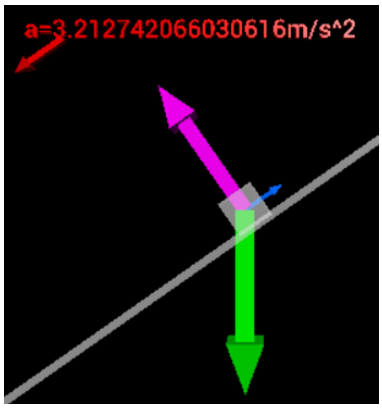
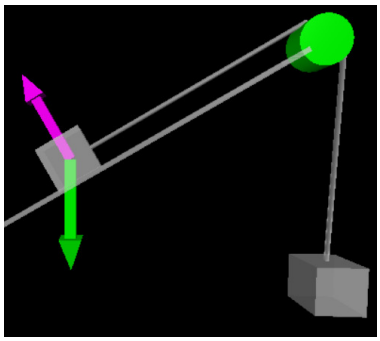
The goal of this assignment is to create simulate with one block on an incline attached to a hanging mass.

1) The code on the following two pages simulates a block on an incline. The code computes the frictional force depending on the angle (above or below the critical angle). For angles above the critical angle the block slides and acceleration magnitude is displayed. A screen shot of this code's output is shown at right. Note: ideally you would take the time to figure out how to display fewer sig figs on the output. I have some ideas for that somewhere but was in a rush to get this file up at the time of printing.



2) The code fragment on starting on page 4 gives you all the pieces to draw a block, incline, pulley, hanging mass and some strings. A screen shot of this code's output is shown at right. First copy and paste this code fragment into a new program.

3) Animate this simulation. Don't forget to add in force arrows for tension, friction, and the weight of the hanging mass. Also display the acceleration with a text box (or print statement).



BlockOnIncline

```
GlowScript 2.7 VPython
from visual import *
#angle of block in DEGREES
angle_degrees = 35
#convert angle to RADIANS
theta = radians(angle_degrees)

#static and kinetic coefficients
#note: code will only produce reasonable results when mu_k < mu_s
mu_s = 0.5
mu_k = 0.3

#initialize mass and magnitude of freefall acceleration
g = 9.8
mass = 1.0

#create a scale factor which is useful for drawing arrows
#by adjusting the scale factor, you can draw the arrows larger or smaller
#this is useful if the arrows are really huge or really small
scale_arrow = 0.5

#draw a horizontal board
incline = box(pos = vec(0, 0, 0),
            length = 20,
            width = 2,
            height = 0.2,
            color = vec(1, 1, 1),
            opacity = 0.5)
#draw block
block = box(length = 1,
            width = 1,
            height = 1,
            color = vec(1, 1, 1),
            opacity = 0.5)
#position block such that it rest on board 3/4 from the left end
block.pos = incline.pos + vec(incline.length/4, block.height/2, 0)
#create a parameter called block_vleocity to start the block from rest
block.velocity = vec(0, 0, 0)

#rotate the two objects about the z-axis (z-axis points out of page)
incline.rotate(angle = theta, axis = vec(0, 0, 1), origin = vec(0, 0, 0))
block.rotate(angle = theta, axis = vec(0, 0, 1), origin = vec(0, 0, 0))

#draw weight arrow
weight = vec(0, -mass*g, 0)
weight_arrow = arrow(pos = block.pos,
                     axis = scale_arrow*weight,
                     color = vec(0, 1, 0))
#draw normal force arrow
normal_mag = mass*g*cos(theta)
normal = normal_mag*vec(-sin(theta), cos(theta), 0)
normal_arrow = arrow(pos = block.pos,
                     axis = scale_arrow*normal,
                     color = vec(1, 0, 1))
```

```
#compute critical angle in RADIANS
theta_critical = atan(mu_s)

#if theta > theta_critical, slipping should occur...use mu_k
if theta > theta_critical:
    fric_mag = mu_k*normal_mag                      #compute magnitude of fric
    fric = fric_mag*vec(cos(theta), sin(theta), 0)  #make fric a vector
    fric_arrow = arrow(pos = block.pos,             #draw fric arrow
                       axis = scale_arrow*fric,
                       color = vec(0, 0.4, 1))
    F_net = normal + weight + fric                  #determine net force vector
    accel = F_net/mass                              #determine a (Newton's 2nd)
    accel_arrow = arrow(pos = vec(-5,5,0),          #draw accel arrow
                        axis = scale_arrow*accel,
                        color=vec(1, 0, 0))
    accel_display = mag(accel)
    accel_label = text(text='a='+accel_display+'m/s^2', #display accel
                        color=vec(1, 0, 0),
                        pos = vec(-6,5,0),
                        height = 0.5)
    t = 0
    dt = 0.01
    sim_speed = 1
    while abs(block.pos.x) < incline.length/2*cos(theta):
        rate(sim_speed/dt)                  #set the frames per second displayed
        block.velocity += accel*dt          #update block velocity
        block.pos += block.velocity*dt      #update block position
        weight_arrow.pos = block.pos        #keep arrows with block
        fric_arrow.pos = block.pos
        normal_arrow.pos = block.pos
        t += dt                             #increment the time
else:
    fric_mag = mass*g*sin(theta)                    #compute fric magnitudue
    fric = fric_mag*vec(cos(theta), sin(theta), 0)  #compute fric vector
    fric_arrow = arrow(pos = block.pos,             #draw fric arrow
                       axis = scale_arrow*fric,
                       color = vec(0, 0.4, 1))
    accel_label = text(text='a=0',                  #write "a=0" on the screen
                        color=vec(1, 0, 0),
                        pos = vec(-5,5,0),
                        height = 0.5)
```

BlockOnInclineWithHangingMass

```
from visual import *
#angle of block in DEGREES
angle_degrees = 30
#convert angle to RADIANS
theta = radians(angle_degrees)

#static and kinetic coefficients
#note: code will only produce reasonable results when mu_k < mu_s
mu_s = 0.5
mu_k = 0.3

#initialize mass and magnitude of freefall acceleration
g = 9.8
mass1 = 1.0
mass2 = 0.2

#create a scale factor which is useful for drawing arrows
#by adjusting the scale factor, you can draw the arrows larger or smaller
#this is useful if the arrows are really huge or really small
scale_arrow = 0.5

#draw a horizontal board
incline = box(pos = vec(0, 0, 0),
            length = 20,
            width = 2,
            height = 0.2,
            color = vec(1, 1, 1),
            opacity = 0.5)
#draw block
block1 = box(length = 2,
            width = 2,
            height = 2,
            color = vec(1, 1, 1),
            opacity = 0.5)
#position block such that it rests on middle of board
block1.pos = incline.pos + vec(0, 0.5*block1.height, 0)
#create a parameter called block_vleocity to start the block from rest
block1.velocity = vec(0, 0, 0)

#draw a cylinder that represents a massless pulley with negligible axle friction
pulley = cylinder(axis=vec(0,0,1), radius = 0.5*block1.width, color= vec(0,1,0))
#reposition pulley to over the edge of the board
pulley.pos += vec(pulley.radius + 0.5*incline.length, 0, -0.25*incline.width)

#draw string1 connecting block 1 to top edge of pulley
string1 = cylinder(radius = 0.1, color = vec(1, 1, 1), opacity=0.5)
string1.pos = pulley.pos + vec(0, pulley.radius, 0.5*mag(pulley.axis))
string1.axis = block1.pos + vec(0.5*block1.width, 0, 0) - string1.pos

#rotate the two objects about the z-axis (z-axis points out of page)
incline.rotate(angle = theta, axis = vec(0, 0, 1), origin = vec(0, 0, 0))
block1.rotate(angle = theta, axis = vec(0, 0, 1), origin = vec(0, 0, 0))
pulley.rotate(angle = theta, axis = vec(0, 0, 1), origin = vec(0, 0, 0))
string1.rotate(angle = theta, axis = vec(0, 0, 1), origin = vec(0, 0, 0))
```

```
#draw a hanging mass directly below the right edge of pulley
block2 = box(length = 2,
             width = 2,
             height = 2,
             color = vec(1, 1, 1),
             opacity = 0.5)
block2.pos = pulley.pos + vec(pulley.radius, - 0.5*incline.length, 0)

#draw string2 connecting block 2 to right edge of pulley
string2 = cylinder(radius = 0.1, color = vec(1, 1, 1), opacity=0.5)
string2.pos = pulley.pos + vec(pulley.radius, 0, 0.5*mag(pulley.axis))
string2.axis = block2.pos + vec(0, 0.5*block2.height, 0) - string2.pos

#draw weight arrow
weight = vec(0, -mass1*g, 0)
weight_arrow = arrow(pos = block1.pos,
                     axis = scale_arrow*weight,
                     color = vec(0, 1, 0))
#draw normal force arrow
normal_mag = mass1*g*cos(theta)
normal = normal_mag*vec(-sin(theta), cos(theta), 0)
normal_arrow = arrow(pos = block1.pos,
                     axis = scale_arrow*normal,
                     color = vec(1, 0, 1))
```

Your goal is to finish this animation and simulate every other project.
When completed, you should be able to input different masses to see which way the blocks move (if at all). In addition, make the code display the value of acceleration on screen. Have the simulation stop if block1 reaches either end of the incline.

First complete the code WITHOUT friction.
I would assume acceleration of block1 up the plane is positive.
Algebraically solve the force equations for acceleration.
Solve those same equations algebraically for tension.
Have the code compute acceleration and tension using your algebraic formula.
Use this acceleration to animate the sim and display the acceleration.
Don't forget to resize each string when the blocks move (update each axis vec).
SAVE A COPY OF YOUR CODE BEFORE TRYING TO ADD IN FRICTION

Check 1: Algebraically compute what value of hanging mass (`mass2`) causes no acceleration. Call this `mass2_crit`. Note: `mass2_crit` depends on the angle! Using theta = 30 degrees, verify your code does *not* accelerate when the input value for `mass2` is `mass2_crit`.
Check 2: Should this code work when the angle theta is negative? Does the simulation produce the expected results when you run it?
Check 3: Before running the code, take a guess. What if `mass2` is much larger than `mass1`? What should the acceleration be? Does your code produce this result?
Check 4: Before running the code, take a guess. What if `mass1` mass is much larger than `mass2`? What should the acceleration be? Using theta = 30 degrees, does your code produce the expected result?

Now try to add in friction
Depending on the angles and masses, the block might slide up, down, or not all!
To determine if block1 sliding or at rest:

1) Compute the MAGNITUDE of normal force
2) Determine the MAGNITUDE of max POSSIBLE friction.
3) Determine the MAGNITUDE of `Force_parallel = T - mass1*g*sin(theta)`
4) IF `Force_parallel > mag(max_POSSIBLE_friction)` block1 slides *up the incline*.
    a. Compute ACTUAL friction magnitude using `mu_k*mag(normal)`
    b. Compute acceleration using force equation
5) ELIF `Force_parallel < -mag(max_POSSIBLE_friction)` block1 slides *down the incline*.
    a. Compute ACTUAL friction magnitude using `mu_k*mag(normal)`
    b. Compute acceleration using force equation
6) ELSE `mag(Force_parallel) <= mag(max_POSSIBLE_friction)` block1 remains at rest.
    Notice bullet 6 is the ELSE to bullets 4 & 5 IF & ELIF.
    a. We know acceleration is zero.
    b. Compute the ACTUAL frictional force using force equations.
7) Use $\theta = 65°, \mu_s = 0.9, \mu_k = 0.4$ & $m_1 = 0.162$ kg. Try several cases for m_2: 0.08, 0.09, 0.10, 0.20, 0.21 & 0.22 kg. I think these values should produce the corresponding results: slide down, at rest, at rest & slide up. You can also do a computation on paper to confirm my suspicions.

Finally, see if your code fails upon putting in a negative angle.
Think: if you use a negative angle, is it possible for the block to slide away from the pulley?
Recall, we are calling "towards the pulley" the positive direction for block1.