Fundamentals of Computer Science - programming exam

Instructions

IMPORTANT - BEFORE YOU START:

1. Edit the `.py` file in Spyder and solve the exercises.
2. Upload the `.py` file to BBoard.

Additional notes

- The exam is **not open book**!
- **Clean your mess** after you finish! Comments and tests/asserts are welcome, but unnecessary code should be deleted.
- **Tests are provided** for each exercise, although they are initially commented out. They are very helpful to understand the intended behavior and to check the correctness of your implementation. Make sure to uncomment and run them as you go though the exercises.
- **Do not** make use of libraries, python's built-in functions, sets, dictionaries, list's methods, etc.. unless otherwise stated. Just plain loops and if/else statements.

Partial 1

Exercise 1 - Unique Elements Checker

Define a function `has_unique_elements(l)` that returns `True` if all elements in the list `l` are unique and `False` otherwise.

We remind that dictionaries and sets are not allowed: use nested loops.

Raise an exception if `l` is not a list.

Example:

```
has_unique_elements([1, 2, 3])  # Output: True
has_unique_elements([1, 2, 2])  # Output: False
```

Exercise 2 - Mode of a list

Write a function called `mode` that takes a list of integers as its only argument and returns the mode of the least, that is the element occurring the largest number of times. Proceed as follows:

1. First sort the list. You can use the built-in python function `sorted` that returns a sorted copy of a list.
2. After the sort, find the mode using a single loop. If more that one element have the maximum number of occurrencies, you should return the smallest one.

Further requirements: a) if the list contains **any** non `int` element raise an exception. b) If the list is empty return `None`.

Examples:

```
In[1]: l = [1, 5, 5, 7, 1, 9, 1]
In[2]: mode(l)
Out[2]: 1

In[3]: l = [1, 3, 3, 5, 9, 9, 10]
In[4]: mode(l)
Out[4]: 3

In[4]: mode([1, 2, 2.1]) # displays an error
```

Exercise 3 - Countsort

Write a function called `countsort` that takes two arguments, a list of integers `l` and a positive integer `k`. Each integer in the list is assumed to be between `0` and `k-1`.

The function should sort the list `l` using the “counting sort” algorithm explained below, and return it. This algorithm can sort lists in O(n) as long as they are small, which is better than any alternative. Here is how it works:

1. Create an auxiliary list of length `k` filled with zeros, call it `c`. You can use the list comprehension syntax to create `c`:

   ```
   c = [0 for i in range(k)]
   ```

   to create the list. The list `c` will be used as a counter: at the end, `c[i]` should represent how many times the number `i` is present in the list `l`.
2. Scan the input list `l` and for each element increase the corresponding counter in `c`.
3. Scan the counter list `c`. For each element `c[i]`: if it's zero then do nothing, otherwise the output list must have the element `i` repeated `c[i]` times. See the note below on how to do it.

For example, given `l=[3, 5, 2, 0, 3]` and `k=6` the counter list will be `c=[1, 0, 1, 2, 0, 1]` because in `l` there is 1 zero, 0 ones, 1 two, 2 threes, 0 fours, 1 five. Then scanning `c` from the start we see that we need to output a zero, skip the ones, then a two, two threes, skip the fours, and finally a five: `[0, 2, 3, 3, 5]`.

Note: Notice that between points 2 and 3 of the procedure, i.e. right after the list `c` was built, we no longer need the contents of `l`. Therefore, we can overwrite it and the sorting can be performed in-place. For the full score, do it like that. Alternatively you create an extra empty list and grow it with the `append` method to perform point 3.

If not passed, the argument `k` should take value equal to the length of the list.

Examples:

```
In [1]: countsort([3, 5, 2, 0, 3], 6)        # SEE EXAMPLE IN THE TEXT
Out[1]: [0, 2, 3, 3, 5]

In [2]: countsort([3, 5, 2, 0, 3], 10)       # AS ABOVE, BUT LARGER k: SAME RESULT
Out[2]: [0, 2, 3, 3, 5]

In[2]: l = [4, 1, 3]; countsort(l, 5)       # FOR THE FULL SCORE, OPERATE IN-PLACE...
Out[2]: [1, 3, 4]

In [3]: l                                    # ...NOTICE THAT l HAS BEEN MODIFIED
Out[4]: [1, 3, 4]
```