Lecture 06

OpAmps

Agenda

00. Stuff – Recap

01. Operational Amplifiers (Op Amps)

02. Basic Coding practice

OH and Covid reminder

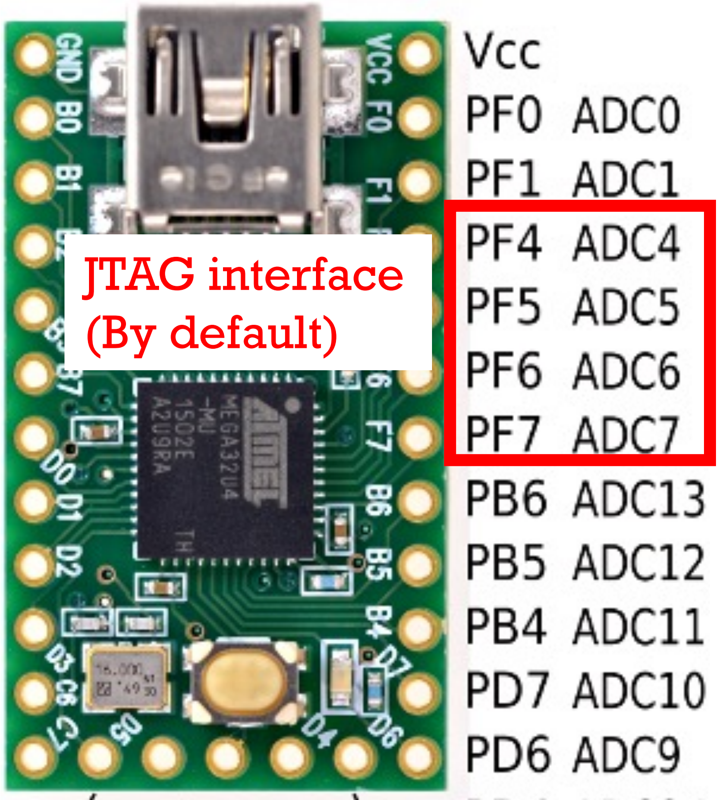
- According to https://coronavirus.upenn.edu/content/dashboard we had 28 students last week that tested positive.
- The lab is getting crowded - especially last night. Please do the openpass every day just to help track things.
- When lab OH is crowded, we will use the piazza_queue on Piazza. Send a message on Piazza to get in the queue for help.

Random Teensy Things

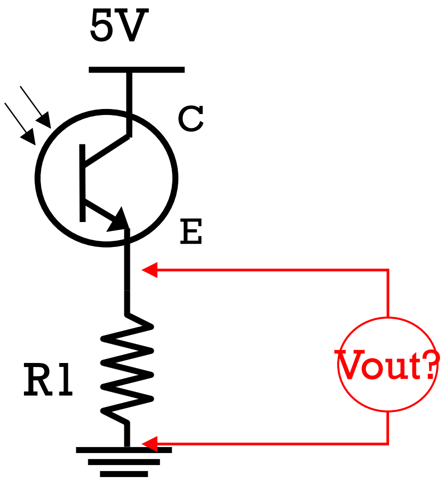
- Using Pins F4-F7 are special (need to disable JTAG for ADC or GPIO)

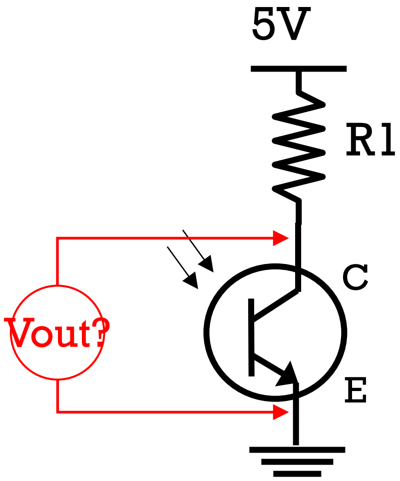
```
teensy_disableJTAG();
```

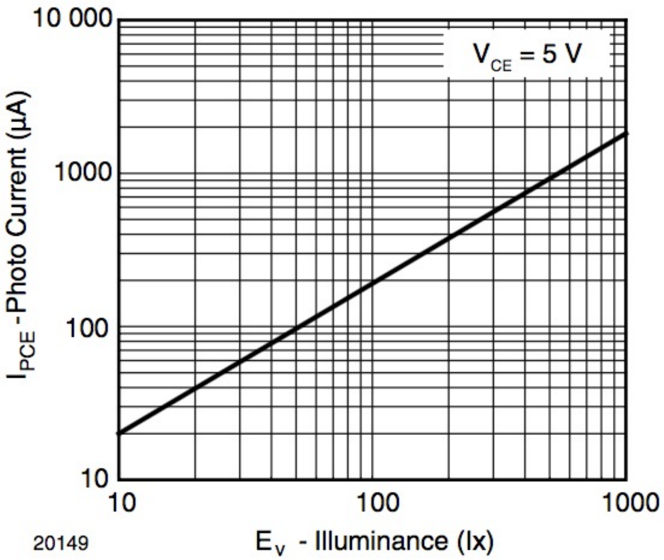
- Better to use `int` than `float` if possible on the Atmega. `float` operations are about 100x's slower and take up a lot of memory.



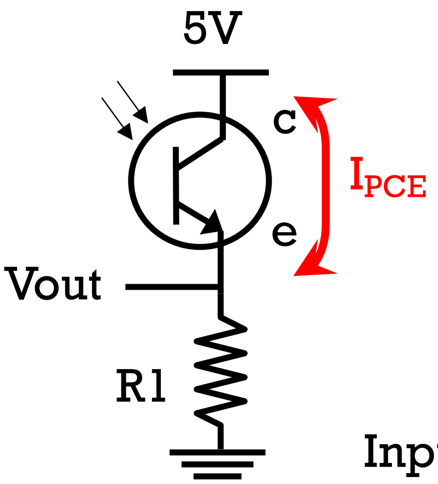
Phototransistors: How to use?







Sensitivity

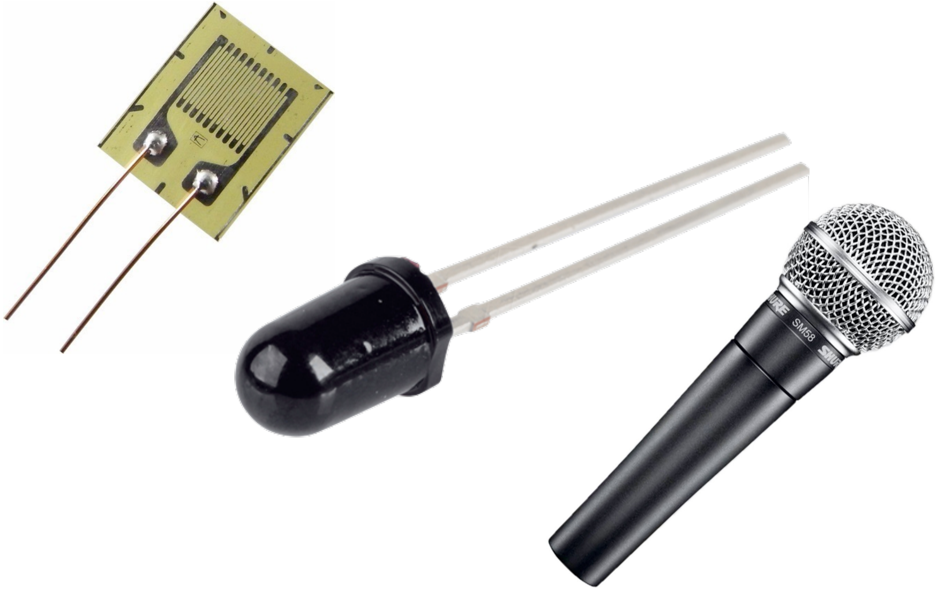


Case	R1	100Ω	1000Ω	10Ω
Light 1	I_{PCE}	2mA	2mA	2mA
	V_{out}	0.2V	2V	0.02V
Light 2	I_{PCE}	3mA	3mA	3mA
	V_{out}	0.3V	3V	0.03V
Input range ΔI_{PCE}		1mA	1mA	1mA
Output range ΔV_{OUT}		0.1V	1V	0.01V

Not sensitive!

How much more gain is there from ^ this case to ^ this case

Typically sensors need to be amplified

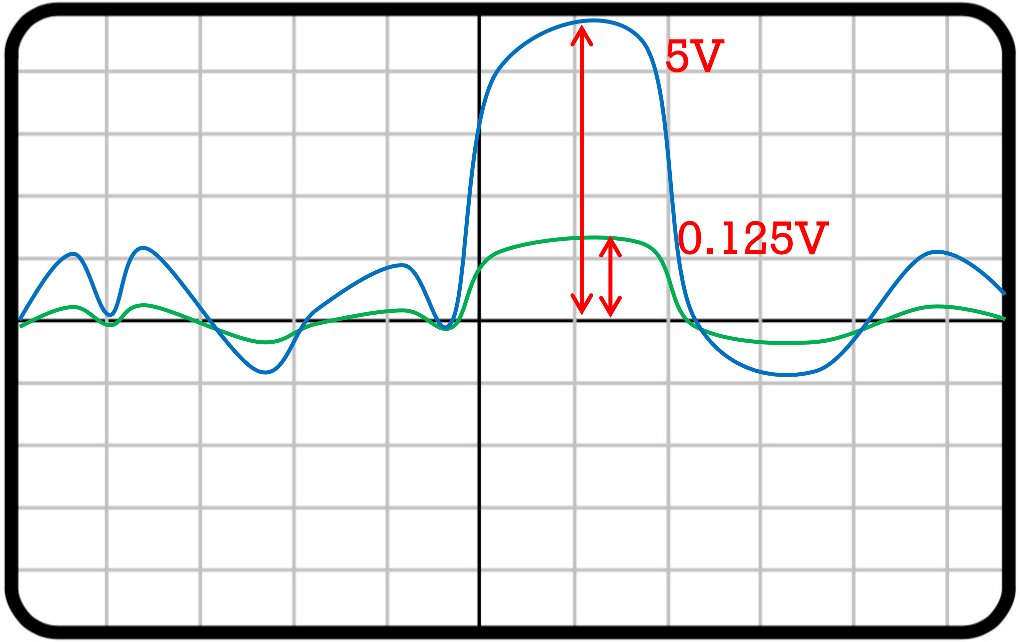


- Strain gauges
- Photodiodes
- Audio signals
- The amount of amplification is called GAIN
  - Gain = Output/Input
  - (Usually) gain = Vout/Vin

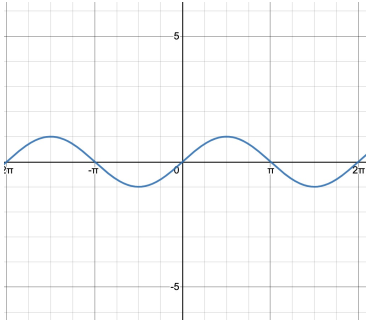
Amplifying small signals.

