```
s = set("abcabc")
```

Which of the following is equivalent to the value of the set `s` above?

A. `{"abcabc"}`
B. `{"abc","abc"}`
C. `{"a","b","c","a","b","c"}`
D. `{"a","b","c"}`
E. `{"abc"}`

Answer: D
The set constructor will add all elements from the iterable to the set. All items in a set must be unique, or alternately the set “de-duplicates” the iterable. The the answer is { “a”, “b”, “c” }, the unique characters in the string.

```
a = [1, 2, 3]
b = [3, 4, 5]
c = []
for val in a:
    if val in b:
        c.append(val)
```

Which of the following produces the same value for `c` (including type) as the code above (assuming that `a` and `b` are defined as above):

```
A. c = set(a+b)
B. c = list(set(a) & set(b))
C. c = list(set(a) | set(b))
D. c = set(a) & set(b)
E. c = list(a&b)
```

Answer: B

The code appends values from a to the list c, only if they are also in b. Thus this is equivalent to set intersection. We need to be careful about types. Set operations are only defined on sets and so we need to convert the lists to sets. And the value of c should also be list and so we want to convert it at the end of the computation.