Logic for Computer Science

01 – Intro

Wouter Swierstra

Utrecht University

Today

- Organisation
- What is logic?
- Why logic?

Organisation

Lecturers: Wouter Swierstra (until Christmas) & Paige Randall North (after Christmas)

Tutorial sessions: In person practice sessions are split over nine different groups. These are not mandatory – but I hope you find them useful.

You should have been allocated to a group in MyTimetable.

The first *werkcollege* is Thursday after the lecture.

Additional support: through MS teams chat - check the Blackboard page for the invitation code.

Quizzes: weekly quizzes to help you keep up.

Website

Al the practical information about the course can be found on the website:

`https://ics.uu.nl/docs/vakken/b1li/`

Website

Al the practical information about the course can be found on the website:

`https://ics.uu.nl/docs/vakken/b1li/`

Check the website and monitor the General Teams channel regularly!

Note: there is no (meaningful content on the) *Blackboard* page.

Website

Al the practical information about the course can be found on the website:

`https://ics.uu.nl/docs/vakken/b1li/`

Check the website and monitor the General Teams channel regularly!

Note: there is no (meaningful content on the) *Blackboard* page.

I'll add updates regularly:

- Latest news
- Slides from the lectures will be available for download from the course website
- Updates to the schedule
- Exercises for the practicals & additional exercises
- New literature and links

English vs Nederlands?

I'll be teaching the lectures in Dutch; Paige will teach in English.

- The book is in English. So are the slides.
- Not all of the supervisors teaching exercises sessions speak Dutch.
- MSc students taking this course as a deficiency or exchange students do not necessarily speak Dutch.

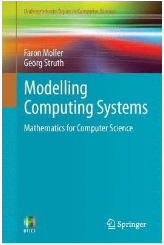
English vs Nederlands?

I’ll be teaching the lectures in Dutch; Paige will teach in English.

- The book is in English. So are the slides.
- Not all of the supervisors teaching exercises sessions speak Dutch.
- MSc students taking this course as a deficiency or exchange students do not necessarily speak Dutch.

Hopelijk leidt dit niet tot al te veel verwarring!

Book



Modelling Computing Systems: Mathematics for Computer Science; Moller and Struth

De .pdf version is available for download via the library for free.

But it may still be worth buying a paper copy, if you prefer.

We've collected a list of errata - linked from the website. Please let us know if you spot a mistake!

Lecture notes

In addition to the book, I have a short set of lecture notes available for download from the website.

We will use these for the last few lectures.

There is a github repository – please open an issue or submit a pull request if you have any suggestions for improvement!

`https://github.com/wouter-swierstra/logic-notes`

Lectures (colleges) & practicals (werkcolleges) & quiz

- 2 lectures and 2 practical sessions per week
  - Tuesday 13:15 – 15:00 college
  - Tuesday 15:15 – 17:00 werkcollege
  - Thursday 09:00 – 10:45 college
  - Thursday 11:00 – 13:00 werkcollege
- The first *werkcollege* will be Thursday morning.
- Starting next week, each Tuesday there will be a quiz to be completed online in Remindo.

Quizzes

10% of your final mark will be determined by a weekly online quiz to be completed in Remindo.

This quiz will be held each Tuesday from 16:00–16:30.

They can be completed on campus or at home.

Quizzes

10% of your final mark will be determined by a weekly online quiz to be completed in Remindo.

This quiz will be held each Tuesday from 16:00–16:30.

They can be completed on campus or at home.

The tests serve as a check – for you and me both – to measure your understanding of the material.

After each (closed) question, you will receive feedback on your answer.

A model solution for each open question will be discussed in the practical session.

I **strongly recommend** studying the relevant material practicing exercises **before** taking the test.

Quiz ground rules

- The quizzes are ‘open book’ – this creates a level playing field for those students doing the exam at home or in class.
- You are forbidden from using AI support when submitting your work. (The solutions systems like ChatGPT generate are not very good, glossing over the key proof steps.)
- I expect you do work on the test **individually** – any sharing of answers or discussion of the questions well be labelled as **fraud** and handled accordingly.
- There will not be a resit opportunity for quizzes.