Scope output of a photo sensor circuit (unamplified)

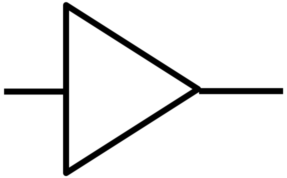
(amplified x40) to get to TTL level



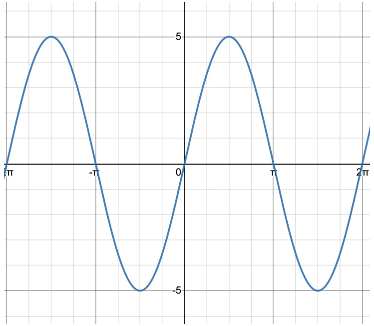
Amplifier symbols



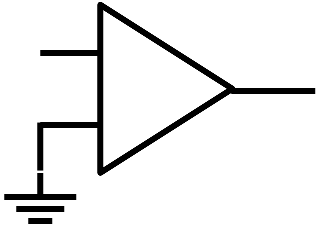
Input
Voltage
signal



Output
Voltage
signal



Input
Voltage
signal

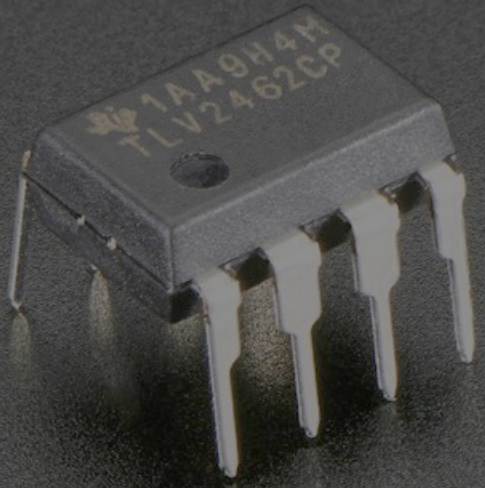


Output
Voltage
signal

01

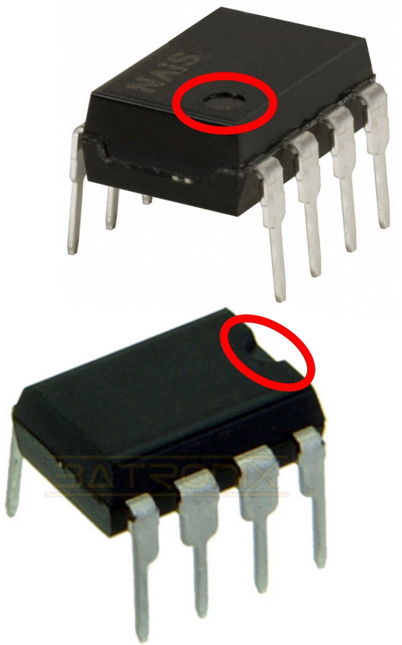
Op Amps

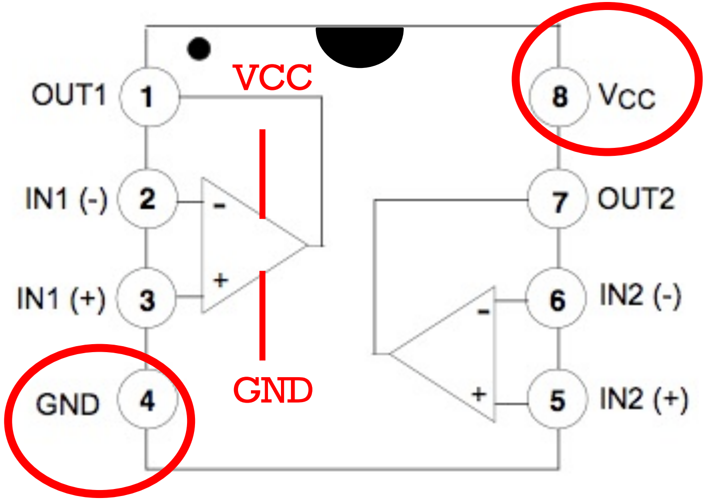
Operational Amplifier



Op Amp

- LM358 in GMLab, generic, low cost, single supply dual opamp.



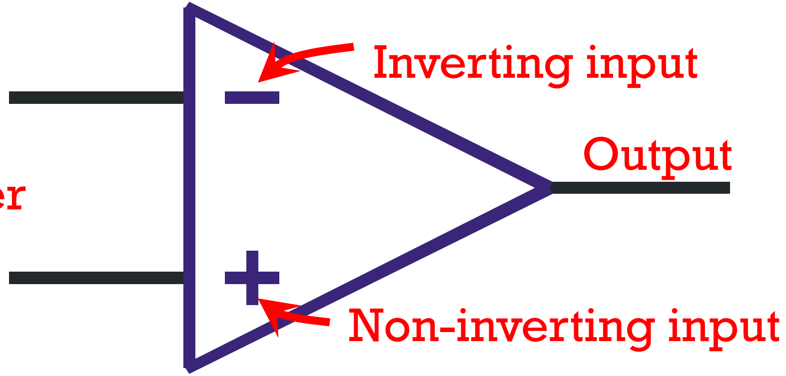


view from the top (pins going into the page)

- TLV272 in GMLab, much better specs, costs 3x more.
  - use this one, not LM358

The Ideal Operational Amplifier

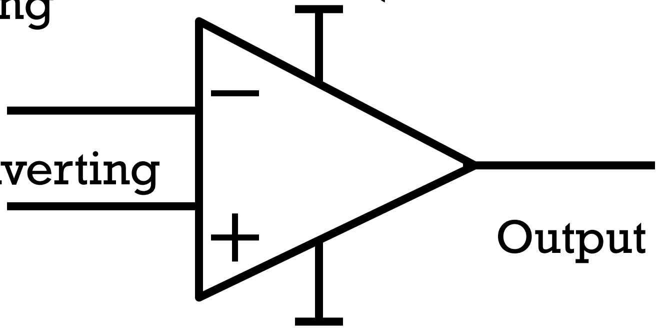
Difference amplifier



Inverting (V)		Noninverting (V)	Output (V)	
Higher	>	Lower	Goes down	As low as possible
Lower	<	Higher	Goes up	As high as possible

Gain is the ratio between difference and output
Op Amp openloop gain is very high, consider it infinite.

