

# Intro to Computer Science

## Previous

- Control statements
  - if-elif-else

## Next

- Boolean algebra

## Readings

Gaddis

- Chapter 3.4

## Readings

Gaddis

- Chapter 3

# Type: bool

## Values

- Boolean literals  
True, False
- Capitalization matters!

## Operations

Negation	not
Conjunction	and
Disjunction	or
Exclusive or	xor
Order	<
	<=
	>
	>=
Equality	==
	!=

# How does this work: negation

- Negation requires a single Boolean operand
- Works by “swapping” the Boolean value
  - Akin to multiplying by -1

x	Operation	Result
0	not 0	1
1	not 1	0



Using 1 and 0 for True and False, respectively (for brevity)

# How does this work: conjunction

- Conjunction requires two Boolean operands
- The only time it's true is when both operands are true

x	y	Operation	Result
0	0	0 and 0	0
1	0	1 and 0	0
0	1	0 and 1	0
1	1	1 and 1	1

# How does this work: disjunction

- Disjunction requires two Boolean operands
- The only time it's false is when both operands are false

x	y	Operation	Result
0	0	0 or 0	0
1	0	1 or 0	1
0	1	0 or 1	1
1	1	1 or 1	1

# Truth tables

- A *truth table* displays the result of a logical expression
- Handy for complex expressions
  - Understanding (proving, even) the expected result
  - Simplifying the original statement

x	not x
0	1
1	0

x	y	x and y
0	0	0
1	0	0
0	1	0
1	1	1

x	y	x or y
0	0	0
1	0	1
0	1	1
1	1	1

# Understanding an expected result

- We often need to convince ourselves that we have a complete understanding of an expressions outcome, regardless of its input
  - More formally, we need to *prove its correctness*
- Usually the case when the expression is complicated
  - Simple method: build a truth table
    - Calculate combinations in order of priority (descending)

```
if not (x and y) or x:  
    print('True')
```

Why is the inner  
block always run?!?!?

x	y	x and y	not (x and y)	not (x and y) or x
0	0	0	1	1
0	1	0	1	1
1	0	0	1	1
1	1	1	0	1

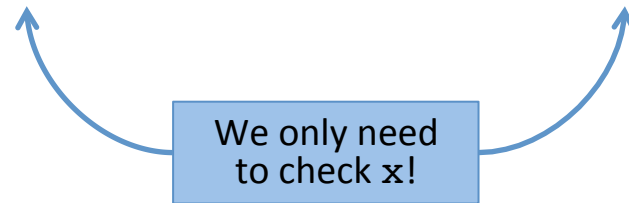
Because it's  
always true!  
(Doh!)

# Reducing a complicated statement

- Other programming constructs can get complicated
  - Boolean operations are no exception
- This opens the door for redundancy!

```
if x and y or x:  
    print('True')
```

x	y	x and y	x and y or x
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

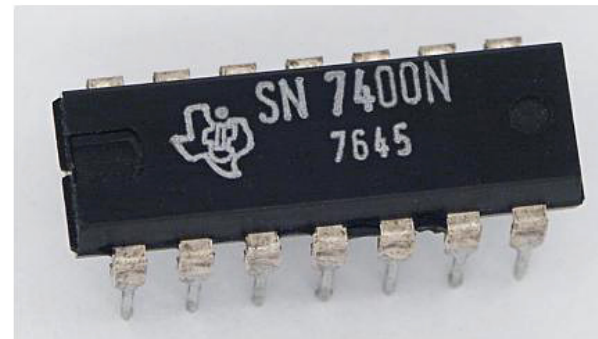
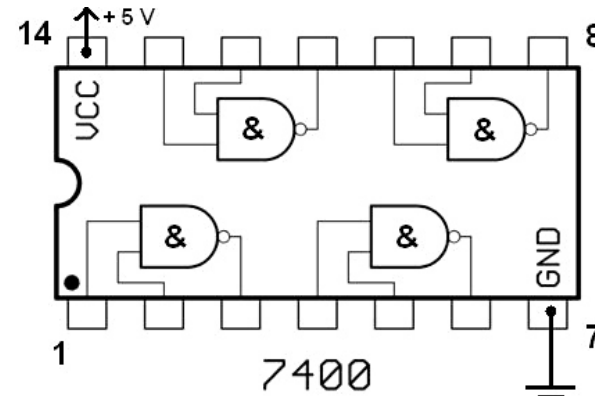




# Boolean simplification is real!

## Hardware

- Boolean logic is implemented using *transistors*
- Transistors contain *gates* that turn a signal on or off
- Transistors
  - Cost money
  - Take up space
- Reducing transistors is a good thing



- Transistors cost real money
- Electronics need lots of them (Apple ships 10s of millions of iPhones per year)

# Boolean simplification is real!

## Software

- Boolean logic is implemented using *decision structures*
- Conditional statements take time to run
  - Which is ultimately money
- Reducing conditionals is a good thing

Evaluating only x would save us lots of time!

```
if x and y or x:  
    print('True')
```

This fun...

