CS120: Introduction to Computer Programming II

Summer 2017

Section 4 Handout: June 30th 2017

Name:________________                        Netid:_________________

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:  
- Incorrect (see below)

Answer:  
- Correct.
- y contains three references to the same list as x
- y[2] is the same list as x
- Thus, y[2].append(100) is the same as x.append(100)

Answer:  
- Incorrect (see above)

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:  
- Close, but incorrect because ‘fred’ and ‘barney’ are probably not string objects. Use str()

Answer:  
- Close, but incorrect because it prints the string instead of storing it to a variable.

Answer:  
- Very close. Each type will be converted to a string by format(), and then dropped into the correct location. However, it does not store the result into any variable.
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:
```python
for i in range(100):
    print(100-i)
```
- Correct. The range object goes from 0 to 99 (inclusive). Thus, it prints out 100, then 99... In the last iteration, it prints 1.

Answer:
```python
r = range(1,101)
reverse(r)
for i in r:
    print(str(i))
```
Technically: reverse() is a method of the list class, not a global function. reversed() is the correct function. But as I said on Piazza, it's OK to overlook that.
- Correct.
- range(1,101) builds the range 1 to 100 (inclusive).
- We then reverse it, so that it is 100 to 1 (inclusive).
- Then we print out the elements (the str() function is unnecessary)

Answer:
```python
for i in range(1,101,-1):
    print(i)
```
- Incorrect.
- It starts at 1, but the step value is -1, so it counts down - and the end point is 101 (above). Thus, the range has no elements.
- Fix it with range(100,0,-1)

Answer:
```python
x = 100
while x > 0:
    print(x)
    x -= 1
```
- Correct.
- In the first pass, we print 100. We decrement x, and stop when x==0.
- Thus, the last thing we print is 1

Answer:
```python
for i in range(100,0,-1):
    print(i)
```
- Correct. (see above)
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
```

- Correct.
- This uses a running value (that starts at 1); we multiply it by ‘val’ over and over
- This does max_pow+1 passes through the loop; each pass adds a value to the list, and then multiplies the running value by ‘val’
- The first value in the list will be 1, the last will be val^{max_pow}

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

- Correct.
- This uses a list comprehension. The iterator goes from 0 to max_pow (inclusive)
- For each iteration, it builds val, raised to the i power.
- NOTE: ** is exponentiation, ^ is bitwise XOR

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

- Incorrect. Three errors:
  - Range ends at max_pow-1 instead of max_pow
  - Indexing into the list instead of appending to it
  - ** is exponentiation, ^ is bitwise XOR
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))

- Correct (but a little inefficient)
- Creates an empty list
- For each word in the input, converts the word to an integer, and appends it to the list
- Sums up the new list, and then prints out the answer.

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

- Correct
- Takes the list of words, created by split...
- ...for each element, converts it to an integer, and then replaces it in the list
- Prints out the sum

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

- Correct
- Has a running total, in a variable
- As words are converted to integers, adds them into the running total
- NOTE: Would have been more elegant to use for w in words:

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

- Correct
- Technically, this is effectively identical to the first answer - it uses the map() function to apply the int() function to each word, and build a new list.
**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:**
```python
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
```

- **Correct**
- As long as the list still has more elements, read the first two, add them, and add them to the `retval`. Then slice those two elements off, then loop back

**Answer:**
```python
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
```

- **Nearly correct**
- The `assert` is wrong, because it uses division instead of modulo
- The `-1` in the range is harmless (but unnecessary). For instance, if the length of the list was 10, then the range should use indices 0,2,4,6,8. It doesn’t matter whether the end point is 9 or 10; since the step is 2, the last loop iteration will have i=8.

**Answer:**
```python
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
```

- **Correct**
- While the range will include the odd and even indices, the `if()` statement means that the code will skip over any indices which are not even. So effectively, this is the same as a `range()` with a step of 2.
**Answer:**

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

- Nearly Correct
- This uses a range which is half as much; for instance, for a list of length 10, the range would be 0,1,2,3,4. For each iteration, it multiplies the iterator by 2.
- However:
  - It uses i instead of k inside of the loop body
  - It never sets the length of the list, so indexing (instead of appending) is not going to work.

**Answer:**

def add_pairs(vals):
    assert len(vals) % 2 == 0
    return [vals[2*i]+vals[2*i+1] for i in range(len(vals)/2)]

- Correct
- This is equivalent to the previous example (except without the bugs).
- It uses a list comprehension:
  - Iterating over half of the length of the list
  - For each element, add the two correct values