Regis University – Physics 305A – Fall 2019
Lab 1: Computational Tools

This lab is a quick intro to some computational tools that we'll be using this semester. It's just some exercises to help you get a bit more confident with spreadsheets and Python so that they aren't obstacles when we start doing real physics experiments. Everyone should work on their own computer for these exercises (if possible), so that everyone completes these individually. However, you are welcome to discuss with your peers to get help. If there aren't enough functioning computers for everyone, try to ensure that everyone understands each step before moving on. In general, if you don't feel confident about this stuff, that probably means that you're the best candidate to be at the keyboard, because the goal for today is *get* confident!

Part A – Spreadsheet Basics

Sometimes in lab you'll want to quickly organize or plot data in a spreadsheet (most commonly, Microsoft Excel, Google Sheets, or LibreOffice Calc – all are roughly equivalent). Some of you already have lots of skills in these programs; others of you may never have significantly used them. If you haven't, don't worry, they're pretty easy! These are some basic exercises to teach you just a few crucial skills (there are far more things you can do in a spreadsheet than what is represented here).

Start whichever spreadsheet program you have access to. Open a new, blank spreadsheet. Enter the number "1" in cell A1 and "2" in cell A2. You can enter a value by clicking on the cell and typing.

Now we want

Part B – Computational Physics Activity 1: The Basics of A Python Computer Program

Part C – Computational Physics Activity 2: Simulating motion at constant velocity

Based on activities VP01 and VP02 in the instructor resources that accompany Ruth Chabay and Bruce Sherwood's Matter and Interactions *textbook.*

So far we've talked about tools for doing basic mathematical calculations and analyzing experimental data, but computers have an additional use in physics: solving physics problems by simulating them numerically. In this course, we will use a Python module called VPython which is designed to make simulating physics with 3D graphics very easy to program.

Unfortunately, the Python environment on Colab doesn't support the 3D graphics of VPython, so we'll have to use a different environment (no big deal – that's the point of different Python environments, to let you use the tools appropriate to the problem you want to tackle). The easiest way to run VPython for the purposes of this lab is in a browser at http://www.glowscript.org/.[1] However, there is one difference when running VPython on Glowscript compared to using more ordinary Python environments: the first line of the program must tell Glowscript what kind of code you'll be running, so in our case the first line should be `GlowScript 2.8 VPython`. When you create a new program on Glowscript, this line should be automatically added.

Please start by watching these three short videos (between 3 and 4 minutes each):

[1] It is also possible to run VPython locally on a computer without using the internet. On the lab laptops, you can do so by running the VIdle environment, found in the Start menu under Physics, but because they have an older version installed, you must import from `visual` rather than `vpython`. If you're feeling *really* adventurous and would like to install VPython on your own laptop, it is a slightly complicated process, with several options described at `https://vpython.org/`.

- http://vpython.org/video01.html
- http://vpython.org/video02.html
- http://vpython.org/video03.html

Each video ends with a "challenge" to you. Attempt each of these challenges before starting into the next video – don't move on until you get it working!

Then, please read this program:

```
from vpython import *

offset = vector(0,-3,0)
position_now = vector(0,0,0)
n = 0
while n < 5:
   sphere(pos=position_now, color=color.red)
   position_now = position_now + offset
   n = n + 1
```

Make a prediction of what the program will do, and write down that prediction before you run it. Did it do what you expected? If not, do you understand why it did something different?
Next, please do the same with the following program:

```
from vpython import *

ball = sphere(pos=vector(-5,0,0), radius=0.5, color=color.green)
block = box(pos=vector(-8,0,0), color=color.yellow)
velocity = vector(0.4, 0.6, 0)
delta_t = 0.1
t = 0

while t < 12:
  rate(100)
  ball.pos = ball.pos + velocity * delta_t
  t = t + delta_t
```

Again, make a prediction, check it by running the program, and explain any discrepancy.

Finally, here is the beginning of a program, which you can add on to. Currently, it draws a magenta sphere just in front of a green wall. The sphere (named "particle") starts at $\vec{r}_i = [-10,0,0]$ m. A velocity vector is defined so that $\vec{v} = [0.5,0,0]$ m/s. Make a prediction; how long should it take for the sphere to go to $\vec{r}_f = [10,0,0]$ m?

```
from vpython import *

wall = box(pos=vector(0,4,-0.5), length=20, height=10, width=0.1, color=color.green)
particle = sphere(pos=vector(-10,0,0), radius = 0.5, color=color.magenta)
velocity = vector(0.5,0,0)
delta_t = 0.1
t = 0
```

Starting with this program, add a `while` loop that moves the "particle" along the wall, from the left side to the right side. Have it stop when it gets to the right side; you can refer to the x component of the vector that represents the ball's position as `particle.pos.x` in the loop's condition expression.

Have your program print out the elapsed time after the completion of the loop. How long did it take? Does this match your prediction?

So, congratulations! You have just written a computer program to solve a physics problem by simulating it numerically. This is a powerful technique that is frequently used by scientists to model systems that are too complicated to understand with pencil-and-paper calculations alone.