PHYS605

HOMEWORK 2

DUE FRIDAY FEB 17.

READING:

- **Exploring Raspberry Pi:**
  - Be familiar with Chapter 2: “Controlling the Raspberry Pi”, and “Shutdown and Reboot”.
  - Chapter 3 is great for those who want to do their own Sysadmin work on the RPi. Much of this chapter is optional.
  - Be familiar with Chapter 3, “Using Git Version Control”, p99 – 108. You will need this to get course code and to store your own code, see below.
  - Read Chapter 4 up to page 138. A lot of this should be review.
- **Practical Electronics:**
  - Read 2.20, 2.21, 2.23 (pay special attention to 2.23.13), 2.24 (pay special attention to 2.24.18).

PROGRAMMING

1. Make sure you have the following applications installed on your own computer. Detailed instruction are available on MyCourses “Installing Python on your own System” (Email me if you run into any issues.)
   a. Install atom.io on your own computer.
   b. Install Git on your own computer.
   c. Install “Anaconda” Python on your computer.
2. Learn about Jupyter Notebooks from the “Notebooks Getting Started” document in our course repository on GitHub:
   https://github.com/mholtrop/Phys605/blob/master/Python/Getting_Started/Notebooks_Getting_Started.ipynb
   a. Create your own notebook on your own system. Try it out a bit.
   b. If you are completely unfamiliar with Python, search for a “Python Tutorial” and work through it, or work through the official Python tutorial.
3. You now have some choices, depending on the level or challenge you are looking for. Note that it is acceptable to not get all the way to an answer on these tasks, but do document the headway you made and **submit your code**, with documentation. You can submit a notebook or commented .py file.
   It is fully expected with this assignment that you use all available resources to accomplish these tasks, so feel free to search for solutions. Note that you do yourself a favor to understand the solutions you find. So see how far you get with one of the following assignments.

I. A MORSE CODE TRANSLATOR (EASIEST CHALLENGE)

Here you will write some code that can take any character string and then return a list that gives instructions for how to send this string in Morse code. You could then use this code to blink out

the Morse code with an LED. Comment all your code! Provide references for any sources you used.

1. Step one. Write a Python program that prints any string you give it to the terminal.
    a. You can either get this string by asking the user to input it. Search “python input string”, and learn how. Note this is slightly different between Python2 and Python3.
    b. The other way to get a string into your program is a command line argument. Fully parsing a command line is tricky, and there is a library called “argparse” to help you with it. Simply getting a string is not so bad, see:
        i. http://www.pythonforbeginners.com/system/python-sys-argv
        ii. https://www.tutorialspoint.com/python3/python_command_line_arguments.htm
    c. You will need to figure out how to run your program from the command line!
2. Next, you need to translate this string. The way to do that in Python is to create a dictionary for the translation. Lookup “Python dictionary” and find a tutorial you like.
    a. Create the dictionary so that morse_code[‘A’] returns the string “.- " and morse_code[‘B’] returns “-… ” etc. For space, return two spaces.
    b. You need to now “read” your string one character at a time, and then use the dictionary to print the morse code out as dots and dashes.
3. If you get here, great. Next you want to change the step 2b to give you a long string of all the dots, dashes and spaces. You now want to write a function that reads this string and for each dot it would want to blink an LED a short time, for each dash a long time, and for each space it would leave the LED off for the same length as a dash. Since you don’t have the LED setup at home, you can write the code and use print statements to check that it does the correct thing.

II. BIG NUMBERS AND PRIME NUMBERS (MEDIUM CHALLENGE)

It is fun to use Python with numbers, since it is quite good at it. In this task, you are to learn about Python lists and how you can manipulate them, and then use your skill to look at some big numbers and write a program that finds prime numbers.

1. Python’s ability to do things with lists is very useful. Learn about Python lists in a tutorial if you do not know about them. Specifically, learn about “Python list comprehension”. In addition to straight up Python, the Numpy package has a lot of tools to work with numeric lists, but we will leave that for later. Check each of the following steps by printing the result.
    a. Make a list with numbers from 0 to 20, sequentially. (Use range())
    b. Make a list in reversed order, from 20 to 0. (Invert the previous list)
    c. Make a list of square numbers, from 0 to 20 squared, using list comprehension to do this.
    d. Make a list of 2^N with N from 0 to 20, using list comprehension to do this.
2. Python dictionaries are distinctly different from lists. They are very useful in different situations.
    a. Create a dictionary that allows you to look up the 2^N numbers and find N, that is the integer log2. So my_log2[16] would return 4, etc. Go up to N=20.
3. Create a simple function that checks if a number is a prime number. Call this is_prime(N) and it will return true if N is a prime. Your code should be fairly efficient.

4. An efficient way to get such a list is a “prime number sieve”, or **sieve** of Eratosthenes (250s BCE). Write a function “get_primes(N)”, that returns a **list** of the N lowest primes. Comment this code and note the source of any information you used.
    a. Use the is_prime(N) function from before to check that your sieve did indeed only find prime numbers.
5. Use your prime number system to create a list of all the prime numbers below 1000. Then filter this list, using list comprehension, to find all the “twin prime” numbers. Twin primes are prime numbers that have a twin that is 2 larger or smaller.
    a. Print your twin primes as pairs (x,x+2)
6. Really, really big numbers.
    a. Python has no problem with really big integers. Have Python print the number 2 to the power 2 to the power 2 … 5 times (2**2**2**2**2)
    b. Put that big number into a variable, “bignum”.
    c. Turn “bignum” into a string with str() and store the result.
    d. Now count (with a program!) how many times the digit 2 appears in this big number.

III. THE BIG CHALLENGE (HARDEST PROBLEM): PERFECT NUMBERS

Google just announced a stock buy-back for an amount that is a perfect number in the 8 billion range. Write a function that finds all the perfect numbers, at least up to 10 billion, but if you are ambitious, up to 2^{100}. Note that you will need an efficient algorithm[†]. Comment any information sources you used.

7. Write a function that returns a list of proper divisors. You can find low perfect numbers by simply scanning, trying every number between 1 and 10000 does not take much time. To do so, you need a function that returns the “proper divisors” of an integer. Your function can return a list, or it can be a generator function.
8. Use the proper divisor generator function in another function that finds the perfect numbers under 10000.
9. Clearly, you want something a lot faster to go up to 2^{33} or so. There is a method found by Euclid in 300 BC to find even perfect numbers. Euler later proved that indeed all even perfect numbers are found using Euclid’s method. Google the method. Both Wikipedia and the Encyclopedia Britannica describe the method.
10. So to find perfect numbers using Euclid’s method you need a way to find Mersenne primes. The easy thing to do is to write a prime number checker, is_prime(N), that returns true if N is a prime. One efficient way to get such a list is a “prime number sieve”, or **sieve** of Eratosthenes (250s BCE), however for more than 2^{20} primes you need too much memory. Write a function “is_prime(N)”, that returns a **true** if N is a prime. You want to be efficient here, else it will take forever to check your primes.
11. You can now implement a perfect number search algorithm using Euclid’s method. Use it to find all the perfect numbers up to the one Google used for their stock buy back.

[†] On my laptop, it took 0.25 sec to go up to 2^{100}.

12. To get to even higher perfect numbers, you need a better prime number finder. Look up the Rabin Miller algorithm, which tells you if a number is a “probably prime”, with high probability. Use it to extend your previous code.