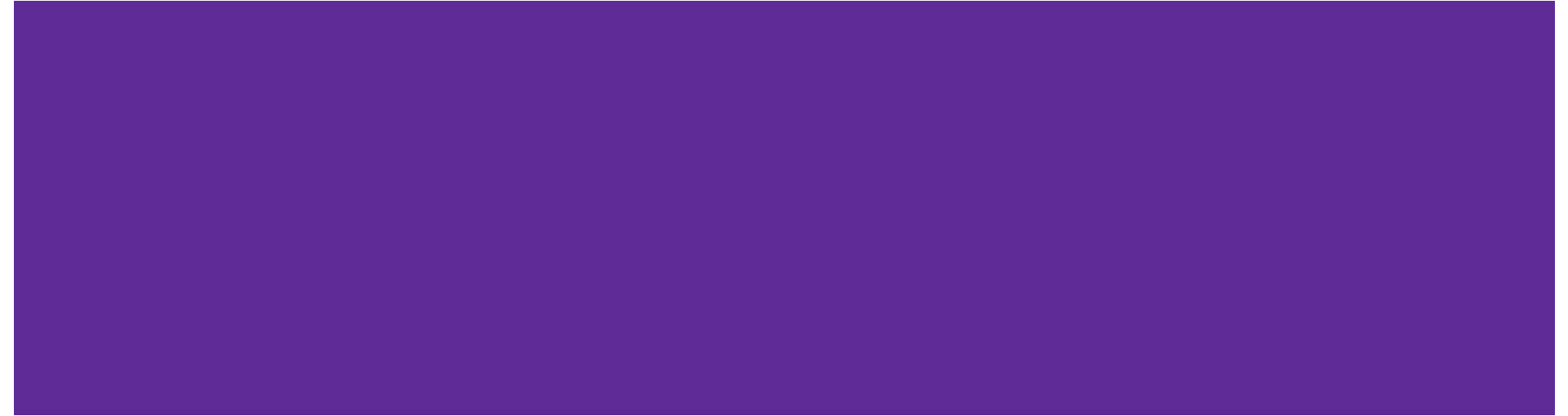


# CSE 421 Section 8

**P, NP, Reductions**

# Administrivia



# Announcements & Reminders

- HW 6
  - If you think something was graded incorrectly, submit a regrade request!
- HW 7
  - Was due yesterday, Wednesday 3/1
- HW 8
  - It's the last homework, woohoo!
  - Due Wednesday 3/8
- Final Exam
  - Scheduled for **Monday 3/13 @ 2:30-4:20 in our normal room, CSE2 G20**

# Problems, P, & NP



# First, some Definitions:

- **Problem:** a set of inputs and the correct outputs
- **Instance:** a single input to a problem
- **Decision Problem:** a problem where the output is “yes” or “no”
- **Reduction:**  $A \leq B$ 
  - Informally: **A reduces to B** means “we can solve A using a library for B”
  - Formally:  $A$  reduces to  $B$  in **polynomial time** if there is an algorithm to solve problem  $A$ , which, if given access to a library function for solving problem  $B$ , calls the library at most polynomially-many times and takes at most polynomial-time otherwise excluding the calls to the library.

# P, NP, and P vs. NP

- **P (“polynomial”)**: The set of all decision problems for which there exists an algorithm that runs in time  $\mathcal{O}(n^k)$  for some constant  $k$ , where  $n$  is the size of the input
- **NP (“nondeterministic polynomial”)**: The set of all decision problems such that for every YES-instance (of size  $n$ ), there is a certificate (of size  $\mathcal{O}(n^k)$ ) for that instance which can be verified in polynomial time
- **P vs. NP**: Are P and NP the same complexity class?
  - That is, can every problem that can be **verified** in polynomial time also be **solved** in polynomial time?

# NP-hard, NP-complete

- **NP-hard:** The problem  $B$  is NP-hard if for all problems  $A$  in NP,  $A$  reduces to  $B$
- **NP-complete:** The problem  $B$  is NP-complete if  $B$  is in NP and  $B$  is NP-hard

# 3-SAT

In simple terms:

**Input:** expression in CNF (AND of ORs) form, where every term has exactly 3 literals.

**Output:** true if there is a variable setting which makes the whole expression true, false otherwise.

More formally:

**Input:**

- A list of Boolean variables  $x_1, \dots, x_n$
- A list of constraints, all of which must be met. Each constraint is of the form:  
 $(z_i \vee z_j \vee z_k)$ , where  $z_i$  is a “literal” (a variable or the negation of a variable).

