ConcepTest

Consider the code below that runs on an arbitrary list `lst`. What does it do?

```
lst = [...] # a list
sum = 0
counter = 1
while counter < len(lst) - 1:
  sum = sum + counter
  counter = counter + 1
```

- A. Computes the sum of the list
- B. Computes the sum of the list, but excludes the first element
- C. Computes the sum of the list, but excludes the last element
- D. Works fine, but crashes on the empty list
- E. None of the above

ConcepTest (Writing a Find)

```
def find(lst, value):
  '''(list, value) -> int
  Return the first occurrence of value in lst.
  If value is not found, return -1.

  >>> find([20, 40, 60], 40)
  1
  '''
  i = 0
  num = lst[i]
  while num != value:
    i = i + 1
    num = lst[i]
  if i < len(lst):
    return i
  return -1
```

In what situation does the above code fail?

- A. It never fails
- B. It fails when the list is empty
- C. It fails when the value is not found in the list
- D. Conditions B and C will both fail

Example: Writing a Find

```
def find(lst, value):
  '''(list, value) -> int
  Return the first occurrence of value in lst.
  If value is not found, return -1.

  >>> find([20, 40, 60], 40)
  1
  '''
```

Several ways to solve this:

1. Using a boolean operator in the condition of a while-loop
2. Using a `return` inside a for-loop

ConcepTest

What is the output of this code?

```
result = []
lst = [10, 20, 30, 40]
for index in range(0, len(lst), 2):
  lst[index] = index
for index in range(len(lst)):
  result.append(lst[index])
print(result)
```

- A. `[0, 1, 2, 3]`
- B. `[0, 20, 2, 40]`
- C. `[10, 20, 30, 40]`
- D. `[0, 2, 4, 6]`
- E. `[1, 2, 3, 4]`