After an exam, it’s common (in the Fall and Spring) to go over the solutions to the test during Section. However, with our mixed class (part online, part in-person), I thought of an alternate way to do things: we’re going to look at possible solutions to some of the problems.

I’ve copied a few of the exam problems into the document here, along with several possible solutions. None of these solutions were taken directly from any student exam, but they represent common types of answers that I saw.

Some of these answers are correct; some are nearly correct, but have subtle bugs; some are wrong. Your job is to grade them:

- If an answer is correct, write that it is correct; give a quick summary of how it works. (Some of the correct answers are a little hard to understand!)
- If an answer is almost correct but has a small bug, identify that bug and explain why it doesn’t work.
- If an answer is wrong, explain why.

Remember, it’s not enough to simply say right/wrong. Explain what you see.

NOTE 1:
A few solutions use functions that you probably have not seen yet. Feel free to explore the Python documentation to find out how they work. And as always, we’ll be roaming the room, answering questions.

NOTE 2:
Midway through the section, we’ll come back together, and I’ll give you a quick overview of how list comprehensions work - since some of the solutions use them. If you are able to explain the answers that use list comprehensions, that’s great - but if you can’t it’s OK to just write “I don’t know list comprehensions yet.” We’ll accept that answer for now. :)
Problem 1(c)

Give the value of x at the end of the following snippet:

$$x = [1, 2, 3]$$
$$y = [x, x, x]$$
$$y[2].append(100)$$

**Answer:**

$$x = [1, 2, 3]$$

**Answer:**

$$x = [1, 2, 3, 100]$$

**Answer:**

$$x = [1, 2, 100]$$

Problem 1(j)

Assume that you have two variables, named `fred` and `barney`. Create a string (using the pattern below) which includes the value of both; store that string into the variable `message`. For instance, if `fred=123` and `barney=[10,11]`, then the string should be:

Fred: 123   Barney: [10,11]

**Answer:**

```python
message = "Fred: {}   Barney: {}
```

**Answer:**

```python
print("Fred: {}   Barney: {}").format(fred,barney)
```

**Answer:**

```python
"Fred: {}   Barney: {}".format(fred,barney)
```
Problem 4(a)

Write a snippet of code which will print all of the integers from 1 to 100, inclusive (but in reverse order). Print one per line.

Answer:
for i in range(100):
    print(100-i)

Answer:
r = range(1,101)
reverse(r)
for i in r:
    print(str(i))

Answer:
for i in range(1,101,-1):
    print(i)

Answer:
x = 100
while x > 0:
    print(x)
    x -= 1

Answer:
for i in range(100,0,-1):
    print(i)
Problem 4(b)

Write a function named `powers(val, max_pow)`. It should return a list of values, which are the powers of `val`, from `val^0 = 1` to `val^{max_pow}`.

Answer:
```python
def powers(val, max_pow):
    cur_val = 1
    retval = []
    for i in range(max_pow+1):
        retval.append(cur_val)
        cur_val *= val
    return retval
```

Answer:
```python
def powers(val, max_pow):
    return [val**i for i in range(max_pow+1)]
```

Answer:
```python
def powers(val, max_pow):
    retval = []
    for p in range(max_pow):
        retval[p] = val^p
    return retval
```
Problem 4(c)

Write a snippet of code that will 1) read a line of input from the user; 2) split it into words (separated by whitespace); 3) convert each word to an integer (you don’t have to check for exceptions here); and 4) print out the sum of all of those integers.

Answer:
```
words = input().split()
vals = []
for w in words:
    vals.append(int(w))
print(sum(vals))
```

Answer:
```
words = input().split()
for i in range(len(words)):
    words[i] = int(words[i])
print(sum(words))
```

Answer:
```
words = input().split()
total = 0
for i in range(len(words)):
    total += int(words[i])
print(total)
```

Answer:
```
words = input().split()
vals = map(int, words)  # look up map() in the documentation!
print(sum(vals))
```
Problem 4(d)

Write a function `add_pairs(vals)` which, when given a list, will return a new list, where each pair of adjacent values has been added together. Thus, the new list is half the length; for instance, `add_pairs([1,10, 23,47])` should return `[11,70]`. Use an assert to ensure that the length of the parameter is an even number of elements.

**Answer:**
```
def add_pairs(vals):
    assert len(vals) % 2 == 0
    retval = []
    while len(vals) > 0:
        retval.append(vals[0]+vals[1])
        vals = vals[2:]
    return retval
```

**Answer:**
```
def add_pairs(vals):
    assert len(vals) // 2 == 0
    retval = []
    for i in range(0,len(vals)-1, 2):
        retval.append(vals[i]+vals[i+1])
    return retval
```

**Answer:**
```
def add_pairs(vals):
    assert len(vals) % 2 == 0
    retval = []
    for i in range(len(vals)):
        if i%2 == 0:
            retval.append(vals[i]+vals[i+1])
    return retval
```
def add_pairs(vals):
    assert len(vals) % 2 == 0
    retval = []
    for k in range(len(vals)/2):
        retval[k] = vals[2*i] + vals[2*i+1]
    return retval