0. [Students did well on this with various answers, so I'll omit my answer.]

1A. Here are short implementations.

```python
def encipherRot13(s):
    return encipherRepeatedPad(s, 'N')

def encipherCaesar(s, p):
    return encipherRepeatedPad(s, p)
```

[This question was asked, in somewhat different language, at the end of our Ciphers assignment. Anyone who understood that assignment in its entirety did well on this question.]

1B. There are short implementations of the first two functions; we just have to construct substitution lists that capture rotation-by-13 and rotation-by-arbitrary-letter. [I'll omit some of the typing.]

```python
def encipherRot13(s):
    key = ['N', 'O', ..., 'Y', 'Z', 'A', 'B', ..., 'L', 'M']
    return encipherSubstitution(s, key)

def encipherCaesar(s, p):
    key = ['A', 'B', ..., 'Y', 'Z']
    k = ord(p) - ord('A')
    key = key[k:] + key[:k]
    return encipherSubstitution(s, key)
```

In contrast, there is no simple way to implement `encipherRepeatedPad()` in terms of `encipherSubstitution()`. In `encipherRepeatedPad()`, each character is transformed based not just on its own value, but also on its position in the plaintext (relative to the pad). For example, an 'S' in one part of the plaintext might be enciphered as 'E', while an 'S' in another part of the plaintext might be enciphered as 'Q'. This is beyond the abilities of `encipherSubstitution()`.

2A. There are $4 \cdot 6$ ways to win with four pieces in a horizontal line. There are $3 \cdot 7$ ways to win with four pieces in a vertical line. There are $3 \cdot 4$ ways to win with four pieces going diagonally up-right, and $3 \cdot 4$ ways to win with four pieces going diagonally up-left. Thus there are 69 ways to win with four pieces in a line. [I gave full credit for the preceding analysis. However, there
are also $3 \cdot 6 + 2 \cdot 7 + 2 \cdot 3 + 2 \cdot 3$ ways to win with five pieces in a line, $2 \cdot 6 + 1 \cdot 7 + 1 \cdot 2 + 1 \cdot 2$ ways to win with six pieces in a line, and $1 \cdot 6 + 0 \cdot 7 + 0 \cdot 1 + 0 \cdot 1$ ways to win with seven pieces in a line. Thus there are really 142 ways to win.]

2B. All ways to win use pieces in the middle column, except for the vertical lines in the other columns. Of the 69 ways to win, $3 \cdot 6$ of them are vertical lines in the other columns. Therefore 51 out of 69 ways to win use pieces in the middle column. [Of the full 142 ways to win, 100 use pieces in the middle column.]

2C. A **heuristic** is a rule of thumb that one adopts to solve a complicated problem, that often improves one’s chances of success but does not guarantee success. (In contrast, an **algorithm** for solving a problem always succeeds.) For Plot Four it appears that “put your pieces in the middle column early” is a decent heuristic; it improves the number of ways that you can win while decreasing the number of ways your opponent can win.

2D. If the board is $b$, then this Boolean expression captures whether ’red’ has won in the bottom of the left-most column:

```python
b[0][0] == 'red' and b[0][1] == 'red' and b[0][2] == 'red' and b[0][3] == 'red'
```

3. [I give a few answers of each kind; you did not need to.]

   *module*: math, cTurtle, random, operator
   
   *class*: Turtle, str, list
   
   *method*: Turtle.forward, str.split, list.index

4. The output is

0
10
20
30
40

A more efficient version is

```python
for i in range(5):
    print i * 10
```
It loops 5 times rather than 50, and replaces % and if with * (which is comparatively fast, by the way). Another efficient version is

for i in range(0, 50, 10):
    print i

As a bonus, here’s an efficient version that uses a while loop:

i = 0
while i < 50:
    print i
    i += 10

[Many students left 0 off the output or appended 50 to the output. This is a common kind of error — misunderstanding where a loop begins or ends — even for experienced programmers. Some students altered the loop counter i within the body of the for loop. This is unorthodox and dangerous and hence should be avoided.]

5.

COMMENT0: Scan the string until ‘href’ is found.
COMMENT1: Scan from there until ‘”’ is found.
COMMENT2: Copy everything between ‘”’ and the next ‘”’ into the result string.
returned: ‘http://www.carleton.edu/’.

[Some students wrote poor comments because they didn’t understand what the code was doing; that is reasonable. Other students wrote poor comments despite perfectly understanding the code. For these students, the problem was that their comments narrated the code — as in “Set k to 0, then start a while loop...” — instead of explaining what the code was supposed to accomplish.]