Quiz ground rules

- The quizzes are ‘open book’ – this creates a level playing field for those students doing the exam at home or in class.
- You are forbidden from using AI support when submitting your work. (The solutions systems like ChatGPT generate are not very good, glossing over the key proof steps.)
- I expect you do work on the test **individually** – any sharing of answers or discussion of the questions well be labelled as **fraud** and handled accordingly.
- There will not be a resit opportunity for quizzes.

These quizzes are there to help you keep up.

If you find yourself unable to answer the questions, constantly looking through the book, or struggling to understand what the question is about – **you need to catch up now**.

Once you are behind, the lectures stop making sense — it’s all too easy to drop out entirely.

Final mark

- The average of the six out of seven best quiz scores (10%)
- Mid-term (40%)
- Final exam (50%)
- There will be a resit opportunity (herkansing) – provided your final mark is at least a 4.0.

There is no opportunity to resit quizzes. Missing one will not effect your mark.

Exams

The midterm exam and final exam will be *digital*.

The questions will be a mix of open and closed questions.

I may choose to ask you to answer some questions on a separate piece of paper.

How do I pass this course?

How do I pass this course? Lectures / hoorcolleges

It sounds obvious – but **you** can do a great deal to make sure you pass the course in one go:

How do I pass this course? Lectures / hoorcolleges

It sounds obvious – but **you** can do a great deal to make sure you pass the course in one go:

- Read through the book before the lecture – what is today's lecture about?

How do I pass this course? Lectures / hoorcolleges

It sounds obvious – but **you** can do a great deal to make sure you pass the course in one go:

- Read through the book before the lecture – what is today's lecture about?
- Come to every lecture – and ask questions when you don't understand something!

How do I pass this course? Lectures / hoorcolleges

It sounds obvious – but **you** can do a great deal to make sure you pass the course in one go:

- Read through the book before the lecture – what is today's lecture about?
- Come to every lecture – and ask questions when you don't understand something!
- Read the material at home carefully. Are you sure you understand everything?

How do I pass this course? Practice sessions / werkcolleges

- Look at the material and exercises before each practice session.

How do I pass this course? Practice sessions / werkcolleges

- Look at the material and exercises before each practice session.

- What did you find difficult? What don't you understand? Try to prepare questions to articulate where you struggled.

How do I pass this course? Practice sessions / werkcolleges

- Look at the material and exercises before each practice session.
- What did you find difficult? What don't you understand? Try to prepare questions to articulate where you struggled.
- Try to do the exercises on your own before the practice sessions. It's OK to get stuck – what help do you need to get unstuck?

How do I pass this course? Practice sessions / werkcolleges

- Look at the material and exercises before each practice session.
- What did you find difficult? What don't you understand? Try to prepare questions to articulate where you struggled.
- Try to do the exercises on your own before the practice sessions. It's OK to get stuck – what help do you need to get unstuck?
- Ask questions to the TAs and your fellow students, if you get stuck.

How do I pass this course? Practice sessions / werkcolleges

- Look at the material and exercises before each practice session.
- What did you find difficult? What don't you understand? Try to prepare questions to articulate where you struggled.
- Try to do the exercises on your own before the practice sessions. It's OK to get stuck – what help do you need to get unstuck?
- Ask questions to the TAs and your fellow students, if you get stuck.
- Do all the exercises that I list.

How do I pass this course? Practice sessions / werkcolleges

- Look at the material and exercises before each practice session.
- What did you find difficult? What don't you understand? Try to prepare questions to articulate where you struggled.
- Try to do the exercises on your own before the practice sessions. It's OK to get stuck – what help do you need to get unstuck?
- Ask questions to the TAs and your fellow students, if you get stuck.
- Do all the exercises that I list.
- Check your solutions and discuss with your work with your peers.

How do I pass this course? Quizzes and exams

How do I pass this course? Quizzes and exams

- Come to the exams and quizzes prepared. Show that you’ve mastered all the material.

How do I pass this course? Quizzes and exams

- Come to the exams and quizzes prepared. Show that you've mastered all the material.
- The closed questions on the quizzes give you feedback about your work – including definitions that you may have misunderstood and sections of the book you should reread.

How do I pass this course? Quizzes and exams

- Come to the exams and quizzes prepared. Show that you've mastered all the material.
- The closed questions on the quizzes give you feedback about your work – including definitions that you may have misunderstood and sections of the book you should reread.
- The open questions will be discussed in class with the TAs. There can be more than one correct answer!

