CSC165 Fall 2014

Mathematical Expression and Reasoning for Computer Science

Section L5101

Larry Zhang

Offices:

➔ BA 5206 (most of the time)
➔ BA 4262 (office hours)

Email: ylzhang@cs.toronto.edu

The teaching team

Instructors:

Danny Heap & Larry Zhang

TAs:

Christina Chung, Yana Davis, Christine Angela Ebeo, Siamak Freydoonnejad, Lin Gao, Gal Gross,

Jason Harrison, Madina Ibrayeva, Nadira Izbassarova,

Natalie Morcos, Shahrzad Pouryayevali,

Adam Robinson-Yu, Ekaterina Shteyn, Elias Tragas,

Eleni Triantafillou, and Yiyan Zhu

Today's agenda

- ➔ Introduction to CSC165
- ➔ Precision and ambiguity
- ➔ Problem solving; Quantifiers

Lecture 1.1: Introduction

Why CSC165

What's in it

How to take it (successfully)

Mathematical Expression and Reasoning for Computer Science

Math

A math problem...

John is twenty years younger than Amy, and in five years' time he will be half her age.

What is John's age now?

Another math problem...

*Let **A**, **B**, and **C** be three statements.*

*The statement "**A** being true implying **B** being true implies **C** being true" is true if and only if either **A** is true and **B** is false or **C** is true.*

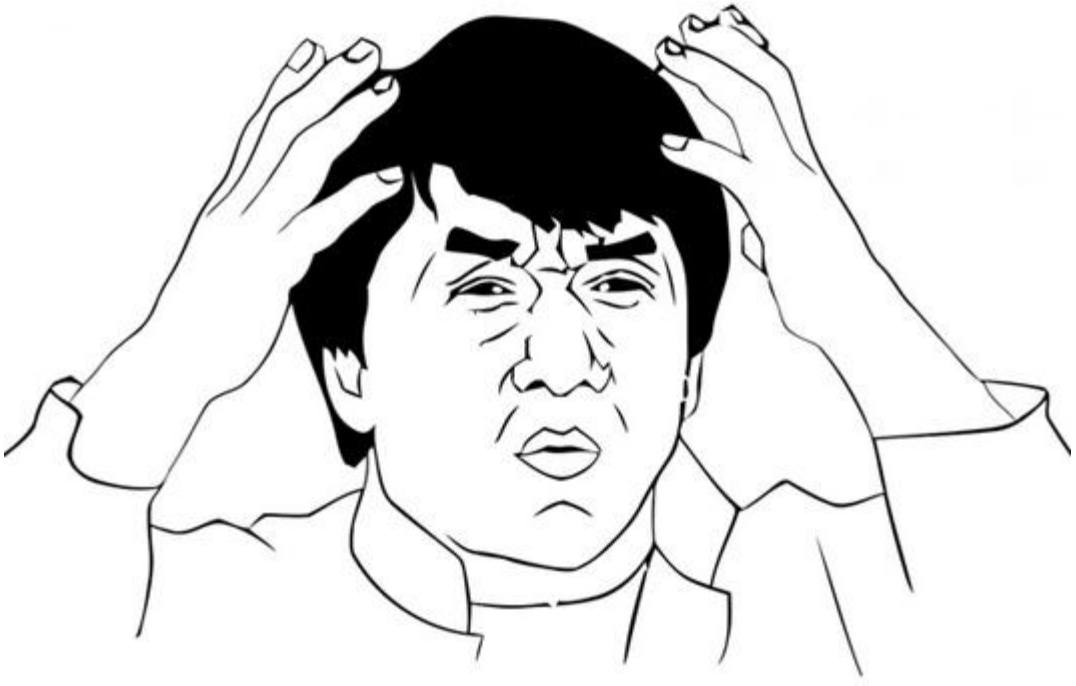
Is it true?

“Another” math problem...

Prove that

$$(A \Rightarrow B) \Rightarrow C \;\Leftrightarrow\; (A \wedge \neg B) \vee C$$

If you are feeling...



You are in the right class!

