1a. alice = [ 3, 5, 11 ] # some odd numbers
    bob = [ 3, 5, 11 ] # some primes
    print(alice is bob) # prints:
    alice.append(7) ###
    print(alice) # prints:
    print(bob) # prints:

Explain what is happening to alice and bob (if anything) on the statement marked ###.

1b. alice = bob = [ 5, 3, 11, ] # some odd and prime numbers
    print(alice is bob) # prints:
    sorted(alice) ###
    print(alice) # prints:
    print(bob) # prints:
    bob.sort() ###
    print(alice) # prints:
    print(bob) # prints:

Explain what is happening on the statements marked ###.

1c. alice = bob = 'odd numbers'
    print(alice is bob) # prints:
    bob.replace('odd','prime') ###
    print(bob) # prints:
    print(alice) # prints:
    alice = alice.replace('odd','uneven') ###
    print(alice) # prints:
    print(bob) # prints:

Explain what is happening on the statements marked ###.
In *Bulls & Cows* we had to pick a word from the dictionary. We had read the dictionary into memory and thus we knew exactly how many words there were. To pick from the words with equal probability (i.e., with a *uniform distribution* of possible selections) we used code like the following:

```python
dictionary = words()
wordNumber = randint(0,len(dictionary)-1)
word = dictionary[wordNumber]
```

We were fortunate to have enough memory to hold the dictionary.

Sometimes, we can't hold all the values in memory at the same time. Instead, we may have fleeting access to a *stream of objects* through an iterable. If we're limited to seeing exactly one object at a time we can still pick uniformly using the following approach:

a. At all times we have a current favorite value, chosen. Because we have not and may never see any values, we initially set chosen to None. If we stop now, this choice cannot be confused with a valid choice in any circumstance.

b. We keep a counter, n, that keeps track of how many values we have seen. Each time we see a new object, item, we increment the counter.

c. As we consider item, we realize we have encountered n values and this may be the last. If the stream stops now we should make sure that item has a 1 in n chance of being selected. We roll an n sided die (i.e., we pick a random number from, say 1 to n) and if the die comes up '1', we update chosen to be item.

d. When the stream stops, chosen is our choice for a random object.

Notice that this program only requires enough memory to hold the values chosen, n, and item.

2. The following code has many errors. Ideally it chooses a random object from a stream of possibilities with equal chance of being selected. To the right, rewrite the code to make it correct and elegant.

```python
def chooseFrom(stream):
    """Picks a value.""
    chosen = "none"
    n = 1
    for item in stream:
        n = n + 1
        if randint(0, n) != 1:
            pass
        else:
            chosen = item
    return chosen
```

*