How do I pass this course? Quizzes and exams

- Come to the exams and quizzes prepared. Show that you've mastered all the material.
- The closed questions on the quizzes give you feedback about your work – including definitions that you may have misunderstood and sections of the book you should reread.
- The open questions will be discussed in class with the TAs. There can be more than one correct answer!
- It's OK to mess up the quizzes – but *learn from your mistakes*.

Common mistakes that lead to you failing the course

The first lecture is easy; the second one isn’t much harder.

Common mistakes that lead to you failing the course

The first lecture is easy; the second one isn’t much harder.

But after 2-3 weeks, the material grows complex quite quickly.

And it gets harder and harder to catch up, once you fall behind.

The pace of new material is much higher than in high school.

The material is more abstract than *Imperatief Programming* or *Computerachitectuur en netwerken*.

Common mistakes that lead to you failing the course

The first lecture is easy; the second one isn't much harder.

But after 2-3 weeks, the material grows complex quite quickly.

And it gets harder and harder to catch up, once you fall behind.

The pace of new material is much higher than in high school.

The material is more abstract than *Imperatief Programming* or *Computerachitectuur en netwerken*.

Try to keep up!

Common mistakes that lead to you failing the course

There are solutions to (almost) all exercises available in the book.

If you get stuck, just check the solution!

Common mistakes that lead to you failing the course

There are solutions to (almost) all exercises available in the book.

If you get stuck, just check the solution!

Wrong! Getting stuck is part of learning.

It's easy to get stuck, check the solution, see that it makes sense and move on.

This learning strategy may work fine in high school.

Common mistakes that lead to you failing the course

There are solutions to (almost) all exercises available in the book.

If you get stuck, just check the solution!

Wrong! Getting stuck is part of learning.

It's easy to get stuck, check the solution, see that it makes sense and move on.

This learning strategy may work fine in high school.

But the questions on the exam don't come with solutions! You need to learn how to tackle problems yourself.

Once you're satisfied with your answer, check the solutions.

The TAs are here to help you get unstuck (without giving away the whole answer); or take a step back and *think deeply* about the problem.

Don’t take it from me...

Bonusquestion:

What is the most important thing that you learned in this course?

That you learn more, and that it is easier to pass a course when you pay attention and make the exercises.

Aantal woorden: 21 (aantal tekens: 106)

What is logic?

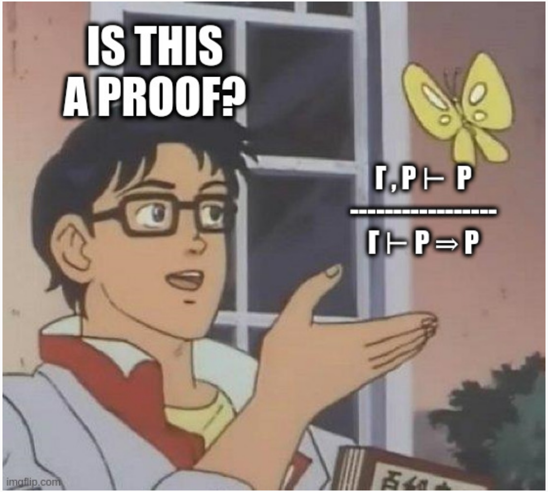
Question: what is a proof?

Logic for computer science

Logic studies the rules of *deduction* – given certain assumptions, what can we conclusions can we draw from them?

By making these rules precise, we can objectively determine if a given statement follows from its assumptions or not.

In summary...


IS THIS
A PROOF?
$\Gamma, P \vdash P$

$\Gamma \vdash P \Rightarrow P$
imgflip.com

Abstraction

When studying logic, we need to be very precise.

Unfortunately, natural languages – such as English or Dutch – are not suitable.

Natural languages are full of ambiguity. Consider a sentence such as:

```
I saw a man on a hill with a telescope.
```

It’s obvious what it means, right?

Abstraction

When studying logic, we need to be very precise.

Unfortunately, natural languages – such as English or Dutch – are not suitable.

Natural languages are full of ambiguity. Consider a sentence such as:

```
I saw a man on a hill with a telescope.
```

It's obvious what it means, right?

There are many different interpretations:

```
There's a man on a hill, and I'm watching him with my telescope.
There's a man, and he's on a hill that also has a telescope on it.
I'm on a hill, and I saw a man using a telescope.
...
```

Abstraction

Open up the book – it's full of mathematical formulas.

To be precise in our reasoning, we sometimes need to work on a more *abstract* level.

As part of this course, this means developing a *language of logic* with a precise meaning that we can use to communicate unambiguously.

We'll study the structure of proofs, independently of the details to the statements and propositions involved.

Abstraction

Open up the book – it’s full of mathematical formulas.

To be precise in our reasoning, we sometimes need to work on a more *abstract* level.

As part of this course, this means developing a *language of logic* with a precise meaning that we can use to communicate unambiguously.

We’ll study the structure of proofs, independently of the details to the statements and propositions involved.

We will often *circle back*, making previous material more precise once you have the mathematical expertise to do so.

Abstraction

A common theme in computer science is to leave out the details that don't matter for the problem at hand.

Analogy: different maps serve different purposes:

- a map to the subway system is useful for planning how to get from one station to another;
- an atlas is useful for studying countries and their geography.
- an street map is useful for finding your way in an unknown city.

Each of these maps leave out certain details and serve a different purpose.

They are all an **abstraction** of reality.

Applications

Just because the logic we will study is **abstract**, doesn't mean there are no applications. Logical deductions pop up over and over again in computer science: . . .

- This method will always return a result greater than 0 because...

Applications

Just because the logic we will study is **abstract**, doesn't mean there are no applications. Logical deductions pop up over and over again in computer science: . . .

- This method will always return a result greater than 0 because...
- The requirements described by our end-user are impossible to fulfill because...

Applications

Just because the logic we will study is **abstract**, doesn't mean there are no applications. Logical deductions pop up over and over again in computer science: . . .

- This method will always return a result greater than 0 because...
- The requirements described by our end-user are impossible to fulfill because...
- The bug must be in this class because...

Modeling computing systems

A **model** is some abstraction of reality that makes it tractable to study.

For example, in high school we often teach the Newtonian model of physics—even if many other models exist that may be more accurate at times.

The mathematics we'll study in this course can be used specifically to model computer programs, or more generally, any system that processes and communicates information.

With a model of computer systems, we can study and predict a system's behaviour, functionality, or performance – in the same way physics can predict the behaviour of billiard balls on a pool table.

Program verification

- **Specification** – an abstract description of what a program must do;
- **Implementation** – a program that (presumably) exhibits the behaviour desired by the specification;
- **Verification** – a proof that an implementation adheres to its specification.

During this course we will study the mathematics that makes such *verification* possible.

The name of this course – *Logic for computer science* – suggests that there are other logic courses taught at the university:

Question

In what other degrees might you find such a course?

The name of this course – *Logic for computer science* – suggests that there are other logic courses taught at the university:

Question

In what other degrees might you find such a course?

- In *Mathematics* – is this proof valid?
- In *Philosophy* – is this argument valid?
- In *Language* – what is the meaning of this sentence?
- In *Law* – what is the correct legal decision?

And possibly many others...

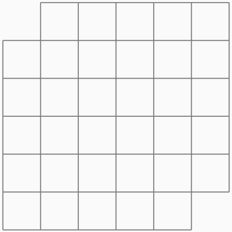
What is logic?

Logic studies the rules of *deduction* – given certain assumptions, what can we conclusions can we draw from them?

By making these rules precise, we can objectively determine if a given statement folllows from its assumptions or not.

What is a proof? And what isn't?

Example: tiling a bathroom



Suppose we need to tile the bathroom above with tiles of size 2x1.

Question

Can we tile this bathroom without breaking or cutting a single tile?

Example: tiling a bathroom

It seems very hard to do...

To prove that it *can* be done, it suffices to find a single succesful tiling.

But how could we possibly prove that it is **not** possible?

Example: tiling a bathroom

It seems very hard to do...

To prove that it *can* be done, it suffices to find a single succesful tiling.

But how could we possibly prove that it is **not** possible?

- We could list every possible tiling and check them one by one, to see that they leave some squares untiled.

  But there are far too many! It's boring and intractable to do this by hand.

Example: tiling a bathroom

It seems very hard to do...

To prove that it *can* be done, it suffices to find a single succesful tiling.

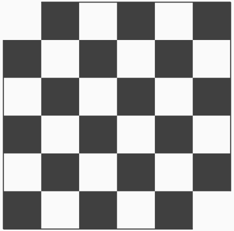
But how could we possibly prove that it is **not** possible?