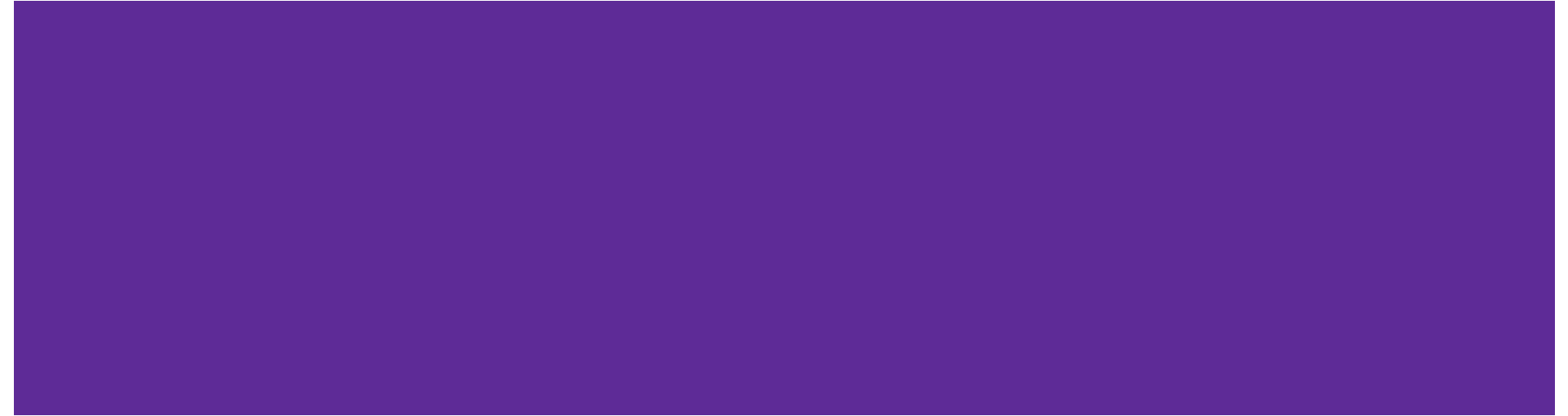
**Output:** true if there is a setting of the variables where all constraints are met, false otherwise.

# 3-SAT

Why is it called 3-SAT? 3 because you have 3 variables per constraint, SAT is short for “satisfiability”. The problem is asking, can you find an assignment that satisfies all of the constraints?

3-SAT is an **NP-Complete problem**. This means every problem in NP can be reduced to it (it is NP-Hard) and it is also in NP.

# 1. SATisfy This



# Problem 1 – SATisfy This

Determine whether each instance of 3-SAT is satisfiable. If it is, list a satisfying variable assignment.

a)  $(\neg a \vee \neg b \vee c) \wedge (a \vee c \vee \neg d) \wedge (b \vee \neg c \vee \neg d) \wedge (\neg a \vee b \vee c) \wedge (\neg b \vee c \vee \neg d)$

b)  $(\neg a \vee b \vee d) \wedge (\neg b \vee c \vee d) \wedge (a \vee \neg c \vee d) \wedge (a \vee \neg b \vee \neg d)$   
 $\wedge (b \vee \neg c \vee \neg d) \wedge (\neg a \vee c \vee \neg d) \wedge (a \vee b \vee c) \wedge (\neg a \vee \neg b \vee \neg c)$

c)  $(a \vee \neg c \vee d) \wedge (\neg a \vee b \vee c) \wedge (b \vee \neg c \vee \neg d) \wedge (a \vee \neg c \vee d) \wedge (a \vee \neg b \vee c)$   
 $\wedge (\neg a \vee b \vee \neg d) \wedge (\neg a \vee c \vee d) \wedge (b \vee \neg c \vee d) \wedge (a \vee c \vee \neg d)$

d)  $(\neg a \vee \neg b \vee c) \wedge (a \vee b \vee \neg c) \wedge (\neg a \vee \neg b \vee \neg c) \wedge (\neg a \vee b \vee c)$   
 $\wedge (a \vee \neg b \vee \neg c) \wedge (\neg a \vee b \vee \neg c) \wedge (a \vee b \vee c) \wedge (a \vee \neg b \vee c)$

Work through (a) and (b) with the people around you, and then we'll go over them together!

