CSCI 241 Data Structures
Practice Set 1: Variables and Arrays

1. Provide a Python function that takes as a parameter an array of positive integers. The function should return a new two-cell array where cell 0 contains the sum of the integers passed in, and cell 1 contains the product of the integers passed in.

```
def sum_product(values):
  # initial sum is 0; product is 1
  results = [0, 1]
  for val in values:
    # add each value to the sum in cell 0
    results[0] = results[0] + val
    # multiply each value into the product in cell 1
    results[1] = results[1] * val
  # after processing each value, the loop should
  # terminate, so we return to the previous indentation
  # level and return the results array (list).
  return results

# From here down is a basic test case.
# normally I would include zero and negatives,
# but the specifications say otherwise.
numbers = [4,2,5,3,1]
# call the function and save its return value
sumprod = sum_product(numbers)

# 4+2+5+3+1 = 15, so we expect to see 15
# in cell 0 of the return value
if sumprod[0] == 15:
  print("Sum OK")
else:
  print("Sum: expected 15, observed " + str(sumprod[0]))

# 4*2*5*3*1 = 120, so we expect to see 120
# in cell 1 of the return value
if sumprod[1] == 120:
  print("Product OK")
else:
  print("Product: expected 120, observed " + str(sumprod[1]))
```

2. Write a Python function that takes as a parameter an array of positive integers. The function should print the average of those integers to the screen.

```
def print_average(positive_values):
  total = 0
  # Accumulate the sum in total
  for val in positive_values:
    total = total + val

  # The specs say to print the result,
  # NOT RETURN IT!
  # Divide by n, convert to string, and display.
  print(str(total/len(positive_values)))

numbers = [4,1,5,2,3]
# The printing is happening in the function.
# No need to print here.
print_average(numbers)
```

3. If an array contains n cells, how many of them must you visit in order to locate the minimum value, assuming the contents are unordered?

   We must visit all n cells. If the values are unordered, we have no way of knowing which cell contains the minimum value. No matter what path through the array I choose, there exists a possible distribution of data where the minimum value is in the last cell I visit. We call this the worst case, and it is leading into our general discussion of performance analysis. When we analyze how well an algorithm performs, we always assume the worst case input. Assuming that your data are arranged a certain way is a leading cause of bugs in programs.

4. Provide an input to the following function that makes the function return None.

```
def noneShallPass(fellowship):
  if len(fellowship) < 9:
    return False
  elif len(fellowship) < 10:
    for i in fellowship:
      if i is None:
        continue
    if i is not None:
      i = None
    else:
      return i
  else:
    return False
  return True
```

If the list has fewer than 9 items, we return `False`. If the list has greater than 9 items (the `elif` clause), we return `False`. There are no paths through this function that do not terminate in a `return` statement, so default return values cannot be the source of the `None`. It must come from the `return i` line in the middle. That means that `i` must have the value `None`, and that must be because the last item in the 9-item array is `None`. We are left to conclude that the input must be an array that contains anything at all cells 0—7, but must contain `None` in cell 8. Here's one:

```
[anything, is, fine, for, the, first, eight, spots, None]
```