Introductory Computer Science Test

Name: _____ Date: _____

Question 1 /10

For this question, you can safely assume the following code works:

```
>>> a = ['a', 'b', 'c', 'd', 'e']
>>> (a[0], a[3]) = (a[3], a[0])
>>> a
['d', 'b', 'c', 'a', 'e']
```

Given the following code:

```
def f1(x, y):
    c = x[:]
    x[0] = 99
    print(c, x, y)
    return x

def f2(x, y):
    c = x[:]
    (c[0], c[y]) = (c[y], c[0])
    print(c, x, y)
    (x[0], c[y]) = (c[0], x[y])
    return c[x[y]]

def f3(x, y, z):
    x[y][z] = x[z][y]
    c = x[:]
    c[0][0] = f1(c[0], c[1][2])
    print(c[1:])
    x[1][1] = f1(c[0], c[1][2])

a = [1, 2, 3, 4, 5]
b = 2
print("STEP 1")
print(f1(a, b))
a = [[6, 5], [[4, 3], 2], 1]
b = 2
print("STEP 2")
print(f2(a, b))
print(a)
a = [[8, 7], [6, [5], 4], [[3, 2], 1]]
b = 1
c = 2
print("STEP 3")
print(f3(a, b, c))
print(a)
```

Return the output of the code here (and clearly indicate where the output is). Feel free to ask for more paper.

Mangled Code

```
def is_palindrome(my_str):
index = 0
index = 1
index = index + 1
my_stk = []
tot_len = len(my_str)
if len(my_str) == 0:
if len(my_str) == 1:
if tot_len % 2 == 0:
if tot_len % 2 == 1:
else:
while my_str != "" and my_str[1] == my_str[-1]:
while my_str != "" or my_str[1] == my_str[-1]:
while my_str != "" and my_str[0] == my_str[-1]:
while my_str != "" or my_str[0] == my_str[-1]:
while index < (tot_len + 1) / 2:
while index < (tot_len - 1) / 2:
while index < (tot_len) / 2:
while index <  tot_len / 2 + 1:
while index <  tot_len / 2 - 1:
while my_stk != [] and my_stk[0] == my_str[index]:
while my_stk != [] or my_stk[0] == my_str[index]:
while my_stk != [] and my_stk[-1] == my_str[index]:
while my_stk != [] or my_stk[-1] == my_str[index]:
my_str = my_str[1:-1]
my_str = my_str[:index + 1] + my_str[index:]
my_str = my_str[:index] + my_str[index + 1:]
my_str = my_str[:index] + my_str[index:]
my_stk.append(my_str[index])
my_stk.append(my_str[0])
my_stk.pop(-1)
my_stk.pop(0)
ret = False
ret = True
ret = my_str == ""
ret = my_str != ""
return ret
return my_stk == []
```

Question 2 /14

Dann made a perfectly good function called `is_palindrome` that takes in a string and returns True if and only if a string is a palindrome. That is, the string read forwards is the same as it is read backwards.

Dann made the following examples and saved it to a text file in the event that he has to re-implement it (for some odd reason). He can easily copy and paste the following into his docstring:

```
>>> is_palindrome("")
True
>>> is_palindrome("a")
True
>>> is_palindrome("ab")
False
>>> is_palindrome("aa")
True
>>> is_palindrome("wow")
True
>>> is_palindrome("wot")
False
>>> is_palindrome("dood")
True
>>> is_palindrome("dork")
False
>>> is_palindrome("rikir")
True
>>> is_palindrome("racecar")
True
```

Dann eventually went downstairs to his basement to read up on Jeremy Lin joining the Beijing Ducks. While he was downstairs, the **CODE MANGLER** struck. Deleting all his comments, removing all indentation, removing duplicate code, and re-arranging lines of code. To make matters worse, there seems to be a bunch of code from another program shuffled in with this code. Help Dann by re-creating the `is_palindrome` function (complete with internal and external commenting) in the space below (or on another sheet of paper) with **ONLY the mangled code on the previous page**. HINT: Carefully think about the examples.

Documentation

```
class list(object)
 |  Methods defined here:
 |  append(self, object, /)
 |      Append object to the end of the list.
 |
 |  clear(self, /)
 |      Remove all items from list.
 |
 |  copy(self, /)
 |      Return a shallow copy of the list.
 |
 |  count(self, value, /)
 |      Return number of occurrences of value.
 |
 |  extend(self, iterable, /)
 |      Extend list by appending elements from the iterable.
 |
 |  index(self, value, start=0, stop=2147483647, /)
 |      Return first index of value.
 |
 |      Raises ValueError if the value is not present.
 |
 |  insert(self, index, object, /)
 |      Insert object before index.
 |
 |  pop(self, index=-1, /)
 |      Remove and return item at index (default last).
 |
 |      Raises IndexError if list is empty or index is out of range.
 |
 |  remove(self, value, /)
 |      Remove first occurrence of value.
 |
 |      Raises ValueError if the value is not present.
 |
 |  reverse(self, /)
 |      Reverse *IN PLACE*.
 |
 |  sort(self, /, *, key=None, reverse=False)
 |      Stable sort *IN PLACE*.
```