Mathematical Expression and Reasoning for Computer Science

Expression Reasoning

Communication

Communication

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

Inefficient communication

***You**: Hi Robot. I want to buy a phone. It should be an Android phone with 4-inch screen, and less than $300…., but if it is not Android it is fine if it is more than $300…., if it is more than $300, it is fine only if it is not Android and not with 4-inch screen…., but if it is with 4-inch screen, it is OK if it is Android or it is more than $300, but not OK if is both Android and more than $300….*

Dear Robot, could you help me find all such phones?

***Robot:** …*

Efficient communication

***You**: Hi Robot.*

*Let **A** be the set of Android phones*
*Let **B** be the set of phones with 4-inch screens*
*Let **C** be the set of phones that cost more than $300.*

Please find $(A \cap B) \triangle C$

***Robot:** Here they are.*

Mathematical Expression and Reasoning for Computer Science

Computer Science

CS needs math

- Graphics — Linear algebra, geometry
- Cryptography — Number theory
- Artificial intelligence — Probability
- Numerical analysis — Calculus
- Networking — Stochastic processes
- Database — Set theory

CSC165 is the foundation of all these maths

Why CSC165?

You need to learn the math in order to do anything interesting in computer science.

What topics are in CSC165

- Logic and expression
  - **the language of math**
- Proof techniques
  - **how to prove things, like a pro**
- Complexity, program running time
  - **how to measure the "speed" of a program**
- Halting problems and computability
  - **what computers cannot do**

How to do well CSC165?

First of all …

Be interested

Course web page

http://www.cdf.toronto.edu/~heap/165/F14/

Larry's slides:

http://www.cs.toronto.edu/~ylzhang/csc165f14/

Course Notes / Info Sheet

- **Course notes**: *de facto* textbook, available at the course web page.

- **Course info sheet**: contract between the students and the teachers, available at the course web page.

Announcements

Will be sent to your **utoronto.ca** email address.

Check email regularly (daily)

Lectures

Tuesdays 6:10~9:00pm, BA1130

Slides updated weekly on course web page

- Pre-lecture slides (print-friendly) available on Mondays
- Lecture slides (full content) available after lecture.

A tip for lectures

Get involved in classroom interactions

- asking / answering questions
- making guesses / bets / votes
- back-of-the-envelop calculations
  - Bring a pencil and some scrap paper

Emotional involvement makes the brain remember better.

Students who interact WIN

Tutorials

Thursdays 7:10~8:30pm (starting from week 2)

- Work on some exercises
- Discuss solutions with TA
- Take a quiz

T5101A: BA3116
T5101B: BA2135
T5101C: BA2159
T5101D: BA3008
T5101E: BA3012

Tutorials are as important as lectures!

Office hours

Thursdays 4pm~6pm, in BA 4262.

Office hours are **G-O-O-D**.

Statistics show that students who regularly go to office hours have significantly higher probability of getting A's, and are almost impossible to fail.

Course forum (Piazza)

http://piazza.com/utoronto.ca/fall2014/csc165h

- Use your **mail.utoronto.ca** email to sign up.
- For discussions among students.
- Instructors and TAs will also be there.
- Don’t discuss assignment solutions before due dates.

Marking scheme

- 3 assignments + “SLOG”: **32%**
- 9 quizzes: **12%**
- 2 term tests: **16%**
- 1 final exam: **40%**
- **Total: 100%**

Must get at least 40% of final exam to pass.

Marking scheme: weights

- 32% taken by 3 assignments and SLOG
  - Best piece takes 10%, next 9%, next 7% and worst piece takes 6%
- 16% taken by 2 term tests
  - the better one takes 10%
  - the worse one takes 6%

courSe LOG

- A (weekly) log of your participation
  - what's something new you learned this week?
  - what material did you enjoy this week?
  - what material challenged you this week?
  - your understanding of this week's topic.
  - ...
- In form of a Blog (*blogger.com*)
  - URLs to be submitted by the end of Week 2.
- Detailed instructions on course web page
  - http://www.cdf.toronto.edu/~heap/165/F14/SLOG/slog.pdf

Assignments

- May work in groups of up to **three** students
  - Submit one single **PDF** file on *MarkUs*
  - Submission must be **typed**, (LaTeX recommended)

- **Collaborate intelligently!**
  - *Rule of thumb*: collaborate in such a way that **all three of you** can pass the final exam

Late work, remarking

- Late works are not accepted.
- In case of events beyond control, provide a documented reason.
- If marking is unfair, submit a remark request form (available at course web page) within a week of receiving the work back.