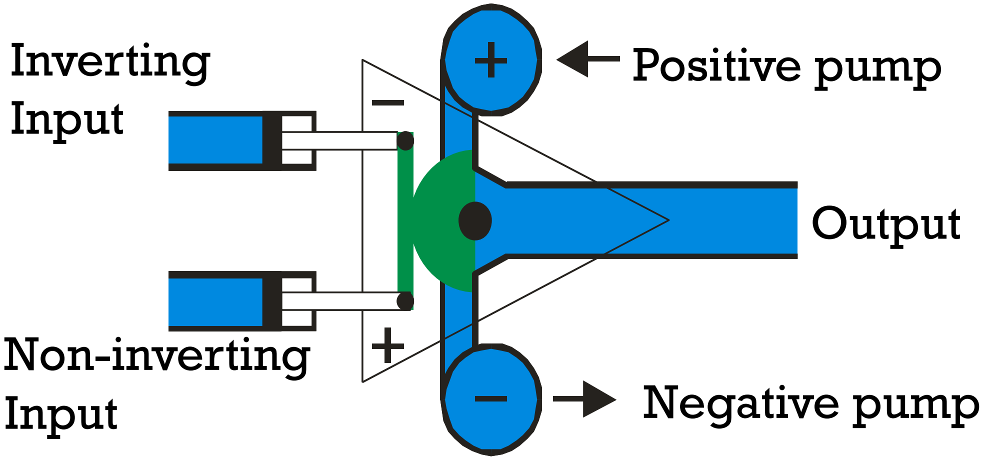
Opamp Positive and Negative Supplies



As low as possible = V-

As high as possible = V+

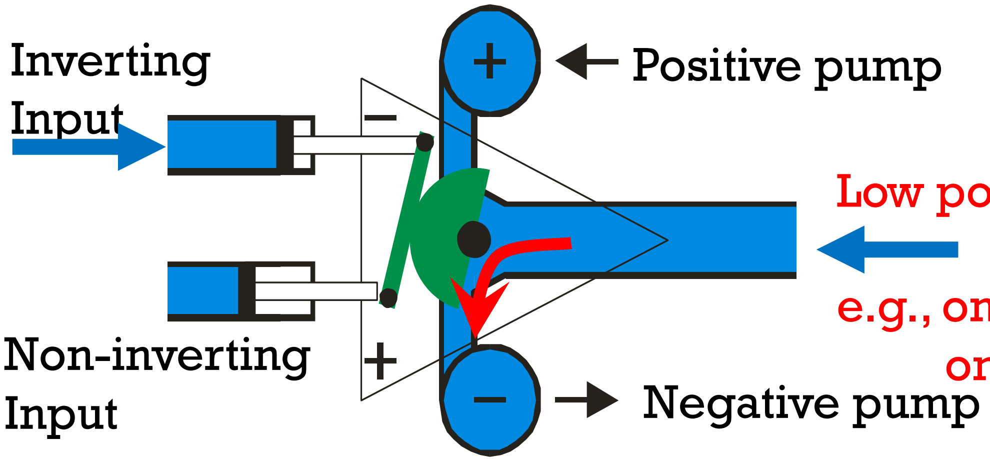
Opamp water analogy



1st Golden rule: Inputs draw no current

Opamp water analogy

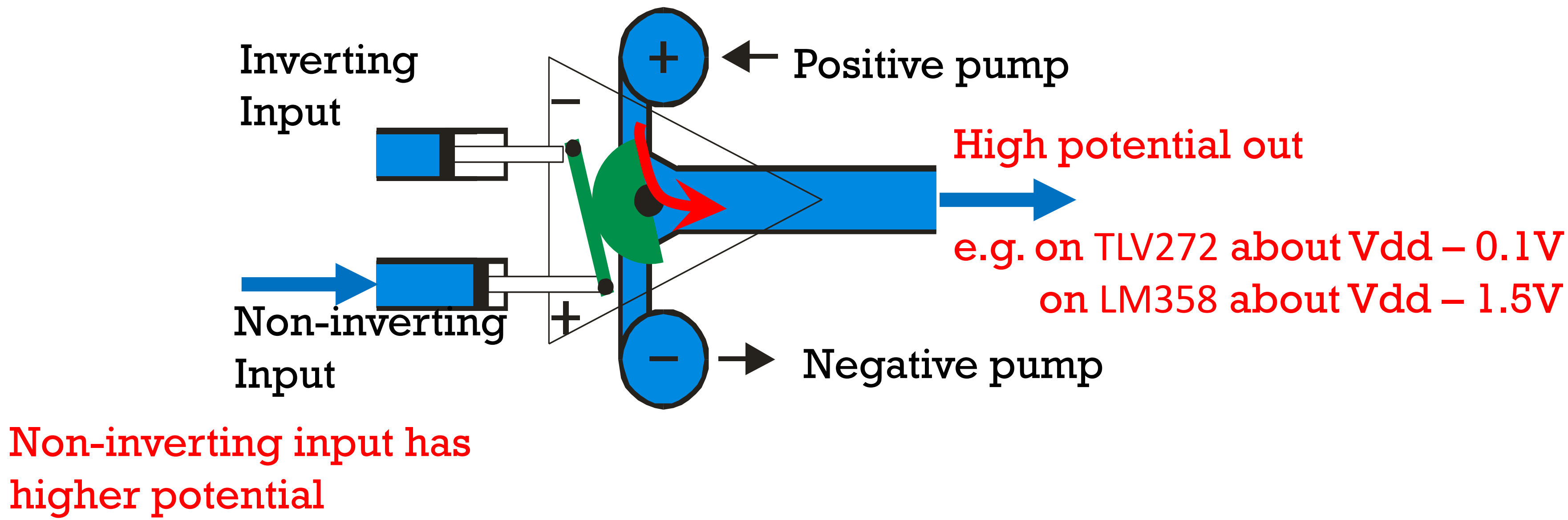
Inverting input
has higher
potential



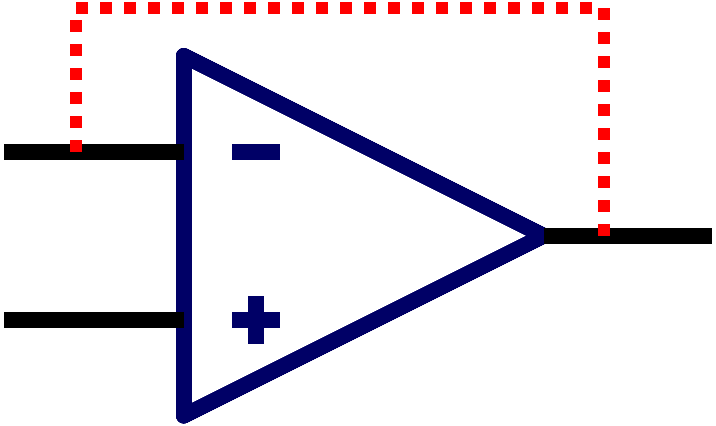
Low potential output

e.g., on TLV272 about 0.1V
on LM358 about 0.02V

Opamp water analogy



Negative Feedback



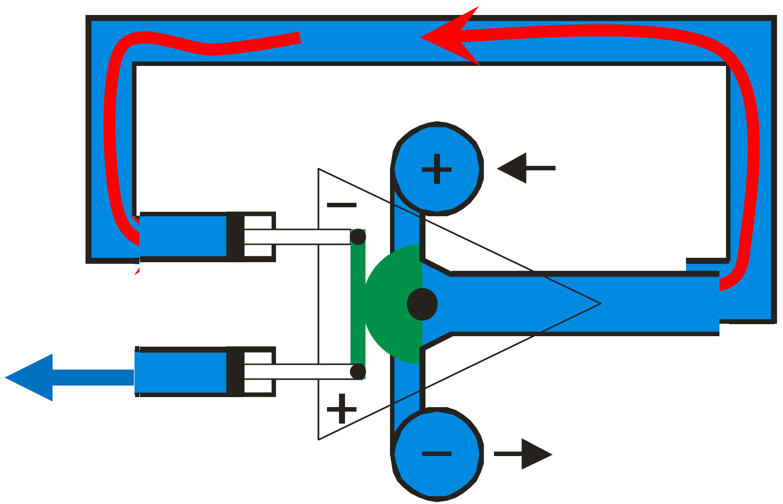
(-)V		(+)V	output?
higher	>	lower	Output drives (-) lower
lower	<	higher	Output drives (-) higher

Until (-) and (+) match

2nd golden rule: Inputs are at same potential (V)

If in negative feedback

Opamp water analogy (Negative Feedback)



If non-inverting input has higher potential than the inverting input while in negative feedback, output drives the non-inverting input lower until it's equal.

Q1: What will happen if non-inverting input has lower potential?

Golden Rules of Ideal Opamps

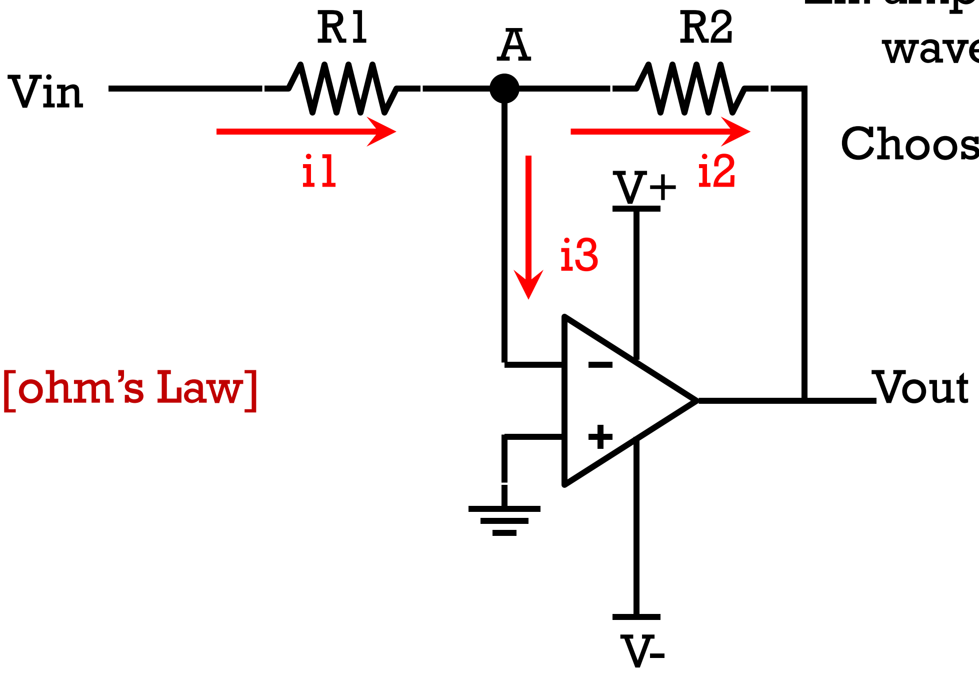
1. Inputs draw no current
2. Inputs are at the same potential
   (when in feedback)

The Inverting Amplifier

$i1 - i2 - i3 = 0$
[Kirchoff's current law]

$i3 = 0$
[1st rule]

$i1 = i2$



Ex: amplify sine wave10x

Choose: R2 =10K
R1 = 1 K

$$\frac{Vin - V_A}{R1} = \frac{V_A - Vout}{R2}$$ [ohm's Law]

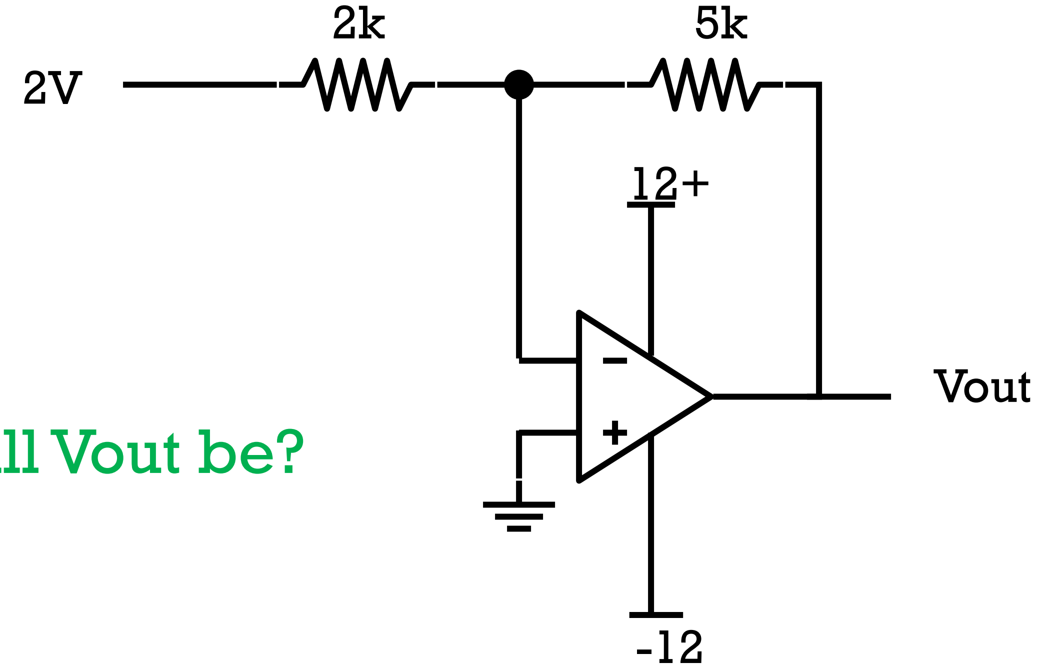
$$\boxed{\frac{Vout}{Vin} = \underbrace{\frac{-R2}{R1}}_{Gain}}$$

You can choose resistors to get desired gain

1st Golden rule: Inputs draw no current
2nd golden rule: Inputs are at same potential (V)

Q2: What can we say about the voltage at point A?

The Inverting Amplifier Exercise

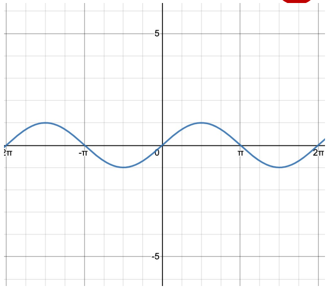


Q3: What will Vout be?

