1. **Interpreter** (10 pts.)
   Open the Python interpreter. Give a short (approx. 1 sentence) explanation of the output of the following statements when they are executed in this order:
   
   (a) $2 \times 3$
   
   (b) "Hello, world!"
   
   (c) 'py' + 'thon'
   
   (d) $2 \times '3'$
   
   (e) 'a'
   
   (f) $'3' + 1$
   
   (g) $1 + '3'$
   
   (h) a
   
   (i) a = 3
   
   a

2. **Scripts** (10 pts.)
   Copy the commands from problem 1 parts (a)-(e) to a script. Save it as `script1.py` then run it (try `python script1.py`). What happens? Copy `script1.py` into `script1new.py` and use the `print` function to show the output of these commands.

3. **Booleans** (10 pts.)
   Open the Python interpreter again. Give a short (approx. 1 sentence) explanation of the following statements:
   
   (a) `'A' == 'A'`
   
   (b) $1 == True$
   
   (c) False + True
   
   (d) True / False
   
   (e) $2 == True$
   
   (f) a = 2
   
   if a:
       print "a"
   
   (g) $(3 == 1) \times 3$
   
   (h) $(3 == 3) \times 4 + 3 == 1$
   
   (i) $3**5 >= 4**4$

4. **Integers and floats** (10 pts.)
   Use the parameters and functions in the `sys` module to learn about the limitations of integers and floating point numbers. Briefly explain the output of the following statements in interactive mode:
   
   (a) $5 / 3$
   
   (b) $5.0 / 3$
(c) $4.0 ** (1 / 2)$
(d) $4 ** (1.0/2)$ (compare with above)
(e) $5 \ % \ 3$
(f) $5.2 \ % \ 3$
(g) $2014 ** 200$
(h) $2014 ** 200.0$ (compare with above)
(i) $1.0 + 1.0e10 - 1.0e10$
(j) $1.0 + 1.0e20 - 1.0e20$

5. **Variables** (10 pts.)
Write a script in the file `script2.py` which uses the variable `name` to hold a string with your name. Assuming your name is First Last, have the script print *Hello, First Last!* when you run it (obviously, you won’t get credit for code that uses `print "Hello, First Last!"`).

6. **Type casting** (10 pts.)
Very often, one wants to ”cast” variables of a certain type into another type. Suppose we have $x = "123"$ but we want $x$ to be an integer. We would just use `int(x)` to get it to be an integer. More generally, whatever type we want to convert $x$ into can be obtained by `desiredtype(x)`. Briefly explain the output of the following statements in interactive mode:

(a) `int(123.9)`
(b) `float('-123.9')`
(c) `int(float('-123.9'))`
(d) `str(1.23)`
(e) `complex('2.5+3e-4j')`
(f) `complex(0+1j)`
(g) `complex(0+j)`
(h) `bool('')` (those are two consecutive apostrophes, no space)
(i) `bool('False')`
(j) `float(None)`

7. **Range** (10 pts.)
As discussed in class, when you type `range(5)` in the interpreter, it returns `[0, 1, 2, 3, 4]`.

(a) What is the type of `range(5)`? We’ll learn more about this later.
(b) Look up the documentation of the `range()` function. If you wanted to use this function to output 7 even numbers starting at 10, what command would you use?
(c) What happens when you type `2*range(5)`. Why?

8. **Basic for and while loops** (12 pts.)

(a) Write a script in `forLoop.py` that prints out the first 10 perfect squares starting at 1. Someone else who looks at your code should easily be able to modify your code to print out however many square numbers they want.
(b) Write a script in `whileLoop.py` that prints out the first \( n \) prime numbers (starting with 2), where \( n \) is the smallest integer such that the sum of the first \( n \) prime numbers is greater than 75. Print out the sum at the end in the format `sum: 91` if the sum were for example, 91.

Someone else who looks at your code should easily be able to modify the minimum sum. It is acceptable to use part of the prime number example code from the lecture.

9. **Finding start codon in messenger RNA** (18 pts.)

In `mRNA.py` you’ll find an mRNA sequence (a string containing some combination of the letters A, C, G, and U). Find the first occurrence of the contiguous sub-sequence AUG, the start codon. Print out the position of the A in that first AUG. Indexing starts at 0.

For example, given this piece of an mRNA sequence:

```
UGCCAUGACUGACUAACUGCAACUGGUAC
```

the output of your script should be in the format `AUG starts in position 4`.

Hints: We’ll learn faster methods of searching strings later, but it is possible to do this problem only using the tools discussed in the first lecture. Your code should work if the sequence in your script were replaced with a non-empty sequence of a different length. Be careful- what if you have a sequence with no occurrences or multiple occurrences of AUG?

**Instructions for submitting the homework**

Before submitting, make sure your code runs on the corn machines. You should submit a compressed/zippered folder to your Drop Box on CourseWork containing the following files:

- `writeup.pdf`
- `script1.py`
- `script1new.py`
- `script2.py`
- `forLoop.py`
- `whileLoop.py`
- `mRNA.py`