Allowing full access to your computer for a take-home test means that certain types of questions can no longer be asked, as you could just type them into your computer and see what the output was. So instead, questions aimed at your ability to understand fundamentals of code will come at things from some different perspectives. I include a few here so you can have an idea of how they might appear.

Practice Questions

1. Write an appropriate doc-string for the below function. It should include both a high-level description of what the program does (which is not just a line by line description) as well as what types of inputs are allowed and what types of outputs are returned.

```
def problem1(a, b=2):
    c = ""
    for n in a:
        if int(n) % b != 0:
            c += n
    return c
```

Solution: There is of course some flexibility in this, but this is what I might write:

```
"""
Takes a sequence of numeric characters and drops all values from the
sequence that are evenly divisible by the given value b.

The input sequence must have elements of consisting of string digits.
If a list, elements comprised of multidigit numbe strings are
possible, but they will be concatenated to a single string at the
end, so it may be confusing.

Args:
    a (sequence of numeric characters,
       either a string or a list of single character elements)
    b (an integer)

Returns:
    (a string): a string containing all input elements that were not
                divisible by b
"""
```

2. An alternative way to test your understanding of a function is to ask you to work backwards. Take, for example, the function:

```
1  def problem2(n):
2      r = 0
3      t = 1
4      while n > 0:
5          r += t * (n % 2)
6          t *= 10
7          n //= 2
8      return r
```

Given this function, which you could of course type into your computer, determine what input value to the argument `n` would end up returning the integer 11011000001. Guessing and checking is a poor method to solving these! Rather, you should look at the code to develop an understanding for what is occurring and let that guide you. It should be very easy to check yourself though!

Solution: Since I'm making a comparison to a number on line 4, it seems likely that `n` is a number of sorts, probably an integer since I'm later looking at a remainder on line 5. Then on line 5 I can see that I'll only be adding `t` if my number `n` happens to be odd on that iteration. Then I play around inputing a few simple `n` values (like 1 and 2) to see if I can get a feel for what is happening. After a few of those, I realize this is just an algorithm to output the binary form of a base-10 number. Which means that the input I'd need to enter in as `n` would just be the base-10 representation of the binary number 11011000001. Which is 1729.

3. Given the function

```
1  def problem3(s):
2      r = ""
3      for i in range(len(s)):
4          if i % 2 == 1:
5              r += s[i]
6          else:
7              r = s[i] + r
8      return r
```

determine what value of the argument `s` returns the string `"street"`.

> **Solution:** `r` is clearly a string, so it would seem the elements of `s` also need to be a string since we are adding them on lines 5 and 7. Given that lists aren't supposed to be on this test, I'd assume that `s` is therefore a string, of which we are looping through the indices. If the index is odd, we add the letter at the end of existing string `r`, whereas if the index is even then we add it at the start of `r`. In this way, it seems we are going to construct the string `r` basically starting from the middle and then alternating sides. So I'd find the center of `"street"` between the `r` and `e` and figure my argument must start with one of them. Since the *second* letter goes to the end, I'd start with `r` and then flip-flop my way outwards, getting `"retest"`. Then I'd of course check it to make sure I hadn't made a little mistake.

4. Given the pair of functions

```
1  def prob4a(s):
2      r = 0
3      new = ""
4      for i in range(len(s)):
5          if s[i].isspace() and is_english_word(s[r:i]):
6              new += prob4b(s[r:i])
7              r = i + 1
8      new += prob4b(s[r:])
9      return new
10
11
12 def prob4b(y):
13     done = False
14     for r in y[::-1]:
15         if done:
16             return r
17         if r.lower() in "aeiou":
18             done = True
19     return " "
```

what input value for `s` would return from function `prob4a` the string `"Hi world"`? You can assume the function `is_english_word` has been correctly imported from the `english` library.

> **Solution:** Looking at `prob4a` first, it would seem that `s` is going to be some sort of string, since we are checking to see if elements of it are spaces and we are checking a slice of it to see if it is an english word on line 5. If the if statement on line 5 is satisfied, when we are passing in a slice of `s` as

our input to `prob4b`, so we can deduce that `y` is also a string in `prob4b`. Looking at `prob4b` then, we can see that there is a boolean flag involved, and that we are looping through whatever string is passed in *backwards*. Then we are checking each lowered letter to see if it is a vowel, at which point we toggle the flag true. But we don't return until the NEXT time through the loop, at which point `r` is the next letter in the sequence, which means what is returned is the letter that appears *before* the last vowel in the string (since we are looping backwards). If there are no vowels in the string, a space is returned.

Resuming back on line 6 in `prob4a` then, we are going to concatenate this letter (or space) to our running string `new`. And then we would reset `r` to the next index. Since we are only entering this code if the current letter was a space, the result is that this loop walks through each word in a string, separated by spaces. We've seen this pattern before, in Pig Latin for instance. So the main loop in `prob4a` is responsible for plucking out each word in a string, and then passing that word onto `prob4b`, which gives back the character before the last vowel. Line 8 is just taking care of the last word in the string, that likely has no space character after it.

To reproduce `"Hi world"` then, we just need to come up with a string of valid english words where the character before the last vowel is the character we need in our desired output. Thus there are LOTS of potential inputs that would result in this output! One potential option though would be: `"Hit tie why wish noon rip lord ding"`. Note that the only tricky word here is that the third word must contain no vowels so that we get the space character.