CSCI E-7 Introduction to Python Programming for Life Sciences

Dino Konstantopoulos, **Final Exam: Part B, in-class Portion**

You have 4 hours for this exam. It should not take that long because you can pick any 4 out of the 6 problems to solve. Each problem is worth 25 points each. The Bonus is worth 20 points. Total maximum points you can obtain: 120 points. Also please generate a pdf and upload your notebook and pdf (and text files from any of the problems below that you were asked to create). Do NOT remove the problem statements. You may provide pseudocode and partial code when stuck for partial credit).

Note that we will credit partial work, as long as you are getting *closer* to the desired solution. Random work (throwing code blindfolded, hoping to get to a solution) will not get you any credit. *Good luck*!

Problem 1 (25 points)

Turn the dictionary `problem1` into a list that includes both keys and values. [5 points) Then, go through each element of the list and split the list into *two* lists: so that all integers are in one list and all strings are in the other. (5 points) Sort both lists in descending order using the python `sorted` function. (5 points) Remove any duplicate integers and print both lists to a text file (problem1.txt). Also show both lists in a cell. (10 points)

Expected output:

['list of stringshereindescendingorder']

[list of numerics in descending order]

In [1]:

```
problem1 = {'RNA':'deoxynucleotides', 'polymerase':1, 'ANN':2, 'DNA':3, 'bases':'deo
:'Introns', 1:4, 'kinetics':5, 'mRNA produced': 'sequence','primers1':62, 221:'proto
```

In [98]:

```
#Turn the key/value pair into a list of tuples
problem1_list = [(k, v) for k, v in problem1.items()]
problem1_list
```

Out[98]:

```
[('RNA', 'deoxynucleotides'),
 ('polymerase', 1),
 ('ANN', 2),
 ('DNA', 3),
 ('bases', 'degenerate'),
 ('polyploid', 'Introns'),
 (1, 4),
 ('kinetics', 5),
 ('mRNA produced', 'sequence'),
 ('primers1', 62),
 (221, 'proton'),
 (-10, 'RNNase'),
 ('seven', 'diploid'),
 ('genomes ', 8),
 (9, 'paralogy'),
 ('primers', 'organellar')]
```

In [99]:

```
#Take the tuples and convert them into a single list
#Source: https://stackoverflow.com/questions/7558908/unpacking-a-list-tuple-of-pairs
test1,test2 = zip(*problem1_list)
problem1_list_merged = test1 + test2
#problem1_list_merged
```

In [100]:

```
#Apply list comprehension to the merged list to divide into a list of integers and s
#Apply the sorted funtion to order in descending order
#Source: https://stackoverflow.com/questions/39824683/filter-lists-elements-by-type-
#Source: https://www.geeksforgeeks.org/sorted-function-python/
integers = sorted([elm for elm in problem1_list_merged if isinstance(elm, int)], rev
string = sorted([elm for elm in problem1_list_merged if isinstance(elm, str)], rever
```

In [101]:

```
print("Sorted Integer List : "), print(integers)
print("\nSorted String List : "), print(string)
```

```
Sorted Integer List :
[221, 62, 9, 8, 5, 4, 3, 2, 1, 1, -10]

Sorted String List :
['seven', 'sequence', 'proton', 'primers1', 'primers', 'polyploid', 'p
olymerase', 'paralogy', 'organellar', 'mRNA produced', 'kinetics', 'ge
nomes ', 'diploid', 'deoxynucleotides', 'degenerate', 'bases', 'RNNase
', 'RNA', 'Introns', 'DNA', 'ANN']
```

Out[101]:

```
(None, None)
```

In [129]:

```
#Remove the duplicate values in each list
#Source: https://stackoverflow.com/questions/7961363/removing-duplicates-in-lists
int_set = sorted(list(set(integers)), reverse = True)
str_set = sorted(list(set(string)), reverse = True)
```

In [118]:

```
print("Sorted Unique Integer List : "), print(int_set)
print("\nSorted Unique String List : "), print(str_set)
```

```
Sorted Unique Integer List :
[221, 62, 9, 8, 5, 4, 3, 2, 1, -10]

Sorted Unique String List :
['seven', 'sequence', 'proton', 'primers1', 'primers', 'polyploid', 'p
olymerase', 'paralogy', 'organellar', 'mRNA produced', 'kinetics', 'ge
nomes ', 'diploid', 'deoxynucleotides', 'degenerate', 'bases', 'RNNase
', 'RNA', 'Introns', 'DNA', 'ANN']
```

