This assignment is worth 0 points. You should be able to understand and complete these problems before starting the first assignment. Items marked 0 bonus points give additional information beyond what is required for this course.

This document is available at http://stanford.edu/~danfrank/cme193/homeworks/hwk-0.pdf.

1. **Python Interpreter**
   You can run the Python interpreter at the command-line on your computer. This allows you to type commands interactively. To see what this looks like, an online interpreter is available at:

   http://shell.appspot.com/.

   For this question, you have to get Python installed and running on your computer. Instructions are on the course web site. Please come to office hours if you have difficulty with this question.

   What are the outputs of the following command sequences in the interpreter?:

   (a) (0 points)
   ```
   >>> import math
   >>> print math.e
   ```

   (b) (0 points)
   ```
   >>> x = 'py' * 4
   >>> x
   ```

   (c) (0 points)
   ```
   >>> print undeclaredvariable
   ```

   For questions (2) and (3), using the Python interpreter is recommended but not required.

   For questions (4), (5), and (6), we have not covered all of the syntax for defining functions (that will be in the second lecture). We have also not covered how to write and execute Python scripts. If you are comfortable with this, please feel free to go ahead and implement the code. Otherwise, just treat the code as pseudocode.

2. **Arithmetic**
   State to what numeric value the following statements will evaluate in Python. Assume that the variable `x` has a numeric value of `10.0`. Remember, that `**` is the exponent function. `3 ** 2` $= 3^2 = 9$.

   (a) (0 points)
   ```
   11 * 2 - 4
   ```

   (b) (0 points)
   ```
   x + 2 ** 3 - 17 / 2
   ```

   (c) (0 points)
   ```
   x + 2 ** 3 - 17.0 / 2
   ```

   (d) (0 bonus points)
   ```
   1 if x / 5 > 6 else 2
   ```
   Note: this is called a ternary conditional operator.

3. **Logic**

   State whether the following statements are `True` or `False` as it would be evaluated in Python (i.e., how it was described in lecture). Assume that the variable `x` has a boolean value of `True`. No need for explanation, except for the bonus question.

   (a) (0 points)

   ```
   2 == 2 and 1 > 0
   ```

   (b) (0 points)

   ```
   ('py' * 3 + 'thon' == 'pypypython') and (1 < 2 < 3)
   ```

   (c) (0 points)

   ```
   0 is True or not x
   ```

   (d) (0 points)

   ```
   1 > 2 or x
   ```

   (e) (0 bonus points)

   ```
   'hello' < 'jello'
   ```

   Why is this true or false?

4. **Functions and flow**
   For each of the following Python scripts, state what gets printed.

   (a) (0 points)

   ```
   def func_a():
     a = 3
     b = a * 3
     if b > 8:
       b += 3
     return a + b

   print func_a()
   ```

   (b) (0 points)

   ```
   def func_b(x, k):
     i = 0
     while x > 1:
       x = x / k
       i = i + 1
     return i

   print func_b(10.0, 3)
   ```

(c) (0 points)

```
def func_c(var = 2):
    if var > 2:
        return var
    return 1

print func_c(14) + func_c()
```

(d) (0 points)

```
y = 10
def func_d(x, y):
    return x * y

print y + func_d(2, 3)
```

(e) (0 bonus points)

```
def func_e(x=1, y=2):
    if x == y:
        return func_e(3, 0)
    return x - y

print func_e(5, 5) + func_e() + func_e(y=10, x=11)
```

5. **Sequence products**

The code below computes the product of every other number between the parameter `start` and 20. For example, mult(15) would return $15*17*19 = 4845$ and mult(14) would return $14*16*18*20 = 80640$. You will implement the `errorcheck()` functions in part (a).

```
def errorcheck(start):
    return # part (a)

def mult(start):
    if errorcheck(start):
        return 'ERROR!'

    value = 1
    while (start <= 20):
        value = value * start
        start = start + 2

    return value
```

(a) (0 points)
Suppose we want the function to return an error if either (1) the value of `start` is less than or equal to zero or (2) the value of `start` is strictly greater than 20. What should `errorcheck()` return so that the function follows this behavior?

(b) (0 points)
Describe an abstraction that can be made for this function. For example, in lecture, we changed our $\sqrt{7}$ function to a $\sqrt{x}$ function.

(c) (0 points)
Suppose that your abstraction from part (b) is implemented. What function call can you make so that the function returns the same value as it would have before the abstraction was implemented? For example, in lecture, we could make the call `root(7)`.

(d) (0 points)
Does your `errorcheck()` implementation in part (a) still make sense?

6. **Factorials**
The factorial of a positive integer n is the product $1 * 2 * \ldots * n$. The factorial operator is denoted with '!', for example: $5! = 5 * 4 * 3 * 2 * 1 = 120$.

(a) (0 points)
Suppose we want to write a factorial function in Python. First, the function declaration gives the name of the function and information about the arguments. For example, in lecture 1, we had

```
def root(x, tol=0.1)
```

as the function declaration of our root finder. `x` and `tol` are the arguments, and `tol` has a default value of 0.1.

Write the function declaration for a factorial function with name `factorial` and a single argument, `n`.

(b) (0 points)
Implement the factorial function. Do not worry about using exact Python syntax–pseudocode is fine.

(c) (0 points)
The number of ways to pick a subset of k items from a set of n items, where order of selection does not matter, is denoted $\binom{n}{k}$ (often pronounced "n choose k"). There is a simple formula for the function:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Write the function declaration for a function that computes $\binom{n}{k}$ with function name `nchoosek` and function arguments `n` and `k`.

(d) (0 points)
Implement the `nchoosek` function from part (c) by calling your factorial function from part (b). Again, do not worry about using exact Python syntax–pseudocode is fine.