items = ["A", "B", "C"]
items[0] = "D"

After this code executes, what are the values in the list `items`?

A. "A", "B", "C"
B. "A", "B", "C", "D"
C. "D", "A", "B", "C"
D. "D", "B", "C"
E. "D"

Answer: D
The second statement reassigns the value at index 0. It does not change the length of the list. Note that you can’t perform the same operation on strings. Because strings are immutable, strings do not support item assignment (i.e., assigning to specific indices/characters).
Answer: C

Recall that `len` returns the length of its argument, so `t` must be an integer. Here all copies of "ss" "Mississippi" are replaced with "a" to create "Miaiaippi", which has length 9.
Answer: C
The upper method creates a new string "MISSISSIPPI" (all caps). When we invoke replace on that string (returned by upper), there are no instances of lower case "ss" (Python distinguishes between upper and lower case letters), and so no replacements will be performed. Thus t is "MISSISSIPPI".
a = [2, 4, 6, 8]
a.remove(4)     \[2, 6, 8\]
a.pop(2)        \[2, 6\]

What is the value of a after the above code executes
A. [2, 4]
B. [6, 8]
C. [2, 6]
D. [2, 8]

Answer: C
After `a.remove(4)`, a is [2, 6, 8], after `a.pop(2)`, the item at index 2 is removed, so a is [2, 6]
a = [2, 4, 6, 8]
a.pop(2)
a.remove(4)

What is the value of a after the above code executes
A. [2, 4]
B. [6, 8]
C. [2, 6]
D. [2, 8]

Answer: D
After a.pop(2), the item at index 2 is removed, so a is [2,4,8] and a.remove(4), a is [2,8]
s = "CS"
t = "cs"
s.lower()  # [0, 1]
for i in range(len(s)):
    print(s[i] + t[i])

When run via the green arrow in Thonny, what will the following code print?

<table>
<thead>
<tr>
<th>A. cc</th>
<th>B. Cc</th>
</tr>
</thead>
<tbody>
<tr>
<td>ss</td>
<td>Ss</td>
</tr>
<tr>
<td>C. CSCS</td>
<td>D. cC</td>
</tr>
<tr>
<td>cscs</td>
<td>sS</td>
</tr>
</tbody>
</table>

Answer: B
Invoking the lower method does not change the string s, it remains "CS". The loop executes two iterations, printing the concatenation of the characters in s and t at index 0, then index 1.
Answer: A
When s is assigned to t, both point to “abc”. When we invoke s.upper(), that doesn’t change that string, but instead creates a new string “ABC”. After assigning that value to s, it now points to “ABC”, but that statement does not change that t points to “abc”. Check out what happens at pythontutor.com:
https://pythontutor.com/visualize.html#code=s%20%3D%20%22abc%22%0At%20%3D%20s%0As%20%3D%20s.upper%28%29&cumulative=false&curInstr=0&heapPrimitives=true&mode=display&origin=opt-frontend.js&py=3&rawInputLsJSON=%5B%5D&textReferences=false