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). Go through these steps, making sure you understand how to use them.

1. 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.

2. For this step, you should **not** type any more numbers. We want to continue this column of numbers, so that cell A3 contains the number "3", and so on, up to cell A30, which should contain the number "30". It would be tedious to type all of these numbers by hand; luckily, you can create them automatically in a spreadsheet. Click on cell A1 (containing the number "1") and hold the mouse button down while you drag down to also highlight cell A2. Both should be highlighted now. At the bottom right corner of this two-cell selection, there is a small square. Click on it and hold the mouse button down while you drag the small square downward until the border encompasses all cells from A1 to A30. Release the mouse button, and the cells should be filled as we desired.

3. Let's think of the A column as an x variable, and let's try to fill the B column with x^2 values (e.g., 1, 4, 9, 16, etc.). Select cell B1. To begin typing a formula, press the = button. Once you have entered the = sign, and the typing cursor is still blinking inside cell B1, click once on cell A1. The formula insde cell B1 should now read `=A1`. However, we want to *square* the value from A1, so continue typing the forumla until it reads `=A1^2` and then press enter. You should see that the cell now contains the value 1 as expected. Be sure you understand what you did in this step before moving on!

4. Select the newly-filled cell B1. Click the small square in the bottom right of the selection, and drag down to cell B30. You should now see all of the x^2 values corresponding to the values for x in column A. Notice how easy it is to apply a formula to a huge number of values!

5. Let's plot these values. Select *all* of the filled cells in both columns by clicking and dragging. Click on "Insert" near the top of the window. Find the correct button to insert a line chart. You should now see part of a parabola, curving upwards. Ensure you understand how to select data and plot it before moving on.

6. Explore your spreadsheet program, or ask another student how, until you figure out how to fit a straight line to your plot (it'll be a pretty terrible fit, since you are fitting a line to a parabola!).

These are the extreme basics of spreadsheets. You can create most types of formulas and most types of plots. You should **never** be calculating lots of things by hand in lab, and you should **never** be repeatedly typing formulas in a spreadsheet. **Always** find a way to do the tasks mostly automatically using appropriate computing features – if you don't know how, ask!

Do not move on to Part B or discard this spreadsheet until I check that you have completed it.

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

In this course, you will sometimes need to complete tasks in the Python programming language. Python is an extremely powerful, common, and easy to use programming language. I use it for all of my data analysis for my research, and it is used by nearly every major company. It's also used by artists, sports teams, musicians, medical professionals, and nearly every other type of profession. You can use it to study black holes, control a robot, make interactive art, or even just automate basic daily tasks. So it's worth knowing a little bit of it. It's also very useful for physics.

As we go through Parts B and C of this lab, feel free to try things and experiment. You can't break anything – we can always reset Python. A big part of programming is tinkering and trying things (and, of course, getting things wrong a whole bunch of times as a result!). I definitely see more error messages than successes when I work on Python code for my research; that's just part of figuring out how to program things!

You can run Python natively on your computer, but most of you probably don't have it installed, so we'll use a website that runs your code online for you. Go to `https://trinket.io/python3/01d155eeac`.

You should see a window with two sections. On the lefthand side is where you write your Python code. On the righthand side, the output from your code will be displayed.

Let's start with the basics. It is a very dumb, very old tradition to make your first program in a new language be one that simply prints the phrase, "Hello World!" to the screen. Try doing this in Python by typing `print("Hello World!")` in the left window. Click the play button at the top to run it; you should see Python print that phrase. How did work? Basically, `print()` is a built-in command in Python. It displays whatever is inside the parentheses on the screen. The quotation marks tell it to print the exact characters to the screen, rather than trying to do simplifying math on whatever you've typed.

See how it works differently without the quotes. Try replacing your command the two lines of code below (at the same time, one line after the other):

```
print("2+3")
print(2+3)
```

Run the code. What has happened? What have the two commands done differently?

Notice that Python runs your commands in order, from top to bottom. This is a fundamental idea in programming, so keep it in mind – it determines how you should structure your code.

We've discovered that we can print things to the screen and do arithmetic in Python. Sometimes we want to do more complex calculations. Python can use *variables*, much like algebraic variables. Variables store information. For examples, try running the following code:

```
a=12
print(a)
```

First we store the value in the variable `a`, then, when we print that variable, Python prints the information we stored. We can likewise do math with variables. Try running this code:

```
# words after a # symbol are comments, python ignores them
a=12
b=4**2 # the ** operator is Python's command for exponents
cow=a+b
print(cow)
```

Explain what this code has done in plain English.

When we do calculations, we might want to use more complex mathematical functions. Python lets you *import* more complicated, premade commads to help you with many tasks. (This is part of why Python is so useful: people have created commands that let you do all sorts of things with almost no effort.) Suppose that we want to take the square root of a number. The `math` contains all the basic math functions. At the beginning of our code, we need to import it. Try this:

```
from math import *
print(sqrt(4))
```

The first line basically says, "from the math module, import everything," and one of the command in that module is `sqrt()`. Once we import it, we can use it.

To conclude this part of the lab, write a set of code that stores the number 55 in a variable called `regis`. In the next **single** line of code, subtract 3 from `regis`, then cube that result, and then take the square root, and store that result in a variable `lacksinternet`. On the next line, print `lacksinternet`. Python should display a value of approximately 374.977. (Hint: You can use parentheses to control order of operations.) Show your working code to me before moving on.

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.

VPython needs to run in a slightly different environment than the regular Python. 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; if you have your own headphones, please use them!). Note that because we are running our code online, you will not need to save the file or anything like that; simply type your code and run it:

- 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:

[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/`. Alternately, if you want to run it online in a browser but hate the `glowscript.org` website, you can run it at `https://trinket.io/glowscript/c13fe6820a`

```
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.