CSc 110 Introduction to Computer Programming I

Final Exam – Practice

Fall 2024

This is an individual exam, and there will be a 120 minute window for you to complete it.

Do not cheat off of those nearby you.

Make sure to write your responses clearly and legible. There's no need to add comments or docstrings to your code.

Each question has a designated box where your response should go. Use a DARK pen or pencil, and write INSIDE the answer boxes provided. Anything outside the box will **not** be considered as part of your response.

You may do extra work to arrive at the response, but the response MUST go **in** its designated box. Anything outside the box will **not** be considered as part of your response.

If you have any questions, please raise your hand.

Question 1

Loop table

Give the function below:

```
def even_up_nested(lists):
  for i in range(len(lists)):
    for j in range(len(lists[i])):
      lists[i][j] += lists[i][j] % 2
  return lists
```

Complete the `loop table` below with the corresponding values of `i`, `j`, `lists[i][j] % 2` and `lists` for the following function call:

```
even_up_nested([[0, 2, 1, 3], [3, 2], [0, 1]])
```

RESPONSE FOR QUESTION 1:

Question 2

Evaluate the code below. Enter in each box what the last line of code in each chunk prints. When the code throws an error, write `ERROR` in the response box.

```
# 2A
numbers = []
numbers.append(10)
numbers.append(1)
numbers.insert(1, 0)
print(numbers)
```

RESPONSE 2A:

```
# 2B
numbers = [1, 2]
numbers.insert(0, 3)
numbers[2] = 100
print(numbers)
```

RESPONSE 2B:

```
# 2C
numbers = [1, 2]
print(numbers[2])
```

RESPONSE 2C:

```
# 2D
numbers = {1, 2, 3, 1, 2}
print(numbers)
```

RESPONSE 2D:

```
# 2E
numbers = {1}
numbers.add(2)
numbers.add(2)
print(numbers)
```

RESPONSE 2E:

Question 3a – Selection Sort

```
def find_min_index(items):
  min_index = None
  for i in range(len(items)):
    if min_index == None or items[min_index] > items[i]:
      min_index = i
  return min_index

def selection_sort(items):
  begin_index = 0
  while begin_index < len(items)-1:
    min_index = find_min_index(items[begin_index:]) + begin_index
    items[begin_index], items[min_index] = items[min_index], items[begin_index]
    begin_index += 1
  return items
```

Using selection sort (code for reference above), how many sweeps and swaps would it take until the list gets sorted? Show your work. Indicate the number of sweeps and swaps in their designated boxes.

[10, 4, 2, 10, 5, 7]

SHOW YOUR WORK FOR QUESTION 3:

SWEEPS:

SWAPS:

Question 3b – Bubble Sort

```
def bubble_sort(items):
  swapped = False
  end = len(items)-1
  while not swapped:
    swapped = True
    for i in range(end):
      if items[i] > items[i+1]:
        items[i], items[i+1] = items[i+1], items[i]
        swapped = False
    end -= 1
```

Using bubble sort (code for reference above), how many sweeps and swaps would it take until the list gets sorted? Show your work. Indicate the number of sweeps and swaps in their designated boxes.

[10, 4, 2, 10, 5, 7]

SHOW YOUR WORK FOR QUESTION 3:

SWEEPS:

SWAPS:

Question 4

Write a Python function called `trim_ends` that has a 2D list as parameter. The function should mutate and return the argument list, removing the first and last element of each sublist (if the sublist is not empty).

Test case:

```
numbers = [ [10, 20, 200, 40],
            [ ], [10],
            [1000, 1000, 10],
            [20, 30, 4, 100] ]
trim_ends(numbers)
assert numbers == [[20, 200], [ ], [ ], [1000], [30, 4]]
```

RESPONSE FOR QUESTION 4:

Question 5

Write a python function that does the following:

1. Its name is `create_list`
2. It takes two arguments, a set of `strings` and an integer `n`
3. It returns a list that contains each string from the set repeated `n` times

```
items = {"banana", "apple", "pear"}
assert create_list(items, 2) == ['banana', 'banana', 'apple', 'apple', 'pear', 'pear']
```

RESPONSE FOR QUESTION 5:

Question 6

See the python code and the contents of the file name `data.txt`. The python code writes content to a file named `result.txt`. You must determine what the contents of `result.txt` will be after the code runs. Put your answer in the response box.