Out[118]:

```
(None, None)
```

In [104]:

```
#Save both lists to a text file!
#Source: https://stackoverflow.com/questions/899103/writing-a-list-to-a-file-with-py
with open('problem1.txt', 'w') as f:
    f.write("Integer List: ")
    for item in int_set:
        f.write("%s " % item)
    f.write("\n\nString List: ")
    for item in str_set:
        f.write("%s " % item)
```

Problem 2 (25 points)

Write a python function that uses methods from the python Stack Class (https://www.tutorialspoint.com/python/python_stack.htm) (note: scroll to bottom of the page) and remove *all* elements from one list and add them to a Stack. (10 points) Use it on one of the sorted lists in Problem 1 above (output above is provided in the example). Print the list and the new Stack in a file called problem 2.txt. Also show the stack and list in a cell. (15 points)

Note: Skipping this problem (unless any of my other answers are even less correct). I do want to better understand the solution once the final is completed. The problem is a little unclear to me, but I do believe that I get the gist. Below is my attempt but would love some feedback - should I be popping off the values and then simultaneously adding them to the newly created stack? I'm just pushing all the values from the list to a new stack and then clearing the list currently (if that was the intention of the question). Also, how would I print a stack - unless you are looking for the object as my _ *str_* function is not working correctly?

In [274]:

```
int_set
```

Out[274]:

```
[221, 62, 9, 8, 5, 4, 3, 2, 1, -10]
```

In [275]:

```
#This is just to deal with testing because I don't want to impact int_set by cleari
int_set2 = [i for i in int_set]
int_set2
```

Out[275]:

```
[221, 62, 9, 8, 5, 4, 3, 2, 1, -10]
```

In [276]:

```
class Stack:

    def __init__(self):
        self.stack = []

    def __str__(self):
        return "Stack: "+self.stack

    def add(self, dataval):
    # Use list append method to add element
        if dataval not in self.stack:
            self.stack.append(dataval)
            return True
        else:
            return False

    # Use list pop method to remove element
    def remove(self):
        if len(self.stack) <= 0:
            return ("No element in the Stack")
        else:
            return self.stack.pop()

    # Use peek to look at the top of the stack
    def peek(self):
        return self.stack[-1]
```

In [277]:

```
def List2Stack(list):

    #Instantiate a stack
    p2Stack = Stack()

    #Add in all values form the input list to the new Stack
    for i in list:
        p2Stack.add(i)
        print("Last Value Added: ", p2Stack.peek())

    #Remove values from list
    list.clear()
    print("List Emptied:", list)
```

In [278]:

```
List2Stack(int_set2)
```

```
Last Value Added:  221
Last Value Added:  62
Last Value Added:  9
Last Value Added:  8
Last Value Added:  5
Last Value Added:  4
Last Value Added:  3
Last Value Added:  2
Last Value Added:  1
Last Value Added:  -10
List Emptied: []
```

Problem 3 (25 points)

Part A: Given the DNA sequence below. return a list of all palindromes (https://en.wikipedia.org/wiki/Palindrome) within the sequence below, that have a length greater than or equal to 3. Provide the number of palindromes found as well. (15 points)

Part B) Print the longest palindrome from the sequence. Do this using code. (10 points)

An ***example*** is reproduced here below for illustration:

If the original sequence is : AATACGGGCATAA The list of palindromes are: ['ATA', 'ATACGGGCATA', 'TACGGGCAT', 'ACGGGCA', 'CGGGC', 'GGG', 'ATA'] There are **7** palindromes in the sequence The longest palindrome is: ATACGGGCATA

The sequence you will work with is given in next cell. It is a real DNA sequence.

In [211]:

```
sequence = "CCGATTTAGCCGTGGAGAGTCGCGCGCTGAGAGGTGGGAGAGGCTAAACGCGACAGCGCAAATCCTTTATAT
```

In [221]:

```
#Source: https://codereview.stackexchange.com/questions/110079/find-all-distinct-pa
def all_palindromes(text):
    """Return list with all palindrome strings within text.

    The base of this algorithm is to start with a given character,
    and then see if the surrounding characters are equal. If they
    are equal then it's a palindrome and is added to results set,
    extend to check if the next character on either side is equal,
    adding to set if equal, and breaking out of loop if not.

    This needs to be repeated twice, once for palindromes of
    odd lengths, and once for palindromes of an even length."""

    results = set()
    text_length = len(text)
    for idx, char in enumerate(text):

        # Check for longest odd palindrome(s)
        start, end = idx - 1, idx + 1
        while start >= 0 and end < text_length and text[start] == text[end]:
            results.add(text[start:end+1])
            start -= 1
            end += 1

        # Check for longest even palindrome(s)
        start, end = idx, idx + 1
        while start >= 0 and end < text_length and text[start] == text[end]:
            results.add(text[start:end+1])
            start -= 1
            end += 1

    return list(results)
```