Term tests and final exam

Test 1 in Week 5:

Tuesday October 7, 6:10~7pm, in class

Test 2 in Week 9:

Tuesday November 4, 6:10~7pm, in class

Final exam:

Some time in December

How many hours a week?

Expectation: 8 hours / week

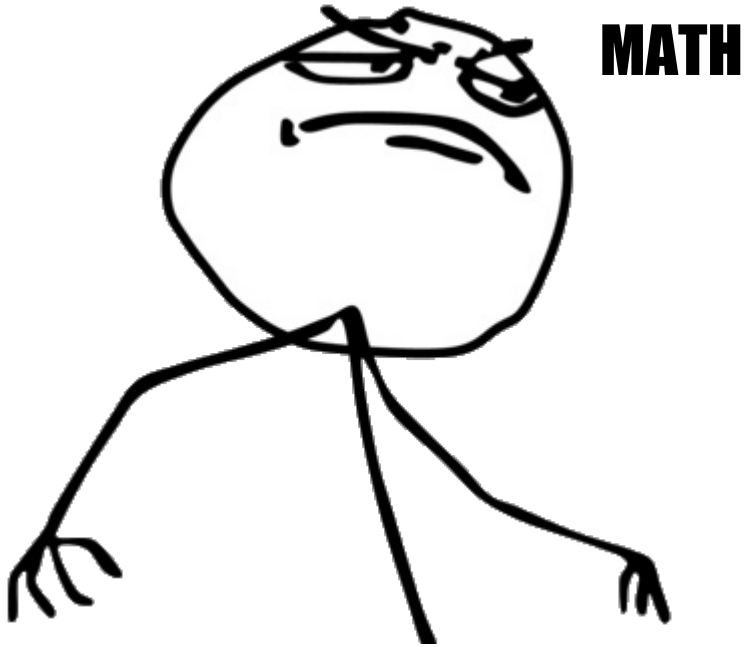
➔ 3 hours in lecture

➔ 2 hours in tutorial

➔ 3 hours in preparing, reviewing and working on assignments

Checklist: how to do well

- Be interested
- Check course web page.
- Check emails regularly
- Go to lectures, interact
- Go to tutorials, interact
- Review course notes
- Discuss on course forum
- Go to office hours
- Work on assignments, and submit on time
- Write SLOGs
- Do well in term tests and final exams



Lecture 1.2: Precision

Course notes: Chapter 1

➔ How to be precise?

➔ How precise to be?

Natural language can be ...

Which one is better?

Students who interact WIN

Natural language can be ...

Ambiguous **Precise**

- ➔ jokes
- ➔ gossip
- ➔ puzzles

- ➔ heart surgery instructions
- ➔ air traffic control
- ➔ computer algorithms

Ambiguity

- Prostitutes appeal to Pope.
- Death may cause loneliness, feeling of isolation.
- Iraqi head seeks arms.
- Police begin to campaign to run down jaywalkers.
- Two sisters reunite after 18 years at checkout counter.

Why are they ambiguous?

➔ **Words with double-meanings**

Precision

How to be precise?

➔ Restrict the meanings of words.

Being in the “club” of a profession means learning the vocabulary (words with restricted meanings) of the club.

➔ For example, for mathematicians

- continuous, open, closed
- group, ring, field
- for all, for each: $\forall$
- there is (exists): $\exists$

PATHS, TREES, AND FLOWERS

JACK EDMONDS

1. Introduction. A *graph* G for purposes here is a finite set of elements called *vertices* and a finite set of elements called *edges* such that each edge *meets* exactly two vertices, called the *end-points* of the edge. An edge is said to *join* its end-points.

A *matching* in G is a subset of its edges such that no two meet the same vertex. We describe an efficient algorithm for finding in a given graph a matching of maximum cardinality. This problem was posed and partly solved by C. Berge; see Sections 3.7 and 3.8.

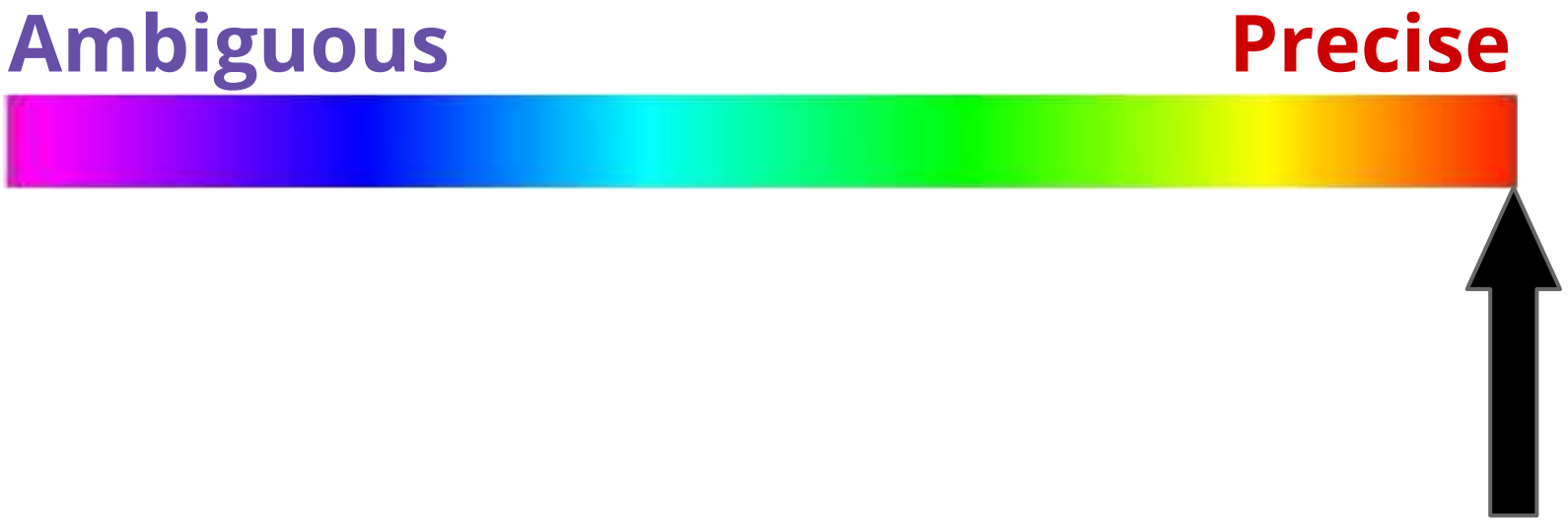
Maximum matching is an aspect of a topic, treated in books on graph theory, which has developed during the last 75 years through the work of about a dozen authors. In particular, W. T. Tutte **(8)** characterized graphs which do not contain a *perfect* matching, or *1-factor* as he calls it—that is a set of edges with exactly one member meeting each vertex. His theorem prompted attempts at finding an efficient construction for perfect matchings.

This and our two subsequent papers will be closely related to other work on the topic. Most of the known theorems follow nicely from our treatment, though for the most part they are not treated explicitly. Our treatment is independent and so no background reading is necessary.

Section 2 is a philosophical digression on the meaning of "efficient algorithm." Section 3 discusses ideas of Berge, Norman, and Rabin with a new proof of

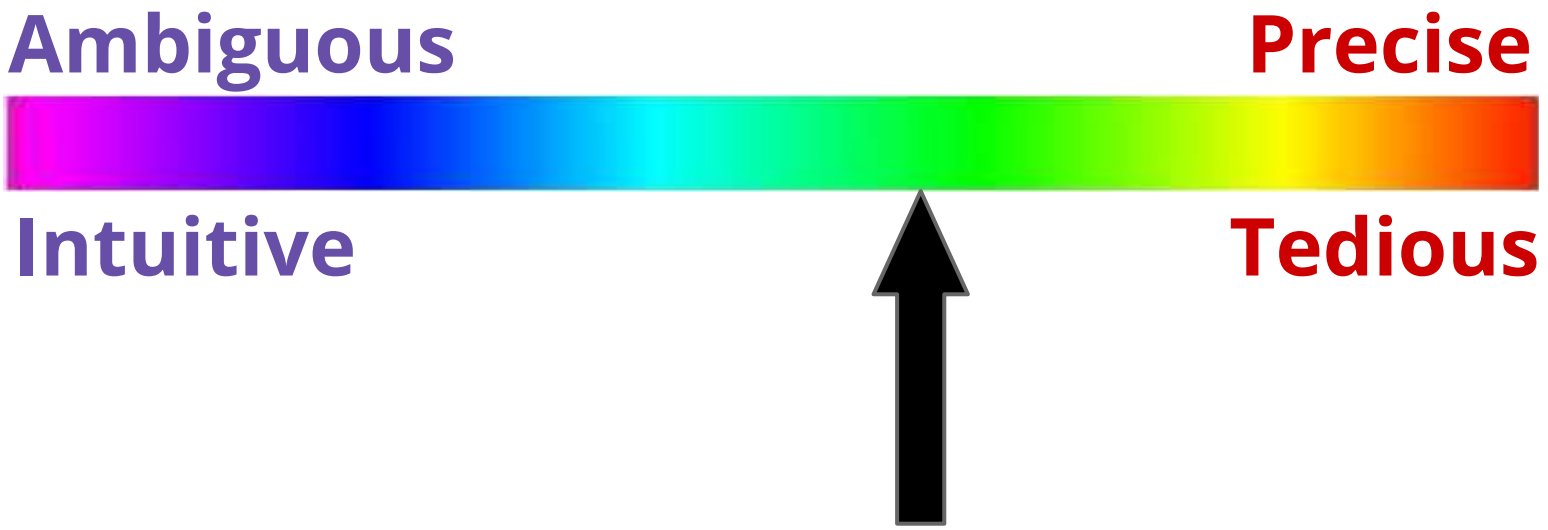
Canadian Journal of mathematics 17.3 (1965): 449-467.

How precise should computers be?



Computers need to execute identical instructions identically.

How precise should humans be?



Humans are as precise as necessary.

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 facts.

Computer language => human language

S1 and S2 are two sets

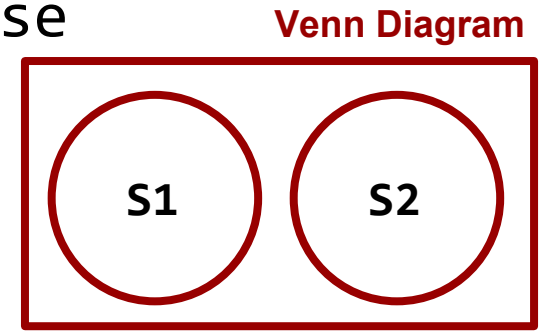
```
def q1(S1, S2):
    '''Return whether ...
    '''

    for x in S1:
        if x in S2 : return False
    return True
```

Computer language => human language

S1 and S2 are two sets

```
def q1(S1, S2):
    '''Return whether every element of S1
       is NOT an element of S2
    '''
    for x in S1:
        if x in S2 : return False
    return True
```

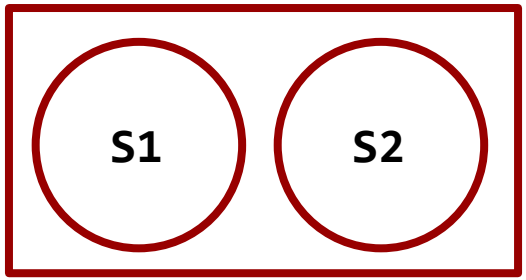


Computer language => human language

S1 and S2 are two sets

```
def q1(S1, S2):
    '''Return whether S1 and S2 have NO
       intersection
    '''
    for x in S1:
        if x in S2 : return False
    return True
```

Venn Diagram



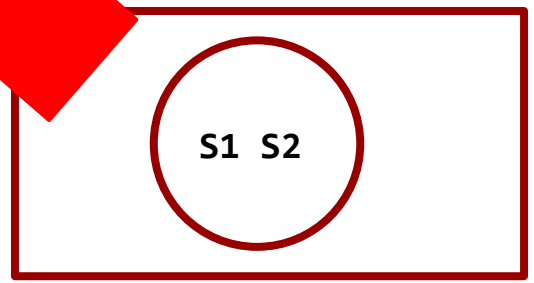
Computer language => human language

```
def q2(S1, S2):
    '''Return whether ...
    '''

    for x in S1:
        if x not in S2 : return False
    return True
```

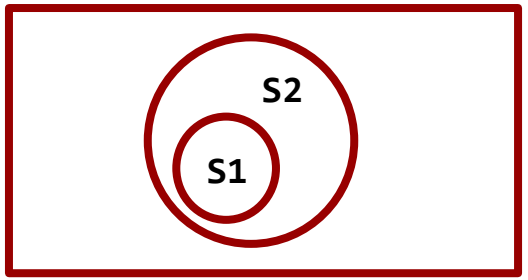
Computer language => human language

```
def q2(S1, S2
    '''Return
       the same
    '''
    for x in S1:
        if x not
    return Tr
```



Computer language => human language

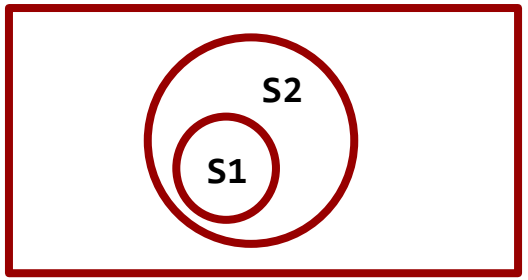
```
def q2(S1, S2):
    '''Return whether all elements in S1
       are in S2
    '''
    for x in S1:
        if x not in S2 : return False
    return True
```



Computer language => human language

```
def q2(S1, S2):
    '''Return whether S1 is a subset of S2
    '''

    for x in S1:
        if x not in S2 : return False
    return True
```



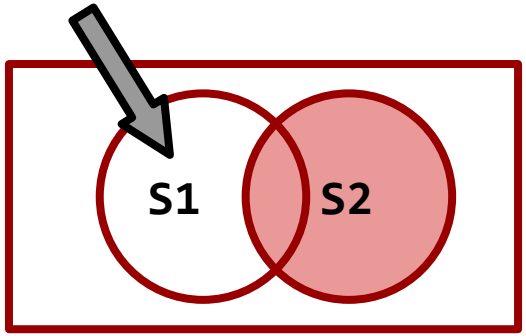
Computer language => human language

```
def q3(S1, S2):
    '''Return whether ...
    '''

    for x in S1:
        if x not in S2 : return True
    return False
```

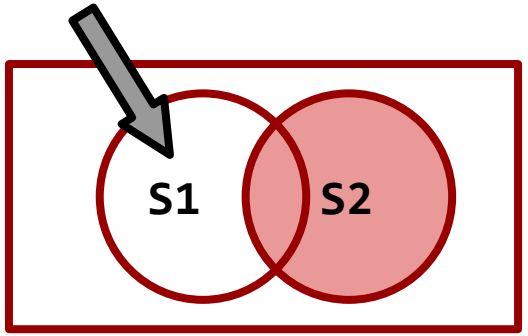
Computer language => human language

```
def q3(S1, S2):
    '''Return whether there exists an element
       in S1 which is not in S2
    '''
    for x in S1:
        if x not in S2 : return True
    return False
```



Computer language => human language

```
def q3(S1, S2):
    '''Return whether S1 is NOT a subset of S2
    '''
    for x in S1:
        if x not in S2 : return True
    return False
```



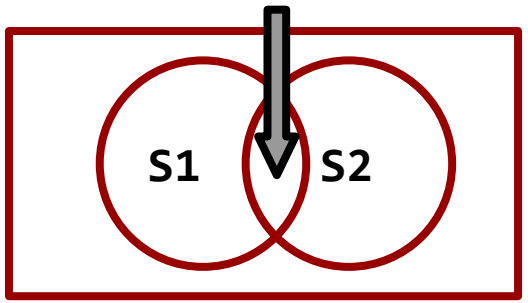
Computer language => human language

```
def q4(S1, S2):
    '''Return whether ...
    '''

    for x in S1:
        if x in S2 : return True
    return False
```

