BRAC University

Department of Computer Science and Engineering

CSE110: Programming Language I

Examination: *Theory Assignment #2*　　　　Semester: *Summer 2022*

Date: __ __ / __ __ / 2022　　　　Deadline: 22 August, 2022

ID: _ _ _ _ _ _ _ _ _	Name: (Please write in CAPITAL LETTERS)	Section: **29**

1. Write a Python program that reads 10 strings and prints all the strings at the end in ascending order of their length such that the shortest string prints first and the longest prints at last. **Must use a dictionary to group the texts of the same length**.

   Hint: You may record the lowest and highest length of the strings while adding those to the dictionary.

Sample Input	Sample Output
Python Java Swift PHP Java C++ R Go JavaScript C	R C Go PHP C++ Java Java Swift Python JavaScript

2. Write a function in Python that takes three arguments: length, data_type and prompt. The function should take the `length` number of inputs from the user and convert it into data_type and append it to a list. Finally, it should return the list of size `length`.

   You are not allowed to change the given code: function calls and data printing. You just need to write/ define the function `**take_input(...)**`.

```
# Write your code to define the take_input function here.

str_list = take_input(3)
str_list2 = take_input(2, str, "Please enter a string value: ")
int_list = take_input(5, int, "Please enter an int value: ")
float_list = take_input(3, float, "Please enter a floating value: ")

print(str_list)
print(str_list2)
print(int_list)
print(float_list)
```

Sample Input	Sample Output

`Please enter an input: Task 2`
`Please enter an input: Functions`
`Please enter an input: List`
`Please enter a string value: Hello`
`Please enter a string value: World`
`Please enter an int value: 110`
`Please enter an int value: 111`
`Please enter an int value: 220`
`Please enter an int value: 221`
`Please enter an int value: 331`
`Please enter a floating value: 431`
`Please enter a floating value: 0.12`
`Please enter a floating value: 2.14`
	`['Task 2', 'Functions', 'List']`
	`['Hello', 'World']`
	`[110, 111, 220, 221, 331]`
	`[431.0, 0.12, 2.14]`

3. Write a Python program that, at first, reads 10 strings from the user followed by an integer, **L**, and stores them in a list, **STRINGS**. Next, use **binary search** to find the first string of length **L**. Finally, print the string if there is any string of length **L** otherwise print "Could not find any string of length {**L**}" and also print how many searches it required, check the given sample output to understand the output format.

   Note: *Make sure your output format matches with the sample outputs given below. You must include the double quotations around the found string.*

Sample Input	Sample Output
Java Python Programming is fun Assignment BRAC University Programming Spring Laravel Django React 10	The first string of length 10 found is "Assignment" within 3 searches.
Java Python Programming is fun Assignment BRAC University Programming Spring Laravel Django React 9	Could not find any string of length 9. Searched 3 times.

4. Write a function, **`grouper(...)`**, in Python that takes a list of integer numbers as the only argument/ parameter and returns a tuple of dictionary that groups the numbers according to their parity and a tuple containing all the numbers in descending order.

   You are not allowed to use any built-in sorting functions, rather you must use **Selection Sort**. Moreover, the dictionary should also contain the numbers in descending order. Furthermore, you are not allowed to change the given code: function calls and data printing. You just need to write/ define the function `**grouper(...)**` and ensure the given output is being generated.

```
# Write your code to define the grouper function here.

my_list = [916, 881, 5, 286, 443, 367, 223, 254, 699, 422]
parity_dict, uniq_tuple = grouper(my_list)
for parity in ["odd", "even"]:
    if parity in parity_dict:
        print(parity.upper() + ":")
        for num in parity_dict[parity]:
            print(" >", num)

print("Sorted (non-duplicate) values:", uniq_tuple)
```

Given List	Sample Output
[916, 881, 5, 286, 443, 367, 223, 254, 699, 422]	ODD: > 881 > 699 > 443 > 367 > 223 > 5 EVEN: > 916 > 422 > 286 > 254 Sorted (non-duplicate) values: (916, 881, 699, 443, 422, 367, 286, 254, 223, 5)
[538, 210, 1177, 248, 1029, 795, 6, 1184, 522, 971]	ODD: > 1177 > 1029 > 971 > 795 EVEN: > 1184 > 538 > 522 > 248 > 210 > 6 Sorted (non-duplicate) values: (1184, 1177, 1029, 971, 795, 538, 522, 248, 210, 6)

5. Write a function in Python that takes a list of integer numbers and a boolean flag to determine the order (ascending or descending) of sorting those integers - the default is True which means ascending. The function must return a tuple containing a new sorted list, the number of swaps required for Selection Sort, and the number of swaps required for Bubble Sort respectively.