Note: I'm a little confused by this problem as the example solution above does not include the largest palindrome but incuded a repeat (ATA). I was able to find an example finding unique palindromes (over 1) which is what I understood from the problem initially. If this is completely wrong please refer to my Problem 2. See below for my validation of the example:

In [254]:

```
sequence2 = 'AATACGGGCATAA'
[palindrome for palindrome in all_palindromes(sequence2) if len(palindrome) >=3]
```

Out[254]:

```
['CGGGC', 'GGG', 'ACGGGCA', 'TACGGGCAT', 'AATACGGGCATAA', 'ATACGGGCATA
', 'ATA']
```

In [268]:

```
#Part A
#Use a list comprehension to sort out any palindomes that are less than 3 in length
allPalThree = [palindrome for palindrome in all_palindromes(sequence) if len(palindr
print(allPalThree)
```

```
['CTGGGTC', 'TTATGTGTATT', 'GCCCCG', 'GCTCG', 'CGCGCGC', 'TGTGT', 'GCG
CG', 'CGTGC', 'TGCCCCGGCCCCGT', 'ATACCATA', 'GCGTGCG', 'TGCGAAGGAAGCGT
', 'TATTATGTGTATTAT', 'GTTTTG', 'AACGCGACAGCGCAA', 'GGCAACGG', 'GTCGCG
CGCTG', 'TGCGGTCCTGGCGT', 'CCCGGCCC', 'TTTATATTT', 'GCAACG', 'GAAGGAAG
', 'GGTGG', 'TGCGT', 'CCGGCC', 'GAGTCGCGCGCTGAG', 'CCCC', 'CCGATTTAGCC
', 'AAGAGAA', 'CGAAGGAAGC', 'GTCACTG', 'GTATTATG', 'CTTTC', 'TATAT', '
CGCGC', 'TAGGTATTATGTGTATTATGGAT', 'AATTCGAAGAGAAGCTTAA', 'ATTCGAAGAGA
AGCTTA', 'GGTCCTGG', 'AGGCAACGGA', 'ATTTCTTTCTTTA', 'GTG', 'TGGAGAGTCG
CGCGCTGAGAGGT', 'CCAAATAGGTGTGGATAAACC', 'AGAGTCGCGCGCTGAGA', 'ATTA',
'CAAATAGGTGTGGATAAAC', 'CAGAC', 'ACAATTTCTTTCTTTAACA', 'AAAA', 'TGTCCT
GT', 'AATAGGTGTGGATAA', 'GGTATTATGTGTATTATGG', 'AAACGCGACAGCGCAAA', 'T
TCTTTCTT', 'GGTGTGG', 'GCAAAACG', 'GGAGAGG', 'TAAACGCGACAGCGCAAAT', 'G
AGAGTCGCGCGCTGAGAG', 'CGAAGAGAAGC', 'TTTCTTTCTTT', 'CGATTTAGC', 'CGTTT
TGC', 'CTCACCACTC', 'GTGGAGAGTCGCGCGCTGAGAGGTG', 'AGAAGA', 'AAAATTCGAA
GAGAAGCTTAAAA', 'TCT', 'CCCCGGCCCC', 'TTTCTTT', 'AATTTCTTTCTTTAA', 'TC
ATACCATACT', 'GTCCTG', 'AGAGA', 'ATCGCTA', 'GTCATACCATACTG', 'TCGTGGCG
GTGCT', 'AGA', 'GTGTG', 'CGTAGGTATTATGTGTATTATGGATGC', 'ACGCGACAGCGCA'
, 'CACCAC', 'GCGGTCCTGGCG', 'ATGTGTA', 'TGATCGCTAGT', 'GCG', 'TTCGAAGA
GAAGCTT', 'GTGGCGGTG', 'CATAC', 'AGGA', 'TATTAT', 'GTATTATGTGTATTATG',
'TTGCCCCGGCCCCGTT', 'CGC', 'TAGGTGTGGAT', 'TTT', 'CTCTC', 'TGCAAAACGT'
, 'ATTATGTGTATTA', 'CAATTTCTTTCTTTAAC', 'ACTGGGTCA', 'ATGCTCGTA', 'TGG
GT', 'GCGACAGCG', 'CGTGCGGTCCTGGCGTGC', 'CGCGACAGCGC', 'GGG', 'GATTAG'
, 'TAT', 'CTC', 'CGGTCCTGGC', 'CAAC', 'TGT', 'GTAGGTATTATGTGTATTATGGAT
G', 'TGCTCGT', 'CGGC', 'GTATG', 'TCGCT', 'GAAGAGAAG', 'CACAATTTCTTTCTT
TAACAC', 'TGCGTGCGGTCCTGGCGTGCGT', 'GCGAAGGAAGCG', 'ATA', 'GATCGCTAG',
'TTATATT', 'GAAG', 'TCACCACT', 'TAAAAT', 'GAGGCAACGGAG', 'TCGAAGAGAAGC
T', 'CGTGGCGGTGC', 'CGACAGC', 'GGAGAGTCGCGCGCTGAGAGG', 'GACAG', 'TACCA
T', 'TATGTGTAT', 'GGCGG', 'AGGTATTATGTGTATTATGGA', 'TTCTT', 'CAC', 'TC
TTTCT', 'TCGCGCGCT', 'ATTTA', 'AAGAA', 'ATAGGTGTGGATA', 'CTAAACGCGACAG
CGCAAATC', 'AGTCGCGCGCTGA', 'TTTTGCCCCGGCCCCGTTTT', 'CCC', 'AAATAGGTGT
GGATAAA', 'CTGCGAAGGAAGCGTC', 'GATTTAG', 'GCCG', 'GGTCACTGG', 'AAA', '
GCGTGCGGTCCTGGCGTGCG', 'TTTGCCCCGGCCCCGTTT', 'TTTT', 'TCCT', 'AAATTCGA
AGAGAAGCTTAAA', 'TGGCGGT', 'AGGTGTGGA', 'GAGAG', 'CAAAAC', 'TAAAATTCGA
AGAGAAGCTTAAAAT', 'ACTCA', 'CATACCATAC', 'ATGCGTGCGGTCCTGGCGTGCGTA', '
GCCCCGGCCCCG', 'GAG', 'TCACT', 'ACA', 'CTTTATATTTC', 'GTGCGGTCCTGGCGTG
', 'AAGGAA', 'ACCA']
```

