Due: Tuesday 8/20 before 9:15

1. Get the html or pdf file of the Python tutorial from

   `http://docs.python.org/download.html`.

   Read sections 1, 2, 3, 4.1, 4.2, 4.3, 4.6, 5.1, 5.2, 5.3, 5.4, 5.5, 5.6 or as needed to answer the questions 4 and 6 that require basic understanding of Python. Work through examples in the tutorial. (Nothing to be handed in for this question.)

2. Consider the following algorithm:

```
def isprime(n):
   for i =  2, ..., sqrt(n):
      if n is divisible by i return false
   return true
```

   What does the algorithm do? What is the running time of the algorithm expressed as a function of n? Suppose n is represented as a binary number. Is this algorithm polynomial? Is it exponential?

3. Order the following functions from asymptotically smallest to asymptotically largest, indicating ties if there are any:

n^2	$n!$	$n^{2.376}$	$\lg(n!)$	$n^2 \lg n$	$\lg^2 n$
$\ln \ln n$	$n^{1/\lg n}$	$n^{\lg \lg n}$	$\ln n$	1	n
$2^{\lg n}$	$(\lg n)^{\lg n}$	$4^{\lg n}$	$(n+1)!$	$\sqrt{\lg n}$	$n \lg n$

   For simplicity, write $f(n) \ll g(n)$ if $f(n) = o(g(n))$ and $f(n) \equiv g(n)$ if $f(n) = \Theta(g(n))$. For example, the functions n^2, n, $\binom{n}{2}$, n^3 could be sorted as $n \ll n^2 \equiv \binom{n}{2} \ll n^3$.

4. Consider the function `improve` defined as follows:

```
def improve(f):
   known = {}
   def compute(arg):
      if arg in known
         known[arg] = f(arg)
      return known[arg]
   return compute
```

   Take the first function we wrote for computing Fibonacci numbers,

```
def fib(n):
   if n==0 or n==1:
      return 1
   else:
      return fib(n-1)+fib(n-2)
```

enter or load the definitions of `fib` and `improve` in your Python interpreter and then try the following two experiments:

(a) call `fib(20)` and then `fib(30)`

(b) when `fib(30)` halts or you interrupt it, type `fib = improve(fib)` and then call `fib(100)`.

What is going on? What does `improve` do and how does it work?

[Useful Python background: definition of function, functions as return values, dictionaries.]

5. Consider the stable matching problem in a situation where the number of men (n) is greater than the number of women (m). Define an appropriate notion of stable matching for this case. Does there always exist a stable matching? Describe an efficient algorithm to find one if one exists.

6. **[Implementation question]** Implement the stable matching algorithm in Python. Your program should have a function `matchthem` that takes as arguments two lists of rankings: the first line of this function's definition should be

   ```
   def matchthem(mensprefs, womensprefs):
   ```

   Each of the two arguments should be a list of lists of integers, interpreted like the matrices from the class example: `mensprefs[i][j] = k` means that `i`'s `j`-th preference is `k`, analogously for `womensprefs`. The course page links to a file that defines the lists that correspond to the example from the notes.

   Your function `matchthem` should return two dictionaries, one keyed on men and the other on women. For a man `i`, the dictionary entry `manmatch[i]` should be the index of the woman with whom `i` is matched. Similarly, for a woman `j`, the dictionary entry `womanmatch[j]` should be the index of the man with whom `j` is matched. This is more information that is needed, strictly speaking, but there is a reason I am asking for this.

   For example, when run with the provided input, the result should be the two dictionaries

   `{ 0:1, 1:3, 2:0, 3:4, 4:1 }` and

   `{ 1:0, 3:1, 0:2, 4:3, 1:4 }` .

   Hint: consider writing an extra function `prefers` that will test if a woman prefers a man to her current fiancé.

   Question: what is the running time of your algorithm? Which of the "basic" operations are performed in constant time, and which are not?