- We could list every possible tiling and check them one by one, to see that they leave some squares untiled.

  But there are far too many! It's boring and intractable to do this by hand.

Can't we come up with a better idea?

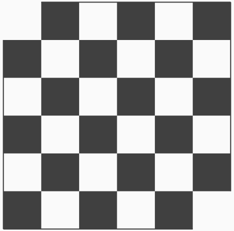
Example: tiling a bathroom



Let’s colour the bathroom as a chessboard, alternating between black and white squares.

Note: there are more ‘black squares’ (18) than there are ‘white squares’ (16).

Example: tiling a bathroom

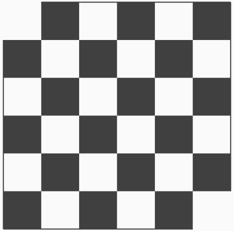


Let’s colour the bathroom as a chessboard, alternating between black and white squares.

Note: there are more ‘black squares’ (18) than there are ‘white squares’ (16).

However we place the first tile, we cover one black and one white square.

Example: tiling a bathroom



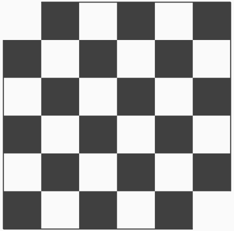
Let’s colour the bathroom as a chessboard, alternating between black and white squares.

Note: there are more ‘black squares’ (18) than there are ‘white squares’ (16).

However we place the first tile, we cover one black and one white square.

But this is also true of the second tile. And the third...

Example: tiling a bathroom



Let's colour the bathroom as a chessboard, alternating between black and white squares.

Note: there are more 'black squares' (18) than there are 'white squares' (16).

However we place the first tile, we cover one black and one white square.

But this is also true of the second tile. And the third...

In the end, there will always be two black squares untiled.

Example: tiling a bathroom

Why show this ‘proof’?

I’m not expecting any of you retile your bathroom with this particular constraints any time soon.

But this proof shows how to prove that something can **never** happen – which is not how we typically think.

Finding such insights is part of what makes computer science fun!

And translating such insights to a precise proof is exactly what this course is about.

Invariants

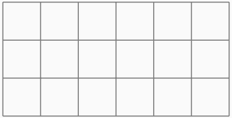
The argument we made was mathematically interesting:

- We made a statement about the number of white tiles and black tiles initially.
- We showed that the statement remaind true regardless of the next tile that is placed

Such a statement is sometimes called an **invariant**

This is particularly useful when reasoning about a computer program's behaviour, where regardless of the user's input, the exact data in memory or on disk, or the exact execution trace of a program, a certain property must hold.

Example: chocolate bar

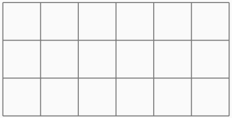


Here is a 3x6 chocolate bar. I can break it along any line in the usual fashion. What is the least number of times I need to break it in order to end up with 18 individual pieces to share among my friends? (I am not strong enough to break two parts of the chocolate bar at the same.)

Question

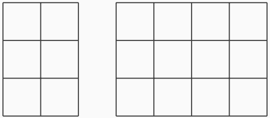
What is the best strategy for breaking the chocolate bar? How many times will I need to break it?

Example: chocolate bar



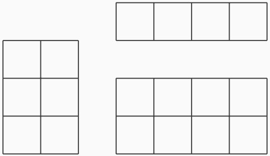
Initially, we have 1 piece of chocolate after 0 cuts.

Example: chocolate bar



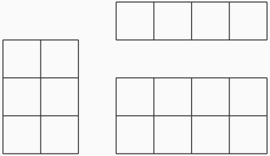
After the first cut, we have 2 pieces.

Example: chocolate bar



After the second cut, we have 3 pieces.

Example: chocolate bar



After n cuts, we have $n + 1$ pieces.

With 17 cuts, we'll break the chocolate bar into 18 pieces.

Another example of an **invariant**.

Back to computer programs

But what do these examples have to do with programming?

Let’s study one last example of an invariant...

Greatest common divisor

When working with fractions, we usually want to *simplify* them, writing $\frac{3}{5}$ rather than $\frac{15}{25}$.

How can we write a computer program that, given two numbers x and y, computes the simplified fraction of corresponding to $\frac{x}{y}$?