In [282]:

```
print("Number of Unique Palindromes greater than three is: ",len(allPalThree))
```

```
Number of Unique Palindromes greater than three is:  177
```

In [283]:

```
#Part B
print("The longest palindrome found: ", max(allPalThree, key=len))
```

```
The longest palindrome found:  CGTAGGTATTATGTGTATTATGGATGC
```

Problem 4 (25 points)

Part A) What is the most common letter found in the words listed below? (10 points)

Part B) What letters *don't* appear (i.e, are missing from the alphabet a-z) in the words in the wordList below? (15 points)

In [146]:

```
wordList = ["hello", "class", "we", "hope", "you", "enjoyed", "the", "semester", "ca
```

In [176]:

```
#Part A
#Source: https://www.hackerrank.com/challenges/collections-counter/problem

from collections import Counter

#Convert all the words to a single string
test = "".join(wordList)

#Convert the single string to a list of letters
wordChar = list(test)

#Use the Counter library to sort and print the results
print(Counter(wordChar))
```

```
Counter({'e': 12, 'o': 7, 'l': 6, 's': 6, 'h': 4, 'c': 4, 'a': 4, 'y':
4, 't': 4, 'u': 3, 'r': 3, 'p': 2, 'n': 2, 'm': 2, 'w': 1, 'j': 1, 'd'
: 1, 'v': 1, 'i': 1, 'b': 1})
```

As seen above, **"e" is the most common letter**!!

In [187]:

```
#Part B
#Source: https://www.geeksforgeeks.org/python-find-missing-additional-values-two-li
import string
az = string.ascii_lowercase #get a through z as a string
azChar = "".join(az)
print("Missing values:", set(azChar).difference(wordChar))
```

```
Missing values: {'k', 'x', 'g', 'z', 'f', 'q'}
```

Problem 5 (25 points)

You are provided the following 3 DNA sequences:

Sequence 1: ATCGTACGATCGATCGATCGCTAGACGTATCA

Sequence 2: actgatcgacgatcgatcgatcacgactd

Sequence 3: ACTGAC-ACTGT--ACTGTA----CATGTG

You have received some messy Sequence DNA data and will need to clean it up. Remove non-DNA characters (DNA consists of `ACTG` or `actg` characters) (5 points). Convert all data to uppercase and combine all 3 sequences into 1 sequence (5 points). Print the length of your new sequence and finally write the length of your new sequence and your new sequence in a file called `mysequence.txt'. Include your new txt file in the zip file submission. Also list the length of your new sequence and the new sequence in a cell. (10 points).

