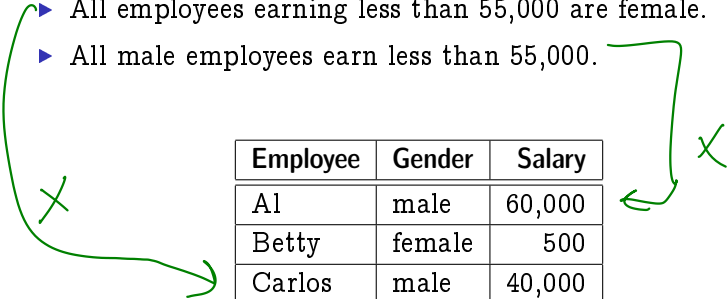
Employee	Gender	Salary
Al	male	60,000
Betty	female	500
Carlos	male	40,000
Doug	male	30,000
Ellen	female	50,000
Flo	female	20,000



universal claims: \forall

How do you verify/disprove:

- ▶ All female employees earn less than 55,000.
- ▶ All employees earning less than 55,000 are female.
- ▶ All male employees earn less than 55,000.



Employee	Gender	Salary
Al	male	60,000
Betty	female	500
Carlos	male	40,000
Doug	male	30,000
Ellen	female	50,000
Flo	female	20,000



existential claims: \exists

Another sort of claim “Some employee earns less than 15,000” appear to be about some individual, un-named, employee. Other phrasings “There exists an employee who earns less than 15,000” or “At least one employee earns less than 15,000” also seem to be about the property of some individual employee. However the small modification “Some employee earns more than 15,000” makes it clear that we are talking about the non-emptiness of {Al, Carlos, Doug, Ellen, Flo}.

Employee	Gender	Salary
Al	male	60,000
Betty	female	500
Carlos	male	40,000
Doug	male	30,000
Ellen	female	50,000
Flo	female	20,000



Notes