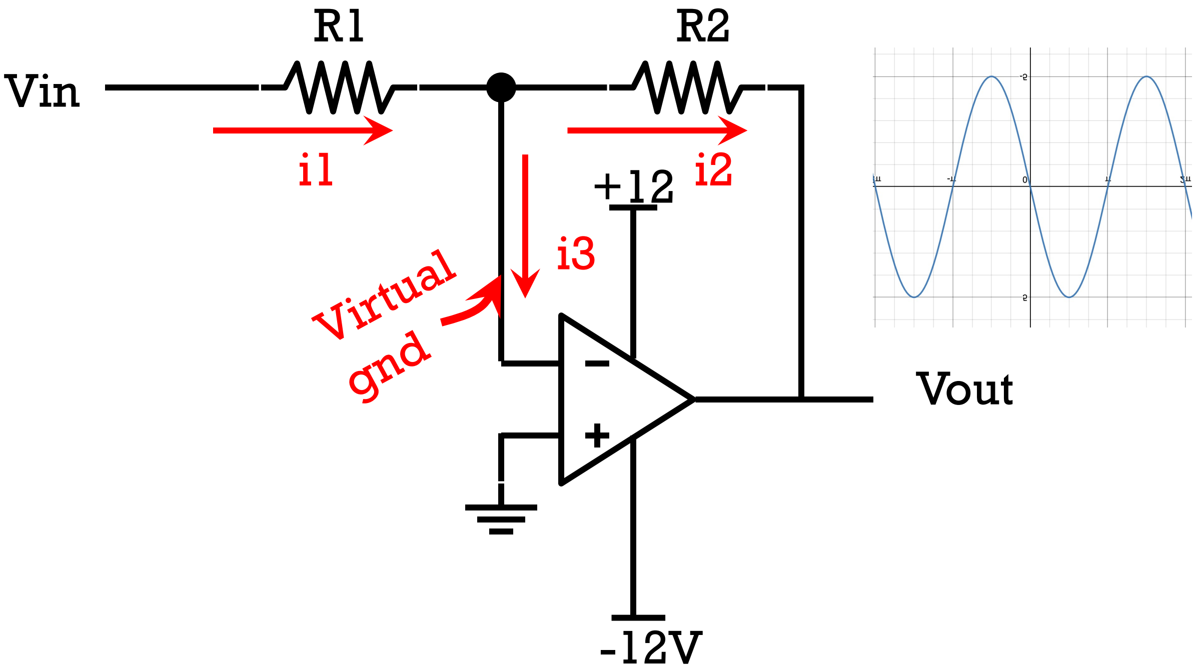
The Inverting Amplifier Exercise

$i3 = 0$

$i1 = i2$



$$\frac{Vin - 0}{R1} = \frac{0 - Vout}{R2}$$

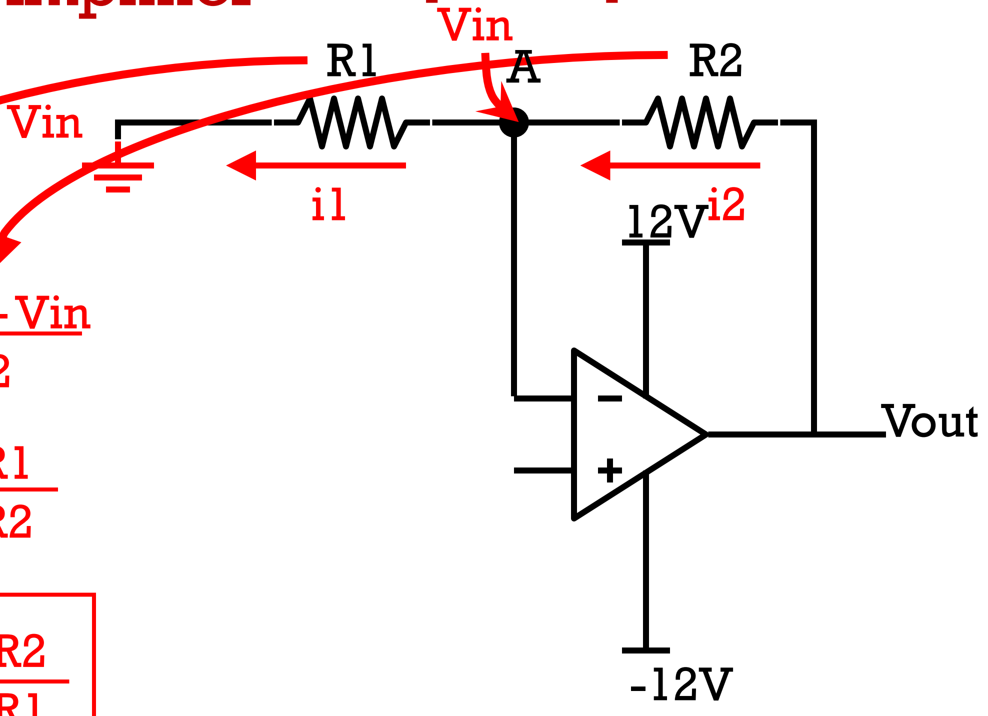


We have a sine wave peaks at +1V and -1V at Vin

Q4 In Chat: What values of R1 and R2 will give us an output flipped sine wave peaks at +5V and -5V?

The Non-Inverting Amplifier

[2^{nd} rule]



[1^{st} rule] $i1 = i2$

[ohm's Law] $$\frac{Vin - 0}{R1} = \frac{Vout - Vin}{R2}$$

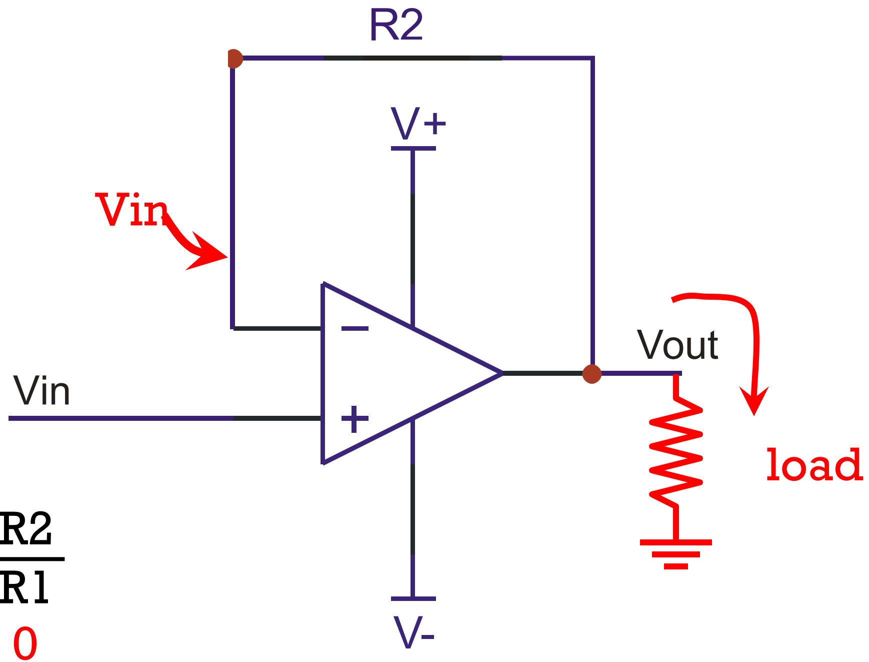
$$\frac{Vin}{Vout - Vin} = \frac{R1}{R2}$$

$$\boxed{\frac{Vout}{Vin} = 1 + \frac{R2}{R1}}$$

Gain

The Non-Inverting Buffer

Vout = Vin



$$\frac{Vout}{Vin} = 1 + \frac{R2}{R1}$$

$$= 1 + \frac{0}{\infty}$$

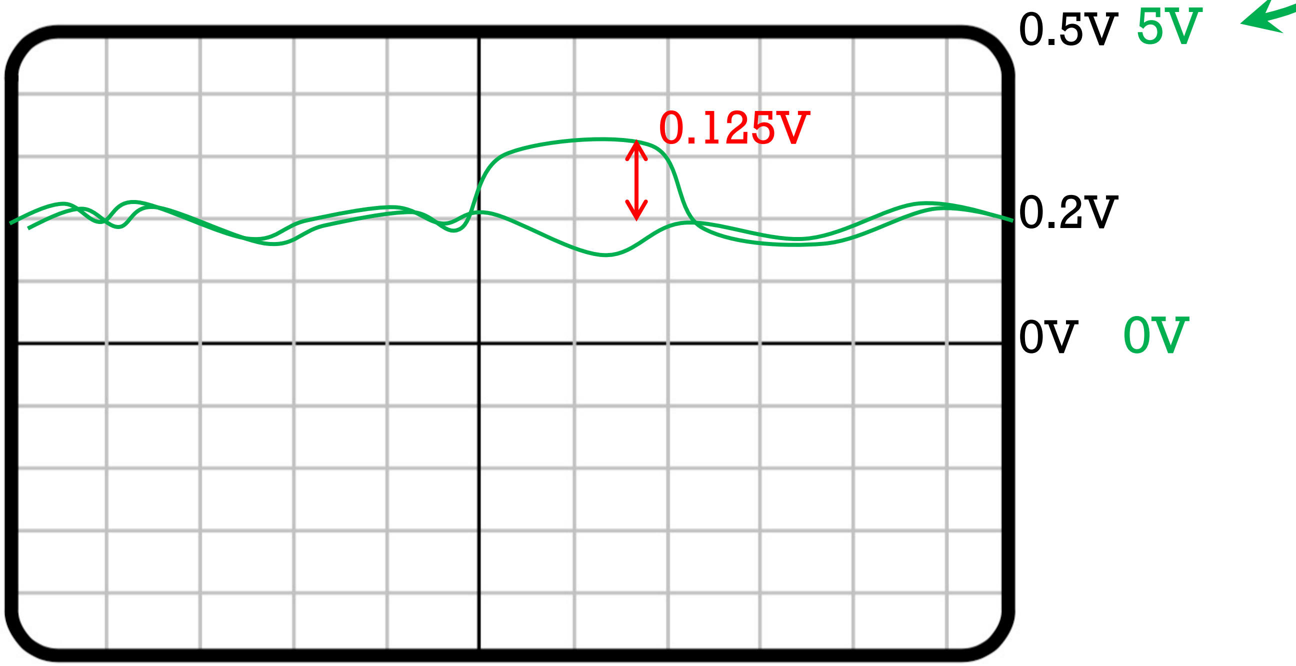
Isolate input, from current stand point
(no (very little) current will flow from accessing Vin)

2nd golden rule: Inputs are at same potential (V)

Amplifying photosensor signals.

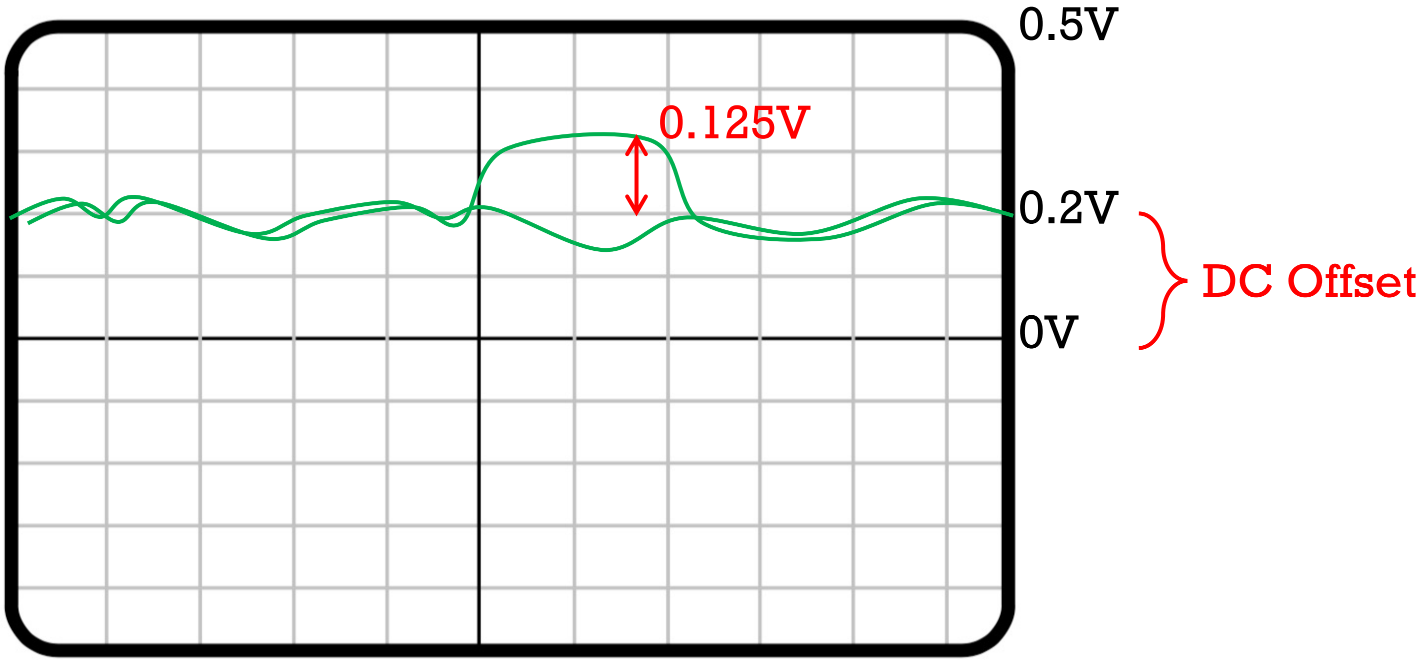
Q5: Draw and hold what the scope trace looks like when the signal is amplified 40x and the scale of the scope changes to 5V.

Scope output of a photo sensor circuit (unamplified)

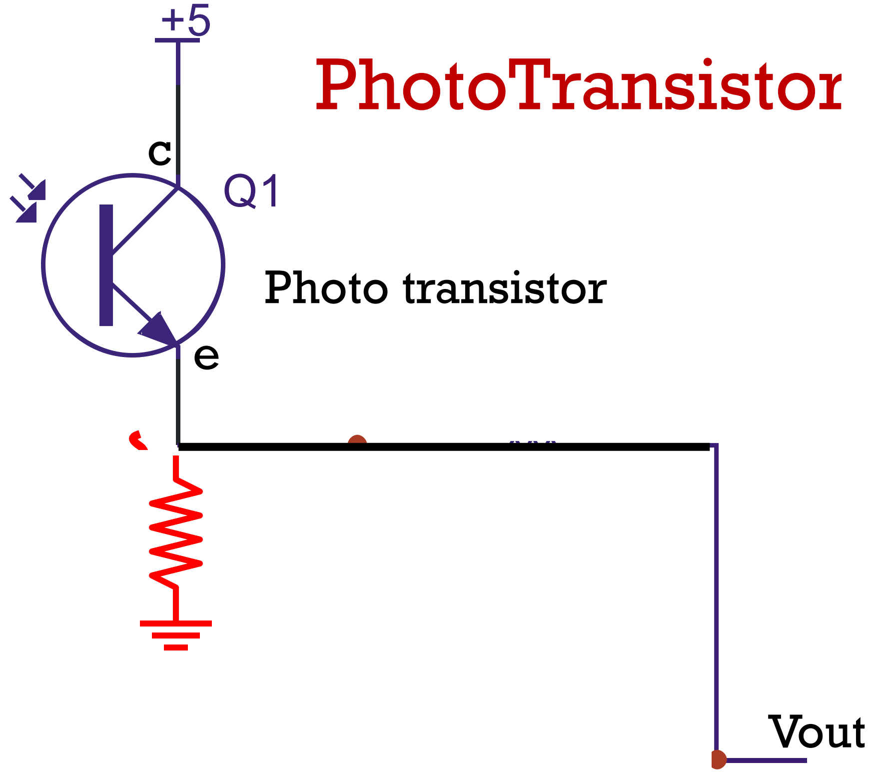


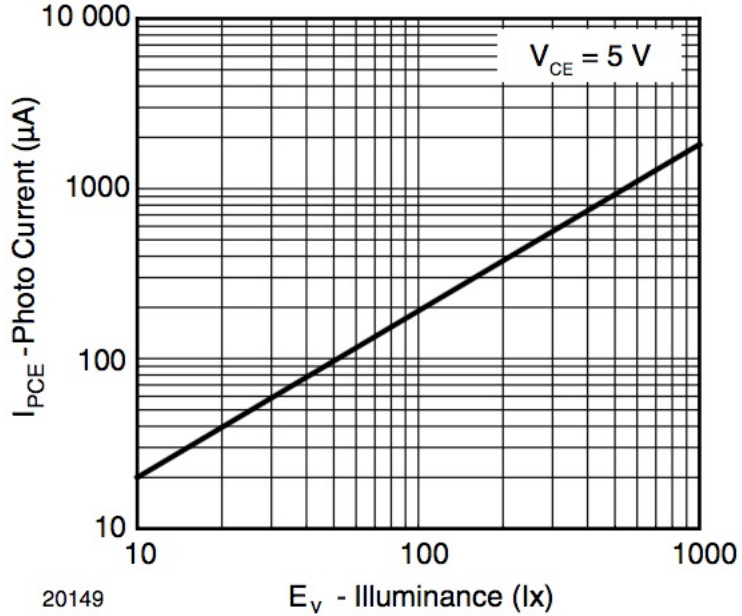
Signal Conditioning: Amplification

Must consider Gain and DC Offset



PhotoTransistor

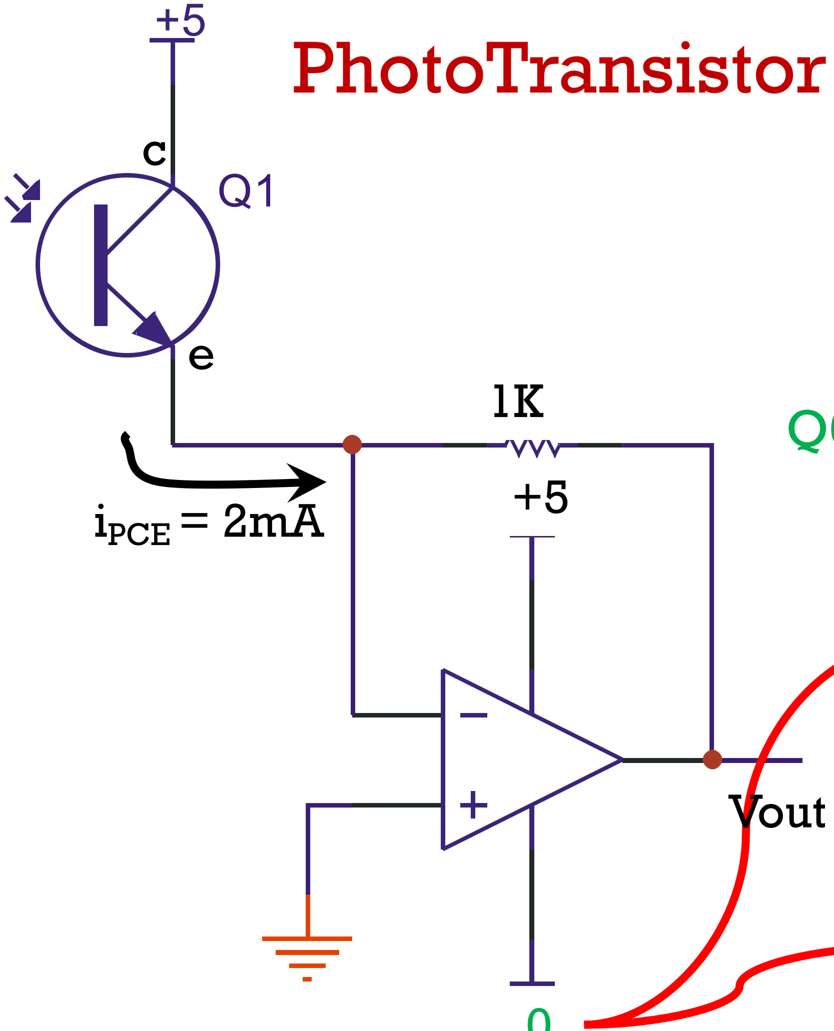




Vout = $- i_{PCE}$ R2 [ohm's Law]

Transresistive configuration

PhotoTransistor



$$V_{out} = -\, i_{PCE}\, R2$$

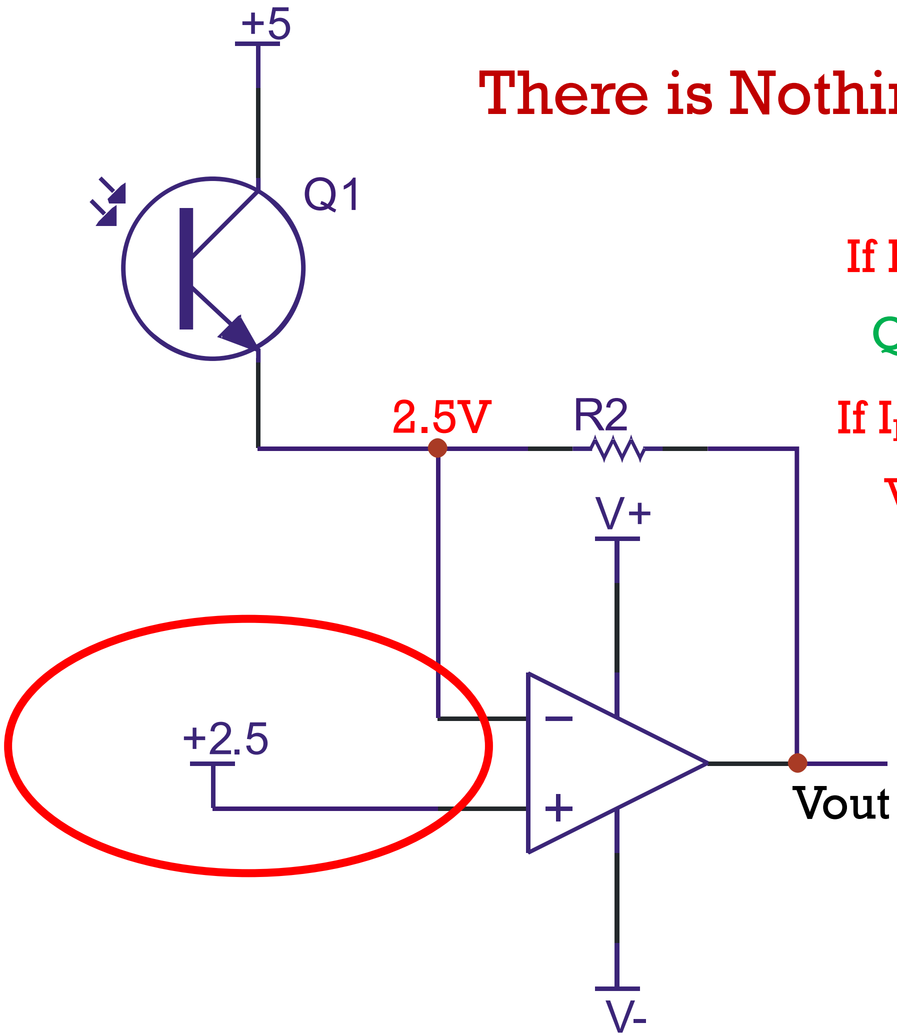
Q6: What is Vout in this case?

With dual supply:
Vout = -2V

Q7: How about now?

With single supply
Should be Vout = -2, but
Vout = 0

There is Nothing Magic about Ground

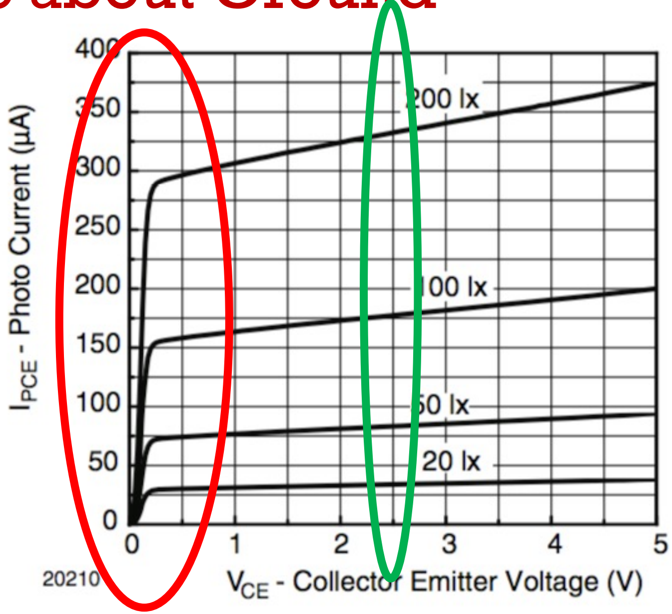


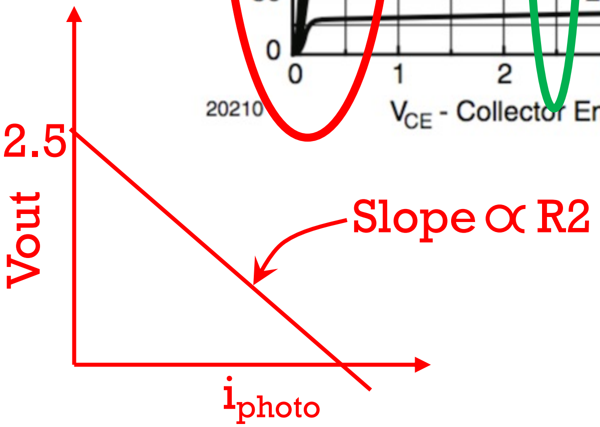
If $I_{PCE} = 0$ mA

Q8: Vout = ?

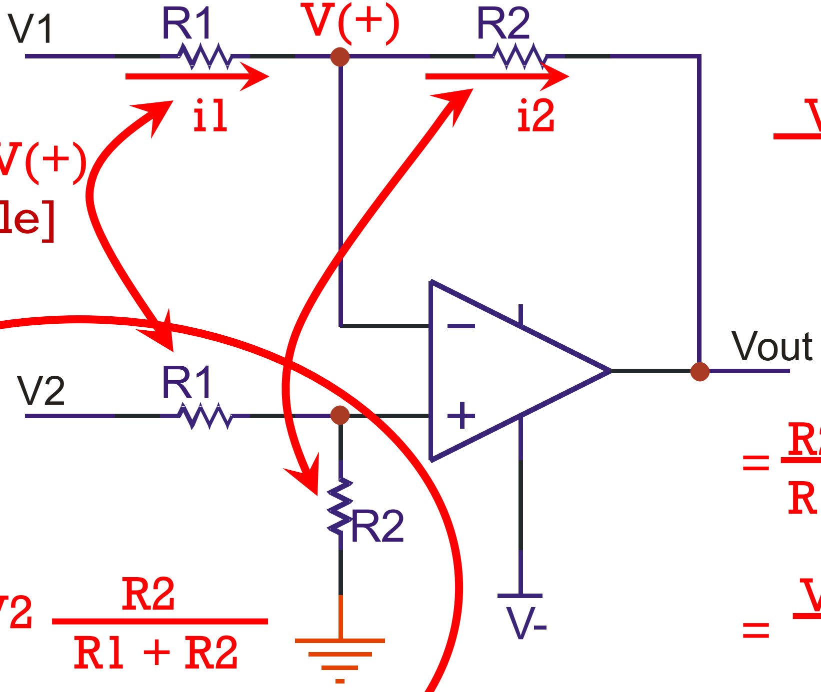
If $I_{PCE} > 0$

Vout < 2.5V





Another Application



V(-) = V(+)

[2^{nd} Rule]

$$V(+) = V2\frac{R2}{R1 + R2}$$

[voltage divider]

$$i1 = i2$$

[1^{st} Rule]

$$\frac{V1 - V(+)}{R1} = \frac{V(+) - Vout}{R2} \quad \text{[Ohm's law]}$$

$$-Vout = \frac{R2}{R1}(V1 - V(+)) - V(+)$$

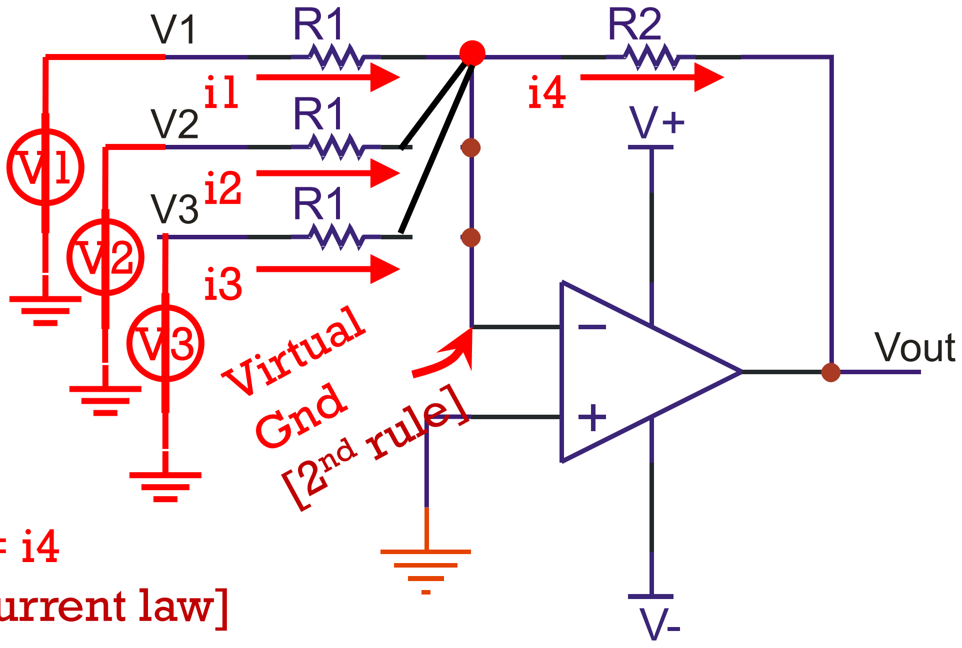
$$= \frac{R2}{R1}\left(V1 - V2\frac{R2}{R1 + R2}\right) - V2\frac{R2}{R1 + R2}$$

$$= \frac{V1R1 + V1R2 - V2R2 - V2R1}{R1 + R2}$$

$$\boxed{(V2 - V1)\frac{R2}{R1} = Vout}$$

(Subtraction) with multiplier

Still Another Application



$i1 + i2 + i3 = i4$

[Kirchoff's current law]

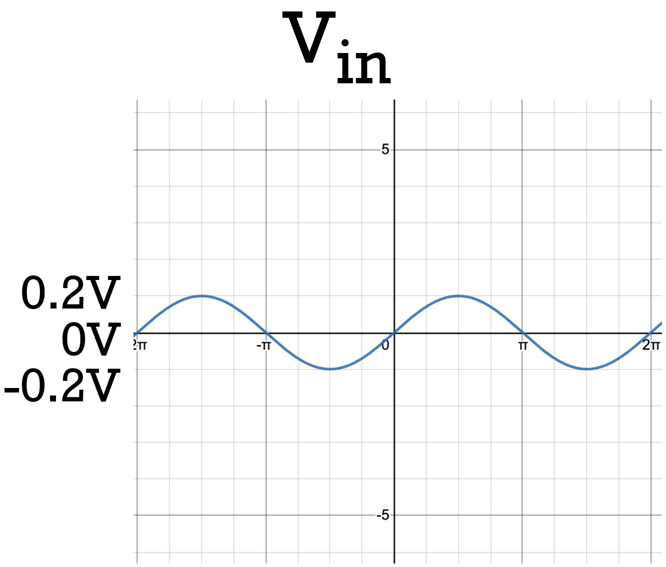
$$\frac{V1}{R1} + \frac{V2}{R1} + \frac{V3}{R1} = \frac{-Vout}{R2}$$

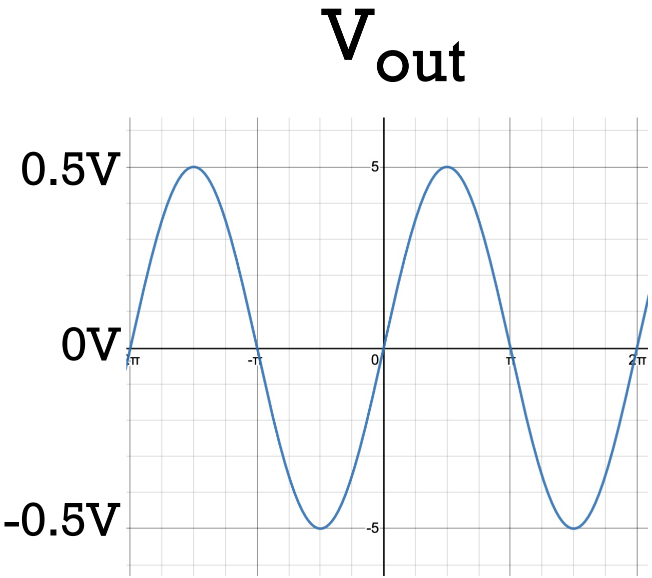
[Ohm's law]

$$\boxed{(V1 + V2 + V3)\,\frac{R2}{R1} = -Vout}$$

(Addition) with multiplier

Example problem:



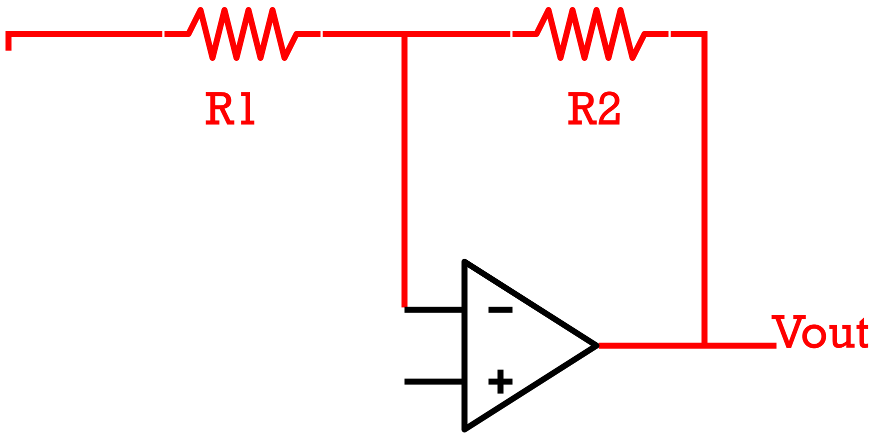


Vin is a sine wave with amplitude 200mV (peak-to-peak)

Q9: Design circuit such that:

Vout has a 1V (pk-pk) sine wave centered at 0V with same phase

Example Solution

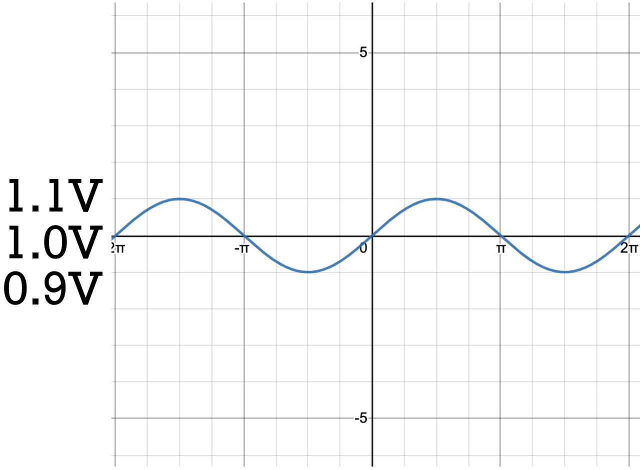


Op Amp Design Methodology

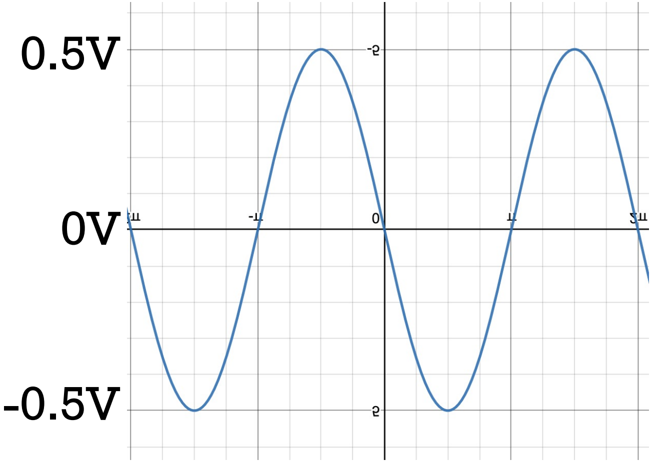
1. Decide inverting or not
2. Decide gain (Vout/Vin)
3. Find offset
    1. Create current equation for node at V(-)
    2. Use ohm's law to introduce R1, R2, Vout, Vin and Voffset
    3. Pick a known value for Vin and Vout.
    4. Solve for unknown Voffset.

Example with offset

V_{in}



V_{out}



V_{in} sine wave with amplitude 200mV (pk-pk) centered on 1V

V_{out} should be a 1V (pk-pk) sine centered at 0V, 180° out of phase

Example with offset Solution

1) Inverting

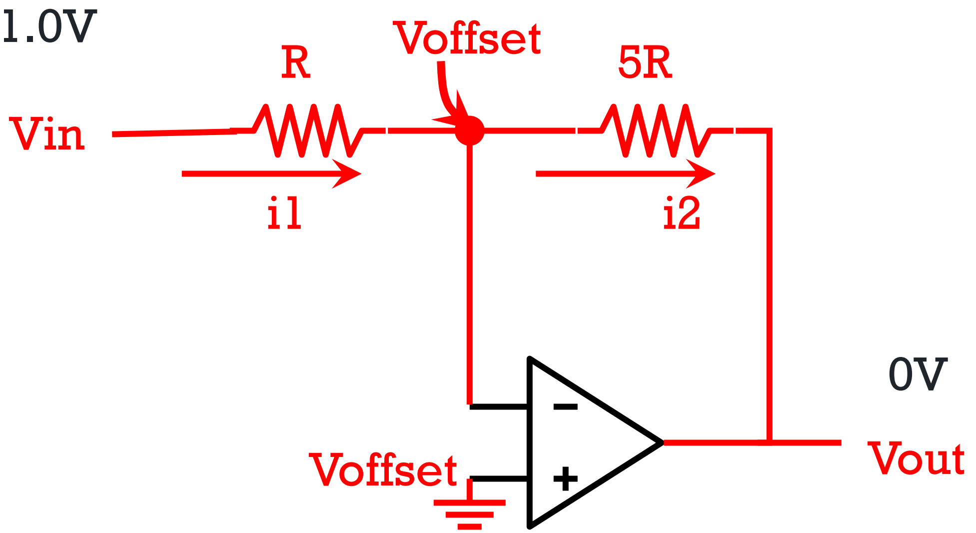
2) Gain = -R2/R1
We want gain of 5

Consider a point, when
Vout = 0V -> Vin = 1.0V

i1 = i2

$$i2 = \frac{V_{offset} - (0V)}{5R}$$

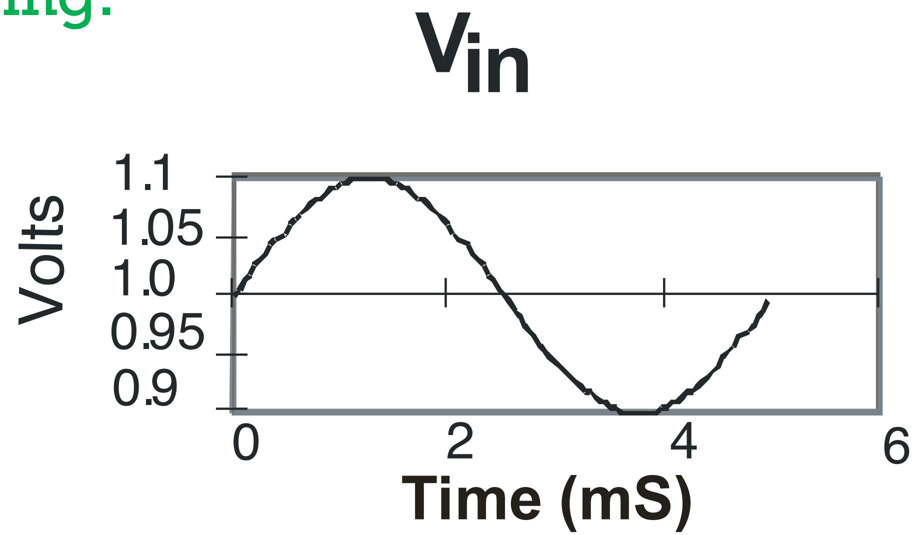
$$i1 = \frac{1.0 - V_{offset}}{R}$$



Q10: In Chat, solve for Voffset

Q11: What can you do if your input signal has an unknown DC offset. It might vary between 1V and 5V?

Practice problem: Design an opamp circuit for the following:



Vin sine wave with amplitude 200mV (pk-pk) centered on 1V

Vout should be a 5V (pk-pk) sine centered on 2.5V and in phase.

This problem and others will be covered in Recitation

02

Basic Coding Practice
Subroutines
Psuedo code

Structured programming to avoid spaghetti code…

There is a tendency in mechatronics students to write code that becomes difficult to follow… adding more and more onto a piece of code that initially worked for some parts, but as it grew in complexity became very hard to follow and harder to debug or add onto because it wasn’t modular, or well thought out in terms of architecture at the beginning.

Structured programming to avoid spaghetti code…

Mechatronics students tend to write difficult to follow code…

Modular code will be easier to follow, debug, and add onto as it gets larger.