Note: You need to write python code to obtain the desired result. You may not do this by hand, although you are free to verify your results by hand.

An example is provided here for illustrative purposes:

Sequence 1: ACT Sequence 2: tca Sequence 3: C-T

You will output in a notebook cell: Your new DNA sequence: ACTTCACT and the Length of the DNA sequence = 8

File: mysequence.txt will contain: Your new DNA sequence: ACTTCACT and the Length of the DNA sequence = 8

Fun FACT: The total length of DNA in 1 human equals 70 round trips from the Earth to the Sun.

Problem 6 (25 points)

Flatten out the following list `mylist`

Note: ***Flattening out*** a structure means to convert all contained substructures into its most basic constituent elements so that the structure only contains these basic constituent elements. You need to write a python program to obtain the desired result listed in the example. You may not do this by hand, although you are free to verify your results by hand. (25 points)

Illustrative example:

- non_flat = [[1,2,3], [4,[5,6]], [7,8,'hello world']]
- flat = [1,2,3,4,5,6,7,8,'hello world']

In [188]:

```
mylist = [[0, 10, 5, 8, 0, 1, 9],
 ['not'],
 ['to'],
 [5, 8, 6, 7, 7, 3, 2],
 [10, 10, 3, 9, 6, 5, 7, 0, 10, 5],
 ['to'],
 [1],
 [2, 8],
 [7, 6, 10, 2, 6, 6, 2, 6],
 [0],
 [2, 9, 2, 1, 2, 2, 9, 7, 3, 6],
 [1, 6, 4, 0, 8],
 ['not'],
 [0, 4, 4, 7, 10, 2, 8, 8, 6, 4],
 ['to']]
```

In [189]:

```
flat_list = [item for sublist in mylist for item in sublist]
```

In [202]:

```
print ('Flattened List: ', flat_list)
```

```
Flattened List:  [0, 10, 5, 8, 0, 1, 9, 'not', 'to', 5, 8, 6, 7, 7, 3,
2, 10, 10, 3, 9, 6, 5, 7, 0, 10, 5, 'to', 1, 2, 8, 7, 6, 10, 2, 6, 6,
2, 6, 0, 2, 9, 2, 1, 2, 2, 9, 7, 3, 6, 1, 6, 4, 0, 8, 'not', 0, 4, 4,
7, 10, 2, 8, 8, 6, 4, 'to']
```

Extra Credit (bonus points) 20 points

Tackle the following `mylist2` example by flattening it similar to problem 6.

It might help to put keyboard focus on the cell below, and use the arrow keys to successively highlight closing brackets and allow python to highlight *find* the corresponding opening bracket, to help you understand the structure of the list.

In [193]:

```
mylist2 = [[[0, 10, 5, 8], 0, 1, 9],
 ['not'],
 ['to'],
 [5, 8, 6, 7, 7, 3, 2],
 [10, 10, [3, 9, [6, [5, [7]]]], 'or', 10, 5]],
 ['to'],
 [[1]],
 [2, 8],
 [[7, 6, 10, [2, 'be', 6], 2, 6],
 [0],
 [2, 9, 2, 1, 2, 2, 9, 7, 3, 6]],
 [1, 6, 4, 0, 8],
 ['not'],
 [0, 4, 4, 7, 10, [2], [[8]], 8, [[[6]]], 4],
 ['to']]
```

In [284]:

```
#Source: https://www.geeksforgeeks.org/python-convert-a-nested-list-into-a-flat-lis
# output list
flat_list2 = []

# function used for removing nested lists in python.
def removeNestings(l):
    for i in l:
        if type(i) == list:
            removeNestings(i)
        else:
            flat_list2.append(i)

removeNestings(mylist2)
print ('Flattened List 2: ', flat_list2)
```

```
Flattened List 2:  [0, 10, 5, 8, 0, 1, 9, 'not', 'to', 5, 8, 6, 7, 7,
3, 2, 10, 10, 3, 9, 6, 5, 7, 'or', 10, 5, 'to', 1, 2, 8, 7, 6, 10, 2,
'be', 6, 2, