1. (15) Write a Python code fragment which will create 5 files, named a1 through a5, i.e. open them for writing.

```python
def inttostring(n):
    m = n
    out = ''
    for i in range(4):
        c = m % 256
        out += _____________
        m /= 256
    return out
```

```python
def stringtoint(s):
    m = 0
    for i in range(4):
        m = ______________
    return m
```

3. (15) Look at the handout on conversion to/from Roman numerals. Suppose `romanNumeralMap` had been implemented as a dictionary rather than as a tuple of tuples, so that for instance one element would be 'M':1000. Show how the `for` loop in `toRoman()` would be rewritten.

4. (15) Suppose we have a list `x` which we want to rotate, i.e. the new `x[0]` will be the old `x[1]`, the new `x[1]` will be the old `x[2]`, ..., and the new `x[len(x)-1]` will be the old `x[0]`. Write Pythonic code to do this (it must be single-statement code for full credit).

5. (15) Suppose our source file `x.py` defines functions `f`, `g` and `h`. Elsewhere in the code, a call is to be made to one of these functions (with no arguments), depending on what the user puts on the command line. For example, if the user types

   ```python
   python x.py g
   ```

   then the program will execute `g`. Write Pythonic code which makes the call (must be single-line code for full credit).

6. (20) TCP treats all bytes sent by a network node as collectively comprising one long message. In some applications, though, we wish to send “records” of a given length, say 80 characters. A function which is supposed to read the next record would start with whatever bytes had been “leftover” from before, and then read from the network until it had obtained 80 bytes. For such applications, we might write a subclass of `socket`. It would have an instance variable named `incoming`, and an instance function named `getrec()` with a single argument, `reclength`. The function `getrec()` would return a string of length `reclength` (or an empty string, if we are at the end of the entire message), which it would obtain by first removing bytes from `incoming` and calling `recv()` as necessary.

Write the function `getrec()`, in a short, Pythonic manner.

**Solutions:**

1. 

```python
f = []
```
for i in range(1,5):
    f.append(open('a'+str(i),'w'))

Note the need for the list here. You could not, for example, keep assigning the result of open() to the same variable, which would mean you could not perform any further operations on the files.

2.

chr(c)
...
256*m+ord(s[3-i])

3. Change the for to

for numeral,integer in romanNumeral(Map.items())

Note that this presumes that dictionary itmes are stored in the order in which they are shown in the code which uses them. This works on the CSIF platform, but a more careful version would sort first.

4. For example,

x = x[1:] + [x[0]]

5. The easiest and most straightforward way (given our course materials) is

exec sys.argv[1]+'()'

6.

def getrec(self,reclength):
    self.incoming = [] # not destructive, as NO characters will be left over
    # from the last call to this function, since we
    # read until we get exactly reclength characters
    ncharsread = 0
    while 1:
        chunk = self.recv(reclength-ncharsread)
        if chunk == []: return []
        self.incoming += chunk
        ncharsread += len(chunk)
        if ncharsread == reclength:
            return self.incoming