You can use your sheet of notes but should have no electronic devices out in front of you. The various parts of each of the four problems below are largely independent of one another, which means you can solve the later parts of the problem even if you did not correctly solve the earlier parts. The exception is Problem 3, in which part (b) does depend on part (a). 1(b) is a bit tricky. Do all of your work in the blue exam book provided.

Each part of each problem is worth 10 points.

1. The function qualifying, whose code is shown in Figure 1, is a simplified version of one of those ‘can I get into college?’ decisions. (As you probably can guess, the function max returns the larger of the values x and y, and min returns the smaller of the two values.)

   ```python
   def qualifying(m,v):
       if max(m,v)>=600:
           if min(m,v)>=500:
               return True
       if m+v>=1200:
           return True
       return False
   ```

   Figure 1. The function for Problem 1.

   (a) What is the value returned by this function when it is called with arguments 700,400? With the arguments 650,500?

   (b) Suppose the line if  m+v>=1200 is changed to elif  m+v>1200. This changes the behavior of the function. Give an example of a pair of values m and v for which the original function and this modified version give different results.

   (c) Write a function equivalent to qualifying that does not use any if statements. (It can be done in a single line.)

   **Solution:** The function returns True if either the maximum of the two scores it at least 600 and the minimum at least 500, or if the total score is at least 1200. Neither of these holds for the pair 700,400, so the first answer for (a) is False; but the first condition, on the maximum and minimum values, just barely holds for the pair (650,500), so the second answer for part (a) is True.
We can formulate the condition that the function returns True as

\[(\min(m,v) \geq 500 \text{ and } \max(m,v) \geq 600) \text{ or } (m+v \geq 1200)\]

so the answer to (c) is

```python
def qualifying(m, v):
    return (min(m, v) >= 500 and max(m, v) >= 600) or (m + v >= 1200)
```

Part (b) can be answered by ‘talking it through’, but here I’ve shown a more systematic approach, drawing two little decision diagrams: The one on the left shows the logic of the function as it is written. The one on the right shows what changes if you replace the last if by elif. This shows you how to choose a pair of values that gives True in the original version and False in the altered version: have \(\max \geq 600\), \(\min < 500\), and \(\text{sum} = 1200\). For instance, 800,400 works.
Common errors:

In part (a): Treating 650,500 as a False instance, apparently because students misread \( \min(m,v) \geq 500 \) as \( \min(m,v) > 500 \).

In part (c): Printing rather than returning the result.
Computing the result correctly but not using it, and instead returning True.
Changing the \( \text{or} \) to \( \text{and} \).

Something that is not an error, but you might want to do differently: A number of students wrote the condition in (c) as

\[
\min(m, v) \geq 500 \text{ and } \max(m, v) \geq 600 \text{ or } m + v \geq 1200
\]

That happens to be correct, but for a reason that might not be so evident, namely that \( \text{and} \) takes precedence over \( \text{or} \), in the same way that multiplication takes precedence over addition. (For example, if the correct answer had the form \((A \text{ or } B) \text{ and } C\) it would be a mistake to leave out the parentheses.) I think it is a better practice in these problems to include parentheses around the subexpression with \( \text{and} \)--it makes things easier to read.
2. Consider the function `trits` whose code is shown in Figure 2. Observe that this function prints output rather than returning a value.

```python
def trits(n):
    while n>0:
        print(n%3)
        n=n//3
```

Figure 2. The function for Problem 2

(a) What is printed when you call `trits(2)`? `trits(16)`? `trits(-1)`?

(b) A common error in writing such code is to put statements that belong in a loop outside the loop, and vice-versa. What would be the result of calling `trits(16)` if the statement `n=n//3` were moved to the left, so that it started in the same column as the word while?

(c) Another common error is to use the wrong kind of division. What would be the result of calling `trits(16)` if the statement `n=n//3` were replaced by `n=n/3`?

**Solution.** Traces of the execution of `trits(2)` and `trits(16)` are shown in the tables below. The left-hand column shows the value of `n` each successive time that the while is encountered. The middle column shows the succession of values that are printed

<table>
<thead>
<tr>
<th>n</th>
<th>n%3</th>
<th>n//3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n</th>
<th>n%3</th>
<th>n//3</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For `n=-1`, the body of the while loop is never executed, and the function terminates without printing anything.

For part (b), the value of `n` is not changed within the loop. Thus if we start with `n=16` (or any other value greater than 0), we will repeatedly print the value `n%3` forever. This is an infinite loop.
For part (c), the answer I was looking for (but see below!) is that the program also enters an infinite loop, because we use floating point division to divide 16 by 3, giving 5.3333, then 5.3333 by 3, giving whatever that is, and then again, and again, and again, never reaching 0.

Common errors: Many students wrote 0 and 1,2,0 for the first two parts of (a), apparently treating the last remainder, rather than the last quotient, as zero.

Some students wrote that an argument of -1 in (a) causes an error, which it does not. The condition simply evaluates to False, and execution of the function ends.

In (b), it was common to say that just the first line was printed out---apparently believing that the loop was only executed once, instead of infinitely often.

Now, about (c). Students were understandably uncertain what % does when you apply it to floats rather than ints, and some said that it results in an error. This is a reasonable and respectable answer, especially in light of the fact that we only used // and % in the context of integer division. But in fact, if x is a float, then x//3 is defined as the largest integer less than or equal to x/3, and x%3 is x-3*(x//3). Thus 16%3 is 0.3333. After some hesitation, I decided to give this answer full credit.

The real point though is the ‘never reaching zero’ business, which leads to the ‘correct’ infinite loop answer. As it turns out, that answer is not correct either, although I would not expect any student to know it: While the supply of real numbers is infinite, computers only use a fixed number of bits to represent a float, with the result that there are only finitely many float values. In particular, there is a smallest positive float! What happens when you divide the smallest positive float by 3? You get zero. The function modified as described prints about a thousand or so values, and then terminates.
3. Consider the function `design` in Figure 3. Once again, this function prints output rather than returning a value. (Recall that if the arguments to the `print` function include `end=' '`, then the output does not advance to the next line, and the next thing to be printed will appear immediately to the right of the output. The statement `print()` advances to the next line.)

```python
def design(s):
    m=(len(s)+1)//2
    for row in range(m):
        for column in range(m):
            print(s[row+column],end='')
        print()
```

Figure 3. The code for Problem 3.

(a) What is the output that results when you call `design('ABCDE')`?
(b) You should now have an idea of what this function prints in general. (You might observe that the answer is slightly different, depending on whether the argument is a string of odd or even length.) Write an equivalent function that has only a single `for` statement, instead of nested `for` statements, by using string slices.

Solution: When the argument is 'ABCDE', `len(s)` is 5, so `m` is 3. We can thus trace the execution in the table below

<table>
<thead>
<tr>
<th>row</th>
<th>column</th>
<th>row+column</th>
<th><code>s[row+column]</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>D</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>D</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>E</td>
</tr>
</tbody>
</table>

The successive outputs are the characters in the last column. Because we used the argument `end=' '`, these outputs are printed one after the other on the same line, except when the `print()` statement is executed. This happens just before the row variable is incremented. Thus the output is
In general, the function prints an output of m rows, each with m characters. The jth row begins with s[j], so the whole pattern can be written

```python
def design2(s):
    m=(len(s)+1)//2
    for row in range(m):
        print(s[j:j+m])
```

Common errors: This was one of those problems where you can’t do part (b) unless you get part (a) right. I tried to accommodate solutions to (b) that correctly implement your incorrect ideas about what gets printed in (a), but frankly this was a difficult problem to grade, because there were so many different answers.

A very minor error was to use the code above, but add an additional `print()` within the main loop. This will print an additional blank line between the rows, so it is not exactly equivalent to the original.

4. A standard problem in introductory calculus classes is to find the largest possible volume $V$ in cubic inches of a cylindrical soup can made from $S$ square inches of metal. But you don’t have to do any calculus here! The answer is

$$V = 2\pi \left( \frac{S}{6\pi} \right)^{3/2}$$

(a) The code shown in Figure 4 is supposed to perform this calculation and return the resulting volume. However there are two errors in the code, both of which cause the function to return an incorrect answer. Correct these errors. (The value of of $\pi$ is obtained by importing the math library,

```python
def maximal_volume(s):
    return 2*math.pi*(s/6*math.pi)**3/2
```

Figure 4. Code for Problem 4.
which is why it appears as \texttt{math.pi} in this code.)

(b) Write a main program that uses this function to print a table of the maximum volumes for $S=10, 20, ..., 100$. You don’t have to worry about pretty formatting or displaying a reasonable number of decimal places. Just make sure your table contains two numbers---the values of $S$ and $V$---on each line. This should not be long—not more than four or five lines of code at most.

\textbf{Solution.} As I said in class during the exam, you should assume that the statement \texttt{import math} appeared earlier in the code, so its absence is not to be treated as an error. The two errors are both in the operator precedence in the expression for the volume, and are solved by appropriate placement of parentheses. A correct version is

\begin{verbatim}
def maximal_volume(s):
    return 2*math.pi*(s/(6*math.pi))**(3/2)
\end{verbatim}

For part (b), there are many correct solutions. Here is a particularly simple one:

\begin{verbatim}
for j in range(10):
    s=(j+1)*10
    print (s,maximal_volume(s))
\end{verbatim}

Common errors: These were mostly in part (b): It was imperative that you call the function \texttt{maximal_volume} in the program, and not just reproduce the formula. Often the loop was constructed incorrectly—for instance, in the above it is an error to write \texttt{j+=1} as the last statement inside the \texttt{for} loop. On the other hand, there are many correct ways to construct the loop, using \texttt{range(10, 110, 10)}, for instance, or using a \texttt{while} statement instead of \texttt{for}.

\textbf{Distribution of scores}

This is shown below. The median and mean scores were both around 75. What is important here is not the absolute score, but how you did relative to the rest of the class. I do not give letter grades on exams, but if I were grading
the course based on this one exam only, I would say that scores in the 50-59 range would get between C- and C, in the 60-69 range between C and B-, in the 70-79 range B- to B, in 80-89 B+ to A- and in the 90-100 range A- to A. Scores below 40 cannot be considered passing.