## Problem 1 – SATisfy This

a)  $(\neg a \vee \neg b \vee c) \wedge (a \vee c \vee \neg d) \wedge (b \vee \neg c \vee \neg d) \wedge (\neg a \vee b \vee c) \wedge (\neg b \vee c \vee \neg d)$

## Problem 1 – SATisfy This

$$\begin{aligned} \text{b)} \quad & (\neg a \vee b \vee d) \wedge (\neg b \vee c \vee d) \wedge (a \vee \neg c \vee d) \wedge (a \vee \neg b \vee \neg d) \\ & \wedge (b \vee \neg c \vee \neg d) \wedge (\neg a \vee c \vee \neg d) \wedge (a \vee b \vee c) \wedge (\neg a \vee \neg b \vee \neg c) \end{aligned}$$

# Reductions



# Why do we care about NP-Hard & NP-Complete?

Let  $B$  be an NP-hard problem. Remember, this means that every problem in NP reduces to  $B$ . Suppose you found a polynomial time algorithm for  $B$ ; you now have for free a polynomial time algorithm for every problem in NP, so  $P = NP$ .

On the other hand, if any problem in  $NP$  is not in  $P$  (any doesn't have a polynomial time algorithm), then no NP-complete problem is in  $P$ .

# What can we do with NP-Hard & NP-Complete?

We're pretty sure that there aren't any efficient algorithms for NP-complete problems. When you're asked to write an algorithm for a problem, it is worthwhile for you to be able to tell if the problem is NP-complete or not. How can we do that?

We need to show that a problem that is NP-complete is easier (or at most equal in difficulty) to the problem! If it's harder than an NP-complete problem, then it's NP-hard. If the problem is also in NP, then it's NP-complete as well!

How can we show this? We can write a reduction!

# NP-Completeness Reductions

Given problem  $A$ , to prove that it is NP-hard, we take a known NP-complete problem  $B$ , and show that  $B \leq_P A$ . To prove that  $A$  is NP-complete, we also show that it is in NP.

Essentially, you need to take the input to the NP-complete problem  $B$  and transform it into the input to the problem  $A$  so that an algorithm for  $A$  would return true on this modified input if and only if  $B$  is true on the original input.

The reduction algorithm is the process of transforming that input to  $B$  into the input to  $A$ .

# NP-Completeness Reductions: which way?

How do you remember which direction? The core idea of an NP-completeness reduction is a proof by contradiction:

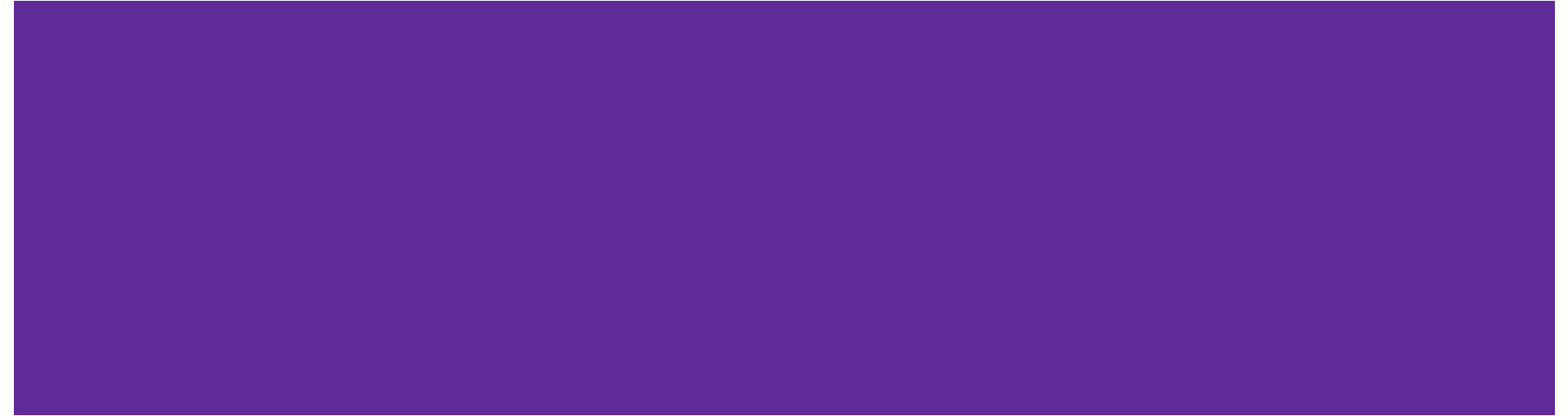
Suppose, for the sake of contradiction, there were a polynomial time algorithm for  $B$ . But then if there were, I could use that to design a polynomial time algorithm for problem  $A$ . But we *really* don't think there's a polynomial time algorithm for problem  $A$ . So we should *really* think there isn't one for  $B$  either!