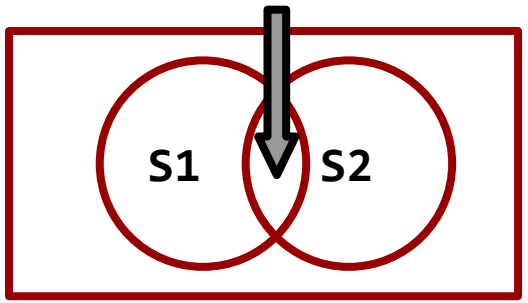
Computer language => human language

```
def q4(S1, S2):
    '''Return whether there exists an element
       in S1 which is in S2
    '''
    for x in S1:
        if x in S2 : return True
    return False
```



Computer language => human language

```
def q4(S1, S2):
    '''Return whether S1 and S2 have
       intersection
    '''
    for x in S1:
        if x in S2 : return True
    return False
```



Challenge question

Find S1 and S2 such that

- ➔ q1(S1,S2) is True, and
- ➔ q2(S1,S2) is True, and
- ➔ q3(S1,S2) is False, and
- ➔ q4(S1,S2) is False

```
def q1(S1, S2):
  for x in S1:
    if x in S2:
      return False
return True
```

```
def q2(S1, S2):
  for x in S1:
    if x not in S2:
      return False
return True
```

```
def q3(S1, S2):
  for x in S1:
    if x not in S2:
      return True
return False
```

```
def q4(S1, S2):
  for x in S1:
    if x in S2:
      return True
return False
```

Challenge question

Find S1 and S2 such that

- ➔ q1(S1,S2) is True, and
- ➔ q2(S1,S2) is True, and
- ➔ q3(S1,S2) is False, and
- ➔ q4(S1,S2) is False

q1: S1 and S2 have NO intersection

q2: S1 is a subset of S2

q3: S1 is NOT a subset of S2

q4: S1 and S2 have intersection

S1=[], S2=[1]

Summary

- ➔ Computer codes are **super precise**.
- ➔ Our human language descriptions are **precise enough**, and also more **intuitive** and helpful for solving complex problems.
- ➔ This is a right level of precision.

Python syntactic sugar

```
S = {1, 3, 5, 7, 9}

{x for x in S if x > 6}
➔ {7, 9}

{x + 10 for x in S if x > 6}
➔ {17, 19}
```

Python syntactic sugar (cont.)

`all(L)`: whether all elements in L are True.

- ➔ `all([True, True, True])`
  - ◆ **TRUE**
- ➔ `all([True, False, True])`
  - ◆ **FALSE**

`any(L)`: whether any element in L is True

- ➔ `any([False, False, False])`
  - ◆ **FALSE**
- ➔ `any([False, False, True])`
  - ◆ **TRUE**

Cooler code for q1 ~ q4

```
def q1(S1, S2):
    '''Return whether every element of S1
       is NOT an element of S2
    '''

    return all([x not in S2 for x in S1])
```

Cooler code for q1 ~ q4

```
def q2(S1, S2):
    '''Return whether all elements in S1
       are in S2
    '''

    return all([x in S2 for x in S1])
```

Cooler code for q1 ~ q4

```
def q3(S1, S2):
    '''Return whether there exists an element
       in S1 which is not in S2
    '''

    return any([x not in S2 for x in S1])
```

Cooler code for q1 ~ q4

```
def q4(S1, S2):
    '''Return whether there exists an element
       in S1 which is in S2
    '''

    return any([x in S2 for x in S1])
```

Lecture 1.3: Problem Solving and Quantifiers

Course notes: Chapter 1,2

Problem solving

The usual patterns

➔ Avoid working on a problem.

➔ Diving in without a plan.

