Physics 411: Homework 2

1. **The semi-empirical mass formula revisited:** As we saw in last week's homework, the nuclear binding energy B of an atomic nucleus with atomic number Z and mass number A can be calculated to a good approximation using the semi-empirical mass formula:

$$B = a_1 A - a_2 A^{2/3} - a_3 \frac{Z^2}{A^{1/3}} - a_4 \frac{(A-2Z)^2}{A} + \frac{a_5}{A^{1/2}},$$

where, in units of millions of electron volts, the constants are $a_1 = 15.67$, $a_2 = 17.23$, $a_3 = 0.75$, $a_4 = 93.2$, and

$$a_5 = \begin{cases} 0 & \text{if } A \text{ is odd,} \\ 12.0 & \text{if } A \text{ and } Z \text{ are both even,} \\ -12.0 & \text{if } A \text{ is even and } Z \text{ is odd.} \end{cases}$$

Last week you wrote a program using this formula to calculate the binding energy B and binding energy per nucleon B/A for given A and Z.

(a) Starting from that program, modify it so that it takes as input just a single value of the atomic number Z and then goes through all values of A from $A = Z$ to $A = 3Z$, to find the one that has the largest binding energy per nucleon. This is the most stable nucleus with the given atomic number. (We expect the most stable nucleus to have A in the vicinity of $2Z$ because most nuclei have about as many neutrons as protons, so going through the values of A from Z to $3Z$ should be enough to find the most stable one.) Have your program print out the value of A for this most stable nucleus and the value of the binding energy per nucleon.

(b) Modify your program again so that, instead of taking Z as input, it runs through all values of Z from 1 to 100 and prints out the most stable value of A for each one, along with the corresponding binding energy per nucleon. At what value of Z does the maximum binding energy per nucleon occur? (The true answer, in real life, is $Z = 28$, which is nickel. You should find that the semi-empirical mass formula gets the answer roughly right, but not exactly.)

(c) Modify your program one final time to make a graph of the binding energy per nucleon for the most stable nucleus, as a function of Z for values of Z from 1 to 100.

☑ **For full credit** turn in a printout of your final program from part (c), your answer to the question in part (b), and your graph from part (c).

2. **Plotting experimental data:**

(a) In the on-line resources you will find a file called `sunspots.txt`, which contains the observed number of sunspots on the Sun for each month since January 1749. The file contains two columns of numbers, the first being the month and the second being the sunspot number. Write a program that reads in the data and makes a graph of sunspots as a function of time.

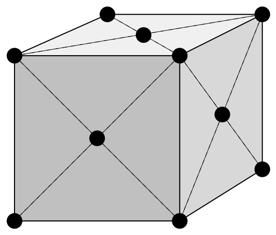
(b) The file `stm.txt` contains a grid of values from scanning tunneling microscope measurements of the (111) surface of silicon. A scanning tunneling microscope (STM) is a device that measures the surface of a solid at the atomic level by tracking a sharp tip over the surface and measuring quantum tunneling current as a function of position. The end result is a grid of values that represent the height of the surface and the file `stm.txt` contains just such a grid of values. Write a program that reads the data contained in the file and makes a density plot of the values. Use the various options and variants you have learned about to make a picture that shows the structure of the silicon surface clearly.

☑ **For full credit** turn in printouts of your two programs and printouts of your plots.

3. **Visualizing lattices:** Example 4.2 in the book gives a program that creates a computer visualization of a simple cubic lattice. Using that program as a starting point, or starting from scratch if you prefer, do the following:

   (a) A sodium chloride crystal has sodium and chlorine atoms arranged on a cubic lattice but the atoms alternate between sodium and chlorine, so that each sodium is surrounded by six chlorines and each chlorine is surrounded by six sodiums. Create a visualization of the sodium chloride lattice using two different colors to represent the two types of atoms. (If you print out the result in black-and-white, make sure to use colors that are clearly distinguishable.)

   (b) The face-centered cubic (fcc) lattice, which is the most common lattice in naturally occurring monatomic crystals, consists of a cubic lattice with atoms positioned not only at the corners of each cube but also at the center of each face, like this:



   Create a visualization of an fcc lattice with a single species of atom (such as occurs in metallic iron, for instance).