data.txt

```
one silver edging
trees leaves are green
this simple request is finally
a moody final countdown
```

```
def is_acceptable(x):
    for i in range(0, len(x)-1):
        if x[i] == x[i+1] and x[i] in "aeiou":
            return True
    return False

def main():
    data = open('data.txt', 'r')
    result = open('result.txt', 'w')
    for line in data:
       words = line.strip('\n').split(' ')
       for word in words:
           z = is_acceptable(word)
           if z:
              result.write(word + '\n')
    data.close()
    result.close()

main()
```

RESPONSE FOR QUESTION 5:

Question 7

Write a function called `star_consonants` that has one string as parameter. The function returns a new string of the same length as the parameter string, with every consonant replaced by an asterisk (`"*"`).

```
assert star_consonants("banana") == "*a*a*a"
assert star_consonants("a") ==  "a"
assert star_consonants("apple") ==  "a***e"
assert star_consonants("") ==  ""
```

RESPONSE FOR QUESTION 7:

Question 8

Write a function called `total` that has one parameter named `file_name`, being the name of a file to read. The function expects that the file to read has one or more integer numbers on it per line. It iterates over the lines and numbers to compute the total of all the numbers from the file. It returns the total.

Example of `data.txt` file

```
5 10 2
1 0
5 1
20
```

```
assert total("data.txt") == 44
```

RESPONSE FOR QUESTION 8:

Question 9

Write a function that does the following:

1. Its name is `average_rows`
2. It has one parameter named `lists`, being a 2D list of float numbers
3. For each list (row) within the 2D list, it should calculate the average of the numbers within, round it at two decimals, and place the resulting average in a new list at the same index
4. It returns the list of the averages

```
assert average_rows([[1.2, 5.4, 4.3, 2.0], [0.0, 1.0]]) == [3.23, 0.5]
assert average_rows([[], [10.5]]) == [None, 10.5]
assert average_rows([[1.0], [2.5, 3.5, 4.5], [0.0, 0.0], [0.0, 2.0]]) == [1, 3.5, 0.0, 1.0]
```

RESPONSE FOR QUESTION 9:

Question 10

Write a function called `mutate_dict` that takes two arguments: a dictionary with string keys and integer values, and a set of strings. The function **mutates** and returns the dictionary argument adding the strings in the set as keys in the dictionary:

1. if the key already exists in the dictionary, do not change anything
2. if the key does not exist in the dictionary, create with with the value zero associated with it

```
test_dictionary = {"z": 1, "x": 2, "r": 20}
mutate_dict(test_dictionary, {"a", "z", "r", "b"} )
assert test_dictionary == {"z": 1, "x": 2, "r": 20, "a": 0, "b": 0}
```

RESPONSE FOR QUESTION 10:

Question 11

Write a Python function called `remove_vowel_ending` that takes a list of strings as argument (you can assume strings are never empty). The function should remove list items that end in a vowel (check for upper or lower case).

```
test_list = ["Peter", "Bob", "Ana", "MARIO", "CEDRIC"]
remove_vowel_ending(test_list)
assert test_list == ["Peter", "Bob", "CEDRIC"]

assert remove_vowel_ending([]) ==  []
```

RESPONSE FOR QUESTION 11:

Question 12

Write a Python function called `remove_vowels` that takes a list of `strings` as argument The function should **mutate** and return the argument list, removing the vowels of each item in the list.

```
test_list = ["Peter", "Bob", "Ana", "MARIO", "CEDRIC"]
remove_vowels(test_list)
assert test_list == ["Ptr", "Bb", "n", "MR", "CDRC"]

assert remove_vowels([]) ==  []
```

RESPONSE FOR QUESTION 12:

Question 13

Write a python function that takes a list of integers representing years, and evaluates whether each year (for example, `2024`) is a leap year or a regular year. The function should return a dictionary with the results.

Leap years are:

- divisible by 4 and not divisible by 100
- divisible by 100 and also divisible by 400

All other cases are regular year.

Test cases (your `leap_year` function definition should work with these function calls):

```
years = [1992, 2000, 1900, 1700, 2024]
result = leap_year(years)
assert result == {1992: 'Leap Year',
                  2000: 'Leap Year',
                  1900: 'Regular Year',
                  1700: 'Regular Year',
                  2024: 'Leap Year'}
```

RESPONSE FOR QUESTION 13:

Question 14

Write python code that given a list of years, it mutates the list by removing the leap years. All your code should be in functions.

RESPONSE FOR QUESTION 14: