A1. `frequencies` is a function [namely, the function that the code is trying to define].

A2. `string` is a string.

A3. `counts` is a list of integers [of length 26].

A4. `char` is a string [of length 1].

A5. `index` is an integer.

A6. `index != None` is a Boolean.

A7. `counts[index]` is an integer [because it's an element of `counts`].

A8. `total` is an integer [because it's a sum of integers].

A9. `sum` is a function [built-in].

A10. `freqs` is a list of floating-point numbers [of length 26].

A11. `i` is an integer.

A12. `range(len(counts))` is a range in Python 3, or a list of integers in Python 2. [Intuitively a range is a list of integers. I accepted either "range" or "list of integers".]

B. Orange is halfway between red, which is $[255, 0, 0]$, and yellow, which is $[255, 255, 0]$. So I'm going to say that orange is somewhere around $[255, 128, 0]$. In binary this is 1111 1111 1000 0000 0000 0000_2. In hexadecimal it's $FF8000_{16}$.

C. [Many students looped up to $n-1$. This is wasteful. I rewarded students who made some attempt to cut that waste down. Many students failed to return n in the case that n is prime.]

```
def factor(n):
    for d in range(2, math.sqrt(n)):
        if n % d == 0:
            return d
    return n
```

D1. Loop through the 13 kinds. For each kind, loop over the hand, counting how many

cards there are in that kind. If there are three or more in that kind, then return `True`. If all of the loops finish without returning `True`, then return `False`.

D2.

```
def hasThreeOfAKind(hand):
    kinds = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'jack', 'queen', 'king', 'ace']
    for kind in kinds:
        count = 0
        for card in hand:
            if card[0] == kind:
                count += 1
        if count >= 3:
            return True
    return False
```

E1. First, 12 is $0000\ 1100_2$. Second, 10 is $0000\ 1010_2$, which bit-flips to $1111\ 0101_2$, which is 1 less than $1111\ 0110_2$, which is -10.

E2. [I'll omit the details, although you should not.] The final answer is $1011\ 1000\ 1000_2$, which is more than 8 bits. It truncates to $1000\ 1000_2$.

E3. Notice that the truncated answer bit-flips to $0111\ 0111_2$, which is 1 less than $0111\ 1000_2$, which is 120. Therefore the truncated answer is -120, which is correct.

F. Here's one solution. Babatope picks a message P randomly. He encrypts it using Alice's public n and e, and sends the encrypted message to "Alice". That is, the challenge is $P^e \% n$. If she is really Alice, then she can decrypt the message back to P, and send this P to Babatope as the response. If Babatope receives P back, then he can be confident that "Alice" is really the owner of d.

It is somewhat important that P be random. [In grading, I reserved 1 point out of 8 for noticing this.] What we really don't want is a P that "Alice" can guess even though she isn't really Alice. For example, always sending the same P to authenticate everyone would be a bad idea. And sending a question that requires human judgment is also not ideal; the whole process should be automated. And sending a question that requires Alice to know Babatope is not ideal; they may have never met in real life.