☑ **For full credit** turn in a printout of your program for part (b) and printouts of the two pictures you created.

Hint: Printing out 3D graphics can be a little tricky. The simplest way to do it is to run the program and then take a screenshot of the window containing the graphics. On a PC running Windows hold down ALT and then press the "Print Screen" button to take a screenshot of the current active window. On a Mac hold down Command and Shift and press the number 4, then press the space bar, then click on a window to take a screenshot of that window. Once you have your screenshot you can paste it into a document and print it out.

4. **Visualization of the solar system:** The innermost six planets of our solar system revolve around the Sun in roughly circular orbits that all lie approximately in the same (ecliptic) plane. Here are some basic parameters:

Object	Radius of object (km)	Radius of orbit (millions of km)	Period of orbit (days)
Mercury	2440	57.9	88.0
Venus	6052	108.2	224.7
Earth	6371	149.6	365.3
Mars	3386	227.9	687.0
Jupiter	69173	778.5	4331.6
Saturn	57316	1433.4	10759.2

   (a) Write down equations for the coordinates x, y of a planet in the plane of the ecliptic at time t, assuming that it lies on the x-axis at $t = 0$ and travels in a circular orbit with radius R.

   (b) Using the facilities provided by the `visual` package, create an animation of the solar system that shows the following:

      i. The Sun and planets as spheres in their appropriate positions and with sizes proportional to their actual sizes. Because the radii of the planets are tiny compared to the distances between them, it will be hard to see them clearly if you display them with their correct proportionate sizes. Instead, therefore, represent the planets by spheres with radii c_1 times larger than the proportionate values and choose a value for c_1 that makes the planets visible. You'll also need to find a good radius for the Sun. Choose any value that gives a clear visualization. (It doesn't work to scale the radius of the Sun by the same factor you use for the planets, because it'll come out looking way too large. So just use whatever works.) For added realism, you may also want to make your spheres different colors. For instance, Earth could be blue and the Sun could be yellow.

      ii. The motion of the planets as they move around the Sun (by making the spheres of the planets move). In the interests of alleviating boredom, construct your program so that time in the animation runs a factor of c_2 faster than actual time. Find a value of c_2 that makes the motion of the orbits easily visible but not unreasonably fast. Use the `rate` function to make your animation run smoothly.

☑ **For full credit** turn in a copy of your program and a snapshot showing the animation it produces.

5. **Recursion:** A useful feature of user-defined functions is *recursion*, the ability of a function to call itself. For example, consider the following definition of the factorial $n!$ of a positive integer n:

$$n! = \begin{cases} 1 & \text{if } n = 1, \\ n \times (n-1)! & \text{if } n > 1. \end{cases}$$

This constitutes a complete definition of the factorial which allows us to calculate the value of $n!$ for any positive integer. We can employ this definition directly to create a Python function for factorials, like this:

```
def factorial(n):
    if n==1:
        return 1
    else:
        return n*factorial(n-1)
```

Note how, if n is not equal to 1, the function *calls itself* to calculate the factorial of $n-1$. This is recursion. If we now say "`print(factorial(5))`" the computer will correctly print the answer 120.

(a) The Catalan numbers C_n are a sequence of integers $1, 1, 2, 5, 14, 42, 132\ldots$ that play an important role in quantum mechanics and the theory of disordered systems. (They were central to Eugene Wigner's proof of the so-called semicircle law.) They can be defined as follows:

$$C_n = \begin{cases} 1 & \text{if } n = 0, \\ \dfrac{4n-2}{n+1}\, C_{n-1} & \text{if } n > 0. \end{cases}$$

Write a Python function, using recursion, that calculates C_n. Use your function to calculate and print C_{100}.

(b) Euclid showed that the greatest common divisor $g(m,n)$ of two nonnegative integers m and n satisfies

$$g(m,n) = \begin{cases} m & \text{if } n = 0, \\ g(n, m \bmod n) & \text{if } n > 0. \end{cases}$$

Write a Python function `g(m,n)` that employs recursion to calculate the greatest common divisor of m and n using this formula. Use your function to calculate and print the greatest common divisor of 108 and 192.

☑ **For full credit** turn in copies of your two programs and the answers you calculated using them.