How to solve it (by George Polya)

1. Understand the problem
   - ➔ what's given, what's required?
2. Plan solution(s)
   - ➔ try more than one plans
3. Carry out your plan
   - ➔ does it lead to somewhere?
4. Review your solution
   - ➔ convince a skeptical peer

Streetcar drama

You are on a streetcar overhearing two persons' conversation about some kids' ages.

You want to know the kids' ages…

A: Haven’t seen you in ages! How old are your three kids now?

B: The product of their ages (integers in years) is 36.

A: That’s doesn’t really answer my question…

B: Well, the sum of their ages is -- [*fire engine goes by*]

A: Hmm… that still doesn’t really tell me how old they are.

B: Well, the eldest one plays piano.

A: Okay, I see, so their ages are -- [*you have to get off the streetcar*]

Understand the problem

We want to find three number **x, y, z** (let **x ≥ y ≥ z**).

We know that **x·y·z = 36**.

The plan of solution

Enumerate all integer combinations **x, y, z**, such that **x·y·z = 36**, then rule out combinations that don't make sense.

~~36, 1, 1~~

~~18, 2, 1~~

12, 3, 1

9, 4, 1

9, 2, 2

6, 6, 1

6, 3, 2

4, 3, 3

There is not enough information to get a unique solution.

Would a skeptical peer be convinced?

A: Haven’t seen you in ages! How old are your three kids now?

B: The product of their ages (integers in years) is 36.

A: That’s doesn’t really answer my question…

B: Well the sum of their ages is -- [*fire engine goes by*]

A: Hmm… that still doesn’t really tell me how old they are.

B: Well, the eldest one plays piano.

A: Okay, I see, so their ages are -- [*you have to get off the streetcar*]

Really understand the problem

- ➔ The product of their ages is 36.
  - ◆ **x·y·z = 36**
- ➔ Knowing the sum of their ages does not lead to a unique solution.
  - ◆ **x+y+z** is not unique among all possible combinations
- ➔ There is an eldest one.
  - ◆ **x** is unique among **x**, **y** and **z**

The new plan of solution

- ➔ **x·y·z = 36**.
- ➔ **x+y+z** is not unique among all possible combinations
- ➔ **x** is unique among **x**, **y** and **z**

36, 1, 1
18, 2, 1
12, 3, 1
9, 4, 1
9, 2, 2
6, 6, 1
6, 3, 2
4, 3, 3

The kids' ages must be 9, 2 and 2.

How to solve it (by George Polya)

1. Understand the problem
   - ➔ what's given, what's required?
2. Plan solution(s)
   - ➔ try more than one plans
3. Carry out your plan
   - ➔ does it lead to somewhere?
4. Review your solution
   - ➔ convince a skeptical peer

Quantifiers

Definition

- A **quantifier** is an expression that indicates the scope of a term to which it is attached.
- e.g., **all** cars are red.
- e.g., **some** cars are green.
- Quantifiers connect properties of elements to properties of sets.

Universal quantification: ∀

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

- ➔ **All** female employees earn less than $55,000.
- ➔ **Every** employee earning less than $55,000 are female.
- ➔ Male employees earns less than $55,000.

How to prove / disprove each statement?

Prove / disprove universal quantification

To prove, verify that **every** element of the domain is an **example** that satisfies the quantification.

To disprove, give a **single** **counter-example** that does **not** satisfy the quantification.

Existential quantification: ∃

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

- ➔ **Some** employee earns over $80,000.
- ➔ **There exists** an male employee who earns less than $27,000.
- ➔ **At least one** female employee earns over $42,000.

How to prove / disprove each statement?

Prove / disprove existential quantification

To prove, give a **single** **example** that satisfies the quantification.

To disprove, verify that **every** element of the domain is a **counter-example** that does **not** satisfies the quantification.

Today's summary

- ➔ Introduction to CSC165
- ➔ How to achieve precision
- ➔ Balance between precision and intuition
- ➔ Some python tricks
- ➔ Methodology of problem solving
- ➔ Quantifiers: universal, existential, how to prove and disprove

Next week

- ➔ More about quantifiers
- ➔ More about proof / disproof
- ➔ Sentences and symbols

Read Chapter 1.5 of Course Notes for mathematical prerequisites.