**Key Idea:** Reduce FROM the known hard problem TO the new problem.

# Strategy for Reductions

1. Read and Understand the Problem
2. Design the Reduction
3. Write the Proof
  - Prove Run-Time
  - Prove correctness; requires TWO implications:
    - If the correct answer is YES, then our algorithm says YES
    - If our algorithm says YES, then the correct answer is YES

## 2. A Fun Reduction



## Problem 2 – A Fun Reduction

Define 5-SAT as the following problem:

**Input:** An expression in CNF form, where every term has exactly 5 literals.

**Output:** true if there is a variable setting which makes the whole expression true, false otherwise.

And 3-SAT as earlier:

**Input:** expression in CNF form, where every term has exactly 3 literals.

**Output:** true if there is a variable setting which makes the whole expression true, false otherwise.

Prove that 5-SAT is NP-complete using 3-SAT.

# Problem 2.1 – Read and Understand the Problem

First understand 5-SAT:

- What is the input type?
- What is the output type?
- Are there any technical terms in the problem you don't know? Are there any words that look like normal words, but are actually technical terms?

Then think about the reduction:

- Which problem are you solving, and which problem are you assuming you have an algorithm for? Make sure your reduction is “going the right direction”
- What is the output type for your reduction?

## Problem 2.2 – Design the Reduction

Now write a reduction. Remember a reduction is an algorithm! It often helps to think about the “certificates” (the thing that makes it a YES instance) and transform from one type of certificate to the other.

## Problem 2.2 – Design the Reduction

Now write a reduction. Remember a reduction is an algorithm! It often helps to think about the “certificates” (the thing that makes it a YES instance) and transform from one type of certificate to the other.

**Hint:** We want to start with the input to 3-SAT and transform it into the input to 5-SAT so that 5-SAT returns true iff 3-SAT would also return true!

## Problem 2.3 – Write the Proof

- a) To be NP-Complete, 5-SAT needs to be in NP. Argue that it is (this argument is usually only 2-3 sentences).
- b) Show your reduction is correct. Remember you need to prove two implications and that the running time is polynomial.

## Problem 2.3 – Write the Proof

- a) To be NP-Complete, 5-SAT needs to be in NP. Argue that it is (this argument is usually only 2-3 sentences).

## Problem 2.3 – Write the Proof

b) Show your reduction is correct. Running Time:

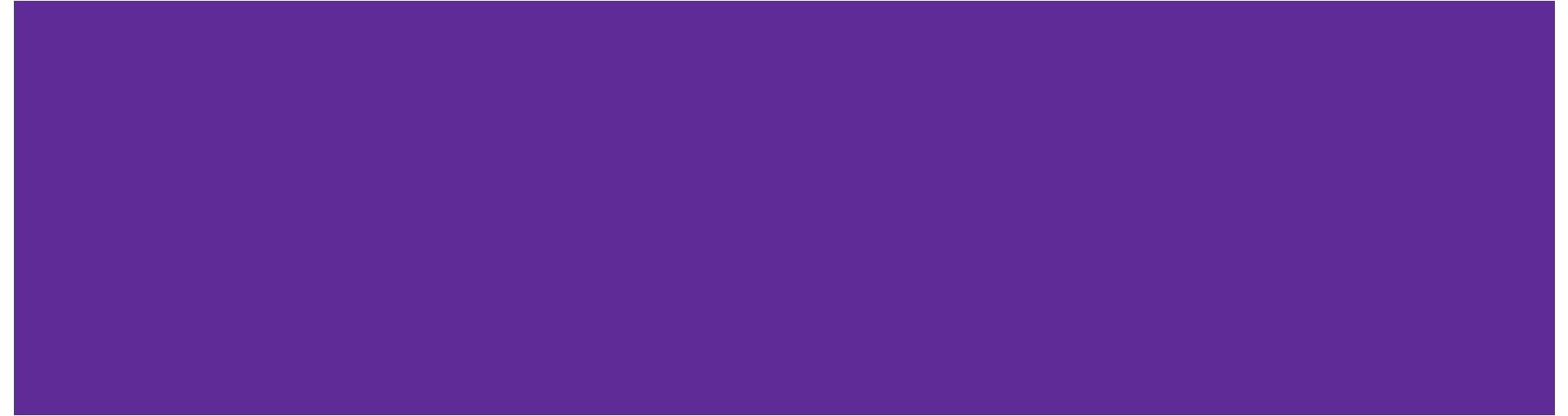
## Problem 2.3 – Write the Proof

- b) Show your reduction is correct. Suppose correct answer is YES and our reduction returns YES:

## Problem 2.3 – Write the Proof

- b) Show your reduction is correct. Suppose our reduction returns YES and correct answer is YES:

## 2. A Tricky Reduction



## Problem 3 – A Tricky Reduction

Define IND-SET as follows:

**Input:** An undirected graph  $G$  and a positive integer  $k$

**Output:** true if there is an independent set in  $G$  of size  $k$  (or more), false otherwise.

And 3-SAT as earlier:

**Input:** expression in CNF form, where every term has exactly 3 literals.

**Output:** true if there is a variable setting which makes the whole expression true, false otherwise.

Prove that IND-SET is NP-complete using 3-SAT.

# Problem 3.1 – Read and Understand the Problem

First understand IND–SET:

- What is the input type?
- What is the output type?
- Are there any technical terms in the problem you don't know? Are there any words that look like normal words, but are actually technical terms?

Then think about the reduction:

- Which problem are you solving, and which problem are you assuming you have an algorithm for? Make sure your reduction is “going the right direction”
- What is the output type for your reduction?

Work through these questions with the people around you, and then we'll go over them together!

## Problem 3.2 – Design the Reduction

Now write a reduction. Remember a reduction is an algorithm! It often helps to think about the “certificates” (the thing that makes it a YES instance) and transform from one type of certificate to the other.

Work through this problem with the people around you, and then we'll go over it together!

## Problem 3.2 – Design the Reduction

Now write a reduction. Remember a reduction is an algorithm! It often helps to think about the “certificates” (the thing that makes it a YES instance) and transform from one type of certificate to the other.

**Hint:** We want to start with the input to 3-SAT and transform it into the input to IND-SET so that IND-SET returns true iff 3-SAT would also return true!

Work through this problem with the people around you, and then we'll go over it together!

## Problem 3.2 – Design the Reduction

Now write a reduction. Remember a reduction is an algorithm! It often helps to think about the “certificates” (the thing that makes it a YES instance) and transform from one type of certificate to the other.

## Problem 3.3 – Write the Proof

- a) To be NP-Complete, IND-SET needs to be in NP. Argue that it is (this argument is usually only 2-3 sentences).
- b) Show your reduction is correct. Remember you need to prove two implications and that the running time is polynomial.

Work through this problem with the people around you, and then we'll go over it together!

## Problem 3.3 – Write the Proof

- a) To be NP-Complete, IND-SET needs to be in NP. Argue that it is (this argument is usually only 2-3 sentences).

## Problem 3.3 – Write the Proof

b) Show your reduction is correct. Running Time:

## Problem 3.3 – Write the Proof

- b) Show your reduction is correct. Suppose correct answer is YES and our reduction returns YES:

## Problem 3.3 – Write the Proof

- b) Show your reduction is correct. Suppose our reduction returns YES and correct answer is YES:

# **That's All, Folks!**

**Thanks for coming to section this week!**  
**Any questions?**