Greatest common divisor

When working with fractions, we usually want to *simplify* them, writing $\frac{3}{5}$ rather than $\frac{15}{25}$.

How can we write a computer program that, given two numbers x and y, computes the simplified fraction of corresponding to $\frac{x}{y}$?

One way to do so is to compute the *greatest common divisor* (or gcd) of both x and y, and then return the fraction

$$\frac{x \div \mathrm{gcd}(x, y)}{y \div \mathrm{gcd}(x, y)}$$

In our example, $\mathrm{gcd}(15, 25)$ should return 5; hence we can simplify $\frac{15}{25}$ to $\frac{3}{5}$.

Examples

- $\gcd(12, 36)$ should return 12
- $\gcd(17, 3)$ should return 1
- $\gcd(15, 15)$ should return 15

These examples hopefully illustrate the specification.

But what program can we write to return the gcd?

Brute force search

We could write a simple for loop that counts down from $max(x, y)$ to 1, looking for the first common divisor.

But there's a much smarter algorithm that is more than two thousand years old....

Euclid's algorithm

Euclid

Euclid's algorithm

- The greatest common divisor of x and x is x.
- When $x > y$, the greatest common divisor of x and y is the same as the greatest common divisor of $(x - y)$ and y.
- When $x < y$, the greatest common divisor of x and y is the same as the greatest common divisor of x and $(y - x)$.

Why does this work?

Euclid's algorithm

- The greatest common divisor of x and x is x.
- When $x > y$, the greatest common divisor of x and y is the same as the greatest common divisor of $(x - y)$ and y.
- When $x < y$, the greatest common divisor of x and y is the same as the greatest common divisor of x and $(y - x)$.

Why does this work?

- The greatest common divisor of x and y must also be a divisor of $x - y$...
- There can be no greater divisor.

(Proofs left as an exercise to be completed at the end of the course)

From algorithm to code

- The greatest common divisor of x and x is x.
- When $x > y$, the greatest common divisor of x and y is the same as the greatest common divisor of $(x - y)$ and y.
- When $x < y$, the greatest common divisor of x and y is the same as the greatest common divisor of x and $(y - x)$.

But this can easily be translated to a simple method in C#!

And now in C#...

```
int Euclid(x,y : int) {
  // Assuming x and y are both greater than 0
  while x != y {
    if (x > y) {
      x = x-y;
    }
    else {
      y = y-x;
    }
  }
  return x;
}
```

Invariants and computer programs

Invariants give us a way to reason about programs *as they are executed*!

We haven't really done any formal logic yet…

… but I hope to have planted the idea that we might want to reason about computer programs precisely – and even prove them correct.

And that's one learning outcome of this course.

Logic for computer science: course summary

- *propositional logic* – the basic logic from which much can be built;
- *sets* – that can be used to model all kinds of data that a computer processes;
- *predicate logic* – that can be used to reason about sets;
- how to *prove* statements in propositional and predicate logic;
- *induction* and *recursion* – allow you to define infinite sets and prove properties of them;
- *functions*, *relations*, *games* and *processes* – that allow you to model computer programs;
- how to use these techniques in a *formal* study of logic;
- how to use these techniques to reason about computer programs;
- how computers can (help) perform logical reasoning.

Recap

- Organisation
- What is logic?

Next lecture – propositional logic

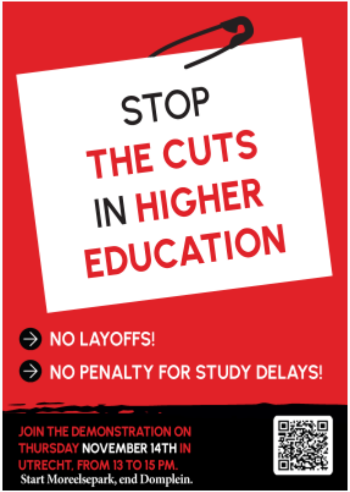
George Boole

Know more?

- Read chapter 0 in the book.

But there are plenty of excellent introductions to logic for a general audience these days:

- *Logicomix – An Epic Search for Truth* door Apostolos Doxiadis and Christos Papadimitriou;
- *The Art of Logic* by Eugenia Chang – how to apply logical thinking to society
- Any book by Raymond Smullyan – who tries to explain important logical results to a general audience through brainteasers.
- Or you may want to check out the other literature suggested on the course website.



Fight! Fight! Fight!

See you Thursday!