Thinking about the code structure beforehand helps

Subroutines

```
#include "teensy_general.h"  // includes the resources included in the teensy_general.h file
#include "t_usb.h"

int main(){
  m_usb_init();                               // init USB for print statements
  set(TCCR3B,CS30); set(TCCR3B,CS32); // Start timer3 with /1024 prescaler

  m_usb_tx_string("first hit ");
  while( bit_is_set(PINC,7)) ;// wait while PC7 is high
  while(!bit_is_set(PINC,7)) ;// wait until PC7 stops being low
  m_usb_tx_uint(TCNT3); m_usb_tx_char(10); m_usb_tx_char(13);

  m_usb_tx_string("second hit ");
  while( bit_is_set(PINC,7)) ;// wait while PC7 is high
  while(!bit_is_set(PINC,7)) ;// wait until PC7 stops being low
  m_usb_tx_uint(TCNT3); m_usb_tx_char(10); m_usb_tx_char(13);

  m_usb_tx_string("first hit ");
  while( bit_is_set(PINC,7)) ;// wait while PC7 is high
  while(!bit_is_set(PINC,7)) ;// wait until PC7 stops being low
  m_usb_tx_uint(TCNT3); m_usb_tx_char(10); m_usb_tx_char(13);

  m_usb_tx_string("second hit ");
  while( bit_is_set(PINC,7)) ;// wait while PC7 is high
  while(!bit_is_set(PINC,7)) ;// wait until PC7 stops being low
  m_usb_tx_uint(TCNT3); m_usb_tx_char(10); m_usb_tx_char(13);
```

Subroutines

```
#include "teensy_general.h"  // includes the resources included in the teensy_general.h file
#include "t_usb.h"

int main(){
  m_usb_init();                             // init USB for print statements
  set(TCCR3B,CS30); set(TCCR3B,CS32); // Start timer3 with /1024 prescaler

  while(1) {
    m_usb_tx_string("first hit ");
    while( bit_is_set(PINC,7)) ;// wait while PC7 is high
    while(!bit_is_set(PINC,7)) ;// wait until PC7 stops being low
    m_usb_tx_uint(TCNT3); m_usb_tx_char(10); m_usb_tx_char(13);

    m_usb_tx_string("second hit ");
    while( bit_is_set(PINC,7)) ;// wait while PC7 is high
    while(!bit_is_set(PINC,7)) ;// wait until PC7 stops being low
    m_usb_tx_uint(TCNT3); m_usb_tx_char(10); m_usb_tx_char(13);
  }
}
```

Subroutines

```
#include "teensy_general.h"  // includes the resources included in the teensy_general.h file
#include "t_usb.h"

#define PRINTNUM(x) m_usb_tx_uint(x); m_usb_tx_char(10); m_usb_tx_char(13)


int main(){
  m_usb_init();                          // init USB for print statements
  set(TCCR3B,CS30); set(TCCR3B,CS32); // Start timer3 with /1024 prescaler

  while(1) {
    m_usb_tx_string("first hit ");
    while( bit_is_set(PINC,7)) ;// wait while PC7 is high
    while(!bit_is_set(PINC,7)) ;// wait until PC7 stops being low
    PRINTNUM(TCNT3);

    m_usb_tx_string("second hit ");
    while( bit_is_set(PINC,7)) ;// wait while PC7 is high
    while(!bit_is_set(PINC,7)) ;// wait until PC7 stops being low
    PRINTNUM(TCNT3);
  }
}
```

Subroutines

```
#include "teensy_general.h"  // includes the resources included in the teensy_general.h file
#include "t_usb.h"

#define PRINTNUM(x) m_usb_tx_uint(x); m_usb_tx_char(10); m_usb_tx_char(13)

void waitforpress() {
  while( bit_is_set(PINC,7)) ;// wait while PC7 is high
  while(!bit_is_set(PINC,7)) ;// wait until PC7 stops being low
  PRINTNUM(TCNT3);
}

int main(){
  m_usb_init();                             // init USB for print statements
  set(TCCR3B,CS30); set(TCCR3B,CS32); // Start timer3 with /1024 prescaler

  while(1) {
    m_usb_tx_string("first hit ");
    waitforpress();

    m_usb_tx_string("second hit ");
    waitforpress();
  }
}
```

Subroutines

```
#include "teensy_general.h"  // includes the resources included in the teensy_general.h file
#include "t_usb.h"

#define PRINTNUM(x) m_usb_tx_uint(x); m_usb_tx_char(10); m_usb_tx_char(13)

int waitForC7RisingEdge() {
  while( bit_is_set(PINC,7)) ;// wait while PC7 is high
  while(!bit_is_set(PINC,7)) ;// wait until PC7 stops being low
}

void waitforpress() {
  waitForC7RisingEdge();
  PRINTNUM(TCNT3);
}

int main(){
  m_usb_init();                          // init USB for print statements
  set(TCCR3B,CS30); set(TCCR3B,CS32); // Start timer3 with /1024 prescaler

  while(1) {
    m_usb_tx_string("first hit ");
    waitforpress();

    m_usb_tx_string("second hit ");
    waitforpress();
```

Probably too much,
Don't need to make this
a subroutine

Subroutines – proper use benefits

- Makes the code clearer to understand
- Remove duplication (simpler is better)
- Makes the code more modular
  - Can reuse
  - Can change elements more easily
  - Breaking down into components makes it easier to build
- May effect speed, (sometimes slightly slower, but usually negligible)

Pseudocode

- Artificial and informal language to help programmers develop algorithms.
- Indent for dependencies or loops

Example calculating class average:

```
Set grade counter to one
While grade counter is less than 115
     Input the next grade
     Add the grade into the total
Set the class average to the total divided by 114
```

This is NOT Pseudo-Code

```
for (signal = GetSignal(); signal == (new_signal = GetSignal());)
    ;

for ( this_time = GetTime(), i=0; i < SAMPLE_SIZE; i++ )
{
    signal = new_signal;
    last_time = this_time;
    while ( signal == ( new_signal = GetSignal() ) )
      ;
    this_time = GetTime();
    delta_Ts[i] = this_time - last_time;
}

avgtime = average(delta_Ts);
if (avgtime > NOISE) openDoor();
```

For this class, Pseudocode is detailed yet readable

Wait for signal to change states

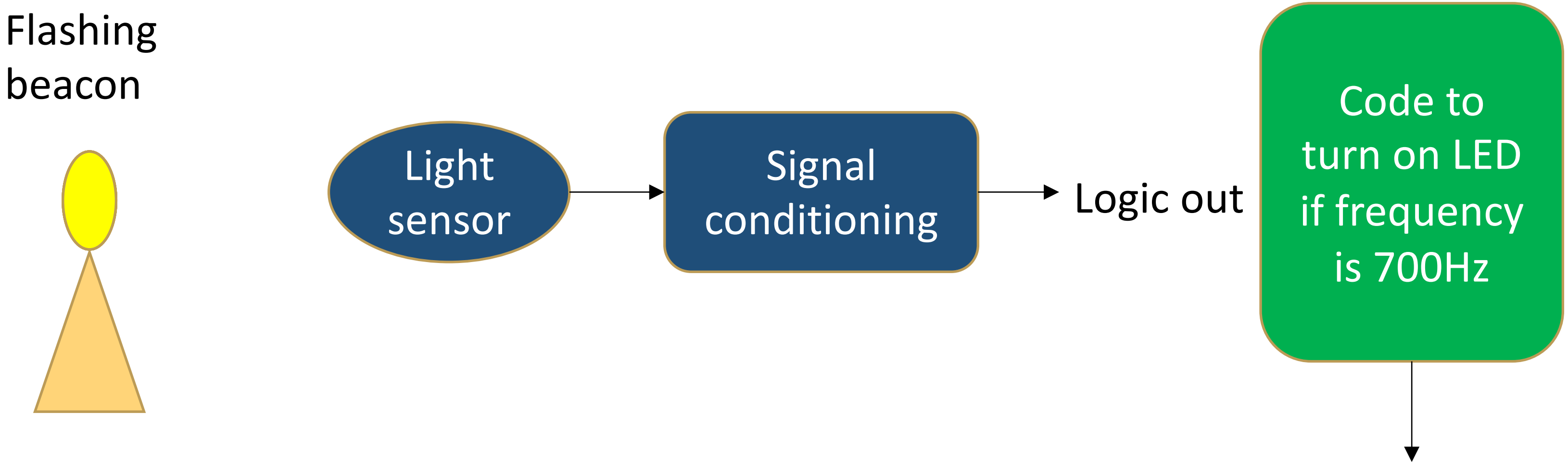
Repeat SAMPLE_SIZE times
    wait for signal to change states
    save the delta-T into an array of samples

Calculate average time
If signal is not noise, open door

Pseudocode Becomes Comments

```
/*  Wait for signal to change states  */
for (signal = GetSignal(); signal == (new_signal = GetSignal());)
    ;
/*  Repeat SAMPLE_SIZE times  */
for ( this_time = GetTime(), i=0; i < SAMPLE_SIZE; i++ )
{
    signal = new_signal;
    last_time = this_time;
    /*  wait for signal to change states  */
    while ( signal == ( new_signal = GetSignal() ) )
      ;
    this_time = GetTime();
    /*  save the delta-T into an array of samples  */
    ticks[i] = this_time - last_time;
}
/* Calculate average time */
avgtime = average(times);
if (avgtime > NOISE) openDoor();   // If signal not noise open door
```

Pseudocode Practice (prelude to Lab 2.3.3)



Q12: Write pseudo code (somewhere between 5 and 10 lines) to turn on LED if the detected flashing light frequency is 200Hz.

If in class you will talk to neighbor, if on zoom, you'll be paired in breakout rooms.

Summary

- OpAmp Golden Rules:
  1. Inputs draw no current
  2. Inputs are at the same potential (in feedback)
- Signal conditioning includes gain and offset.
- Alternating signals of specific frequencies among ambient signals easier if you AC couple (add a capacitor in line).
- Sub

Answer in CHAT

Answer how you feel about each topic below with:

1. I don't understand this topic at all
2. I understand a little, but will need help
3. I understand half, but expect to get the rest later
4. I understand completely already

A. Op Amps

B. Timer Input Capture

C. Phototransistors

Debugging Code

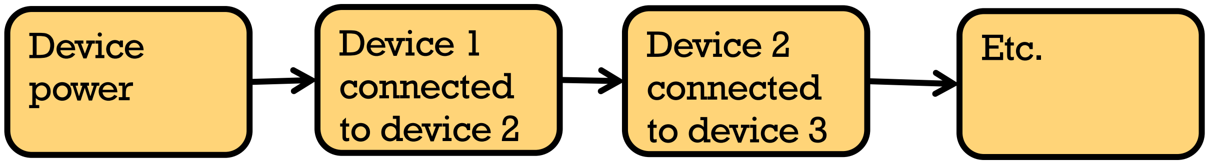
- If you feel you need help with C or programming in general.

  `https://www.onlinegdb.com/`

- Has an online debugger that lets you track the flow of code.
- Note: ATmega specific things won't work (PORTD, Timers etc.)
- Looking into setting it up so we can actually debug ATmega code (e.g. put includes and fake stubs so things can compile).

Debugging Electronics

- Hardware and Software piece - need to check both
- Break your system apart into pieces
  - viewed as linked via inputs and outputs
  - Ideally a chain of parts with 1 output from one part going to 1 input in the next part.



- Check known voltage at each part starting with power and ground.