   You are not allowed to use any built-in sorting functions or change the given code: function calls and data printing. You just need to write/ define the function `` `count_swaps(...)` `` and make sure the given output is being generated.

```
# Write your code to define the count_swaps function here.

non_given_list = [916, 881, 5, 286, 443, 367, 223, 254, 699, 422]
sorted_list, selection_swaps, bubble_swaps = count_swaps(non_given_list)

print("Ascending Sort:")
print("List:", sorted_list)
print("Selection Swaps:", selection_swaps)
print("Bubble Swaps:", bubble_swaps)

print()

sorted_list, selection_swaps, bubble_swaps = count_swaps(non_given_list,
False)

print("Descending Sort:")
print("List:", sorted_list)
print("Selection Swaps:", selection_swaps)
print("Bubble Swaps:", bubble_swaps)
```

Given List	Sample Output
[916, 881, 5, 286, 443, 367, 223, 254, 699, 422]	Ascending Sort: List: [5, 223, 254, 286, 367, 422, 443, 699, 881, 916] Selection Swaps: 8 Bubble Swaps: 26 Descending Sort: List: [916, 881, 699, 443, 422, 367, 286, 254, 223, 5] Selection Swaps: 5 Bubble Swaps: 19

6. Write an `encode_global(...)` function in Python that does not take any arguments. The function should apply Caesar Cipher to the variables `lang`, `ide`, `task`, and `element` containing strings. You must use the concept of local and global scopes. Caesar Cipher only encrypts the alphabets, however, your function should shift every character to 3 positions right on the ASCII table but make sure to make the shifting effects circular for the alphabets, for example, 'X' should be 'A' and 'y' should be 'b' (case-sensitive).

   You are not allowed to change the given code: function calls and data printing. You just need to write/define the function `encode_global(...)` and ensure the given output is being generated.

```python
# Write your code to define the encode_global function here.

lang = "Python"
ide = "Colab"
task = "Question 6"
element = "X-Kryptonite"

print("Language:", lang)
print("IDE:", ide)
print("Task:", task)
print("Element:", element)

print("===============")
encode_global()
print("===============")

print("Language:", lang)
print("IDE:", ide)
print("Task:", task)
print("Element:", element)
```

Sample Output

```
Language: Python
IDE: Colab
Task: Question 6
Element: X-Kryptonite
===============
===============
Language: Sbwkrq
IDE: Frode
Task: Txhvwlrq#9
Element: A0Nubswrqlwh
```

7. Write a Python program including a function `` `fix(...)` ``. The program should read inputs from a file named \`input.txt\`. The file may contain corrupted words such that instead of all alphabets, there are numbers and special characters in place of a few specific alphabets (case-sensitive). The table of those alphabets is given below.

0	1	2	3	4	5	6	7	8	@
o	i	z	E	p	s	G	L	B	a

Your program must use the `` `fix(...)` `` function to fix the words given a string line containing multiple such corrupted words. The function should fix the given string and return the fixed string at the end.

Hint: *You may use a dictionary for the alphabet mapping with numbers and special characters table.*

Sample File	Sample Output
`He770 wor7d!`	`HeLLo worLd!`
`3arth i5 sp1nnin6 f@5t!`	`Earth is spinninG fast!`
`8e5t 0f 7uck!`	`Best of Luck!`

8. Trace the following code:

```
methods = {1: int, 0: str}
result = "0"
a, b = 1, 10
x = 10
i = 0
while i < 8:
    j = 0
    while j < 30:
        q = a + b - (4 + a % 3) / 4
        if int(q) % 10 < 5:
            b = b + int(q)
        else:
            a = a + int(q)
        k = b % a % 3
        method = methods[k]
        result = method(result) + method(x)
        print(result)
        j += b
    i += 1
```

9. Trace the following code:

```
def method1(x):
    return x + int(5 / 3)

def method2():
    return 3.0 % 2

x = 2
p = 1
sum = 0
q = 0.0
while (p < 12):
    q = x + p - method1(sum) / method2()
    sum = sum + x + int(q)
    x += 1
    print(sum)
    if (x > 5):
        p += int(4 / 2)
    else:
        p += 3 % 1
sum = sum + p
print(sum)
```

10. Trace the following code:

```python
string = "Programming is fun"
result = ""
d = {}

for ch in string:
    if ch not in d:
        d[ch] = 0
    d[ch] += 1

for ascii in range(0, 255):
    ch = chr(ascii)
    if ch in d:
        i = 0
        while i < d[ch]:
            result += ch
            i += 1

print(result)
```