... but what if

- x and y were functions
- and a time-dependent transaction relied on their evaluation

```
shares = 0  
for i in purchase_conditions:  
    if x(i) and y(i) or x(i):  
        shares = shares + 1  
  
purchase_stock(shares)
```

# How does this work: xor

- True when the *two* values are different

x	y	Operation	Result
0	0	$0 \wedge 0$	0
1	0	$1 \wedge 0$	1
0	1	$0 \wedge 1$	1
1	1	$1 \wedge 1$	0

# Does this look familiar?

x	y	Boolean operation	Math operation	Result
0	0	$0 \wedge 0$	?	0
0	1	$1 \wedge 0$	?	1
1	0	$0 \wedge 1$	?	1
1	1	$1 \wedge 1$	?	0

Direct result of addition! (x+y)

They're actually both the one's place of the respective addition:


- $0_2 + 0_2 = 00_2$
- $0_2 + 1_2 = 01_2$
- $1_2 + 0_2 = 01_2$
- $1_2 + 1_2 = 10_2$

"One's" place of 1+1

- Remember:  $1_2 + 1_2 = 10_2$

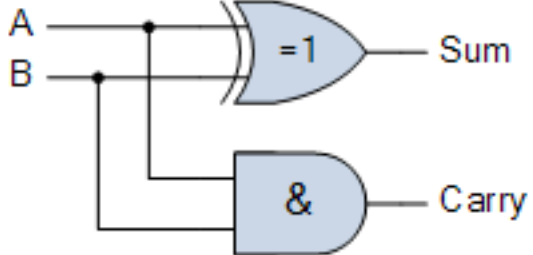
# What about the carry?

x	y	Boolean operation	Math operation	Carry	Ones
0	0	$0 \wedge 0$	$0 + 0$	0	0
0	1	$1 \wedge 0$	$1 + 0$	0	1
1	0	$0 \wedge 1$	$0 + 1$	0	1
1	1	$1 \wedge 1$	$1 + 1$	1	0



We've seen this before!

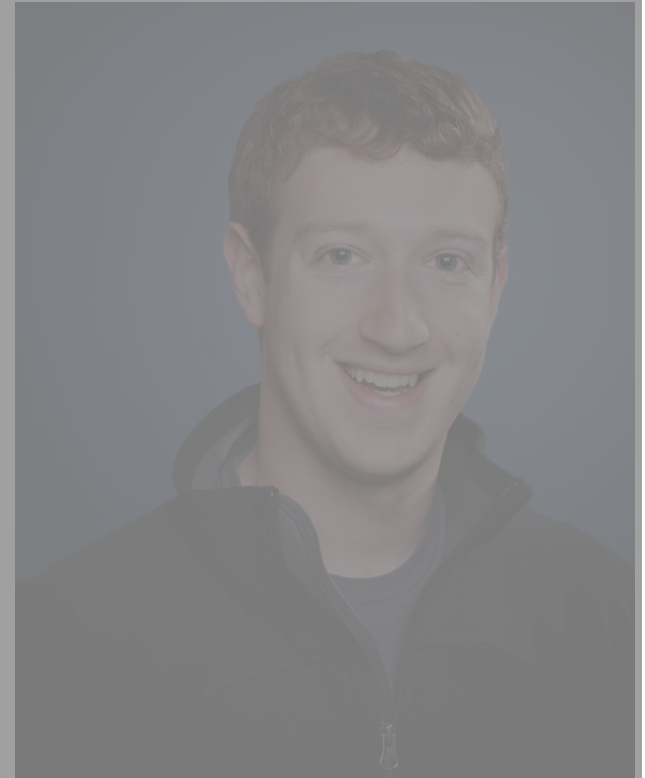
# Technically

Symbol	Truth Table			
	B	A	SUM	CARRY
	0	0	0	0
	0	1	1	0
	1	0	1	0
	1	1	0	1

“DON'T JUST PLAY  
ON YOUR PHONE,  
PROGRAM IT”

“Every girl deserves to take  
part in the technology that  
will change our world, and  
change who runs it.”

“In fifteen years, we'll be teaching  
programming just like reading and  
writing — and wondering why we  
didn't do it sooner.”



## THE COMPUTER SCIENCE OPEN HOUSE

Learn what you can do with a computer science  
education

Tuesday, Sep 26  
7-8 PM, A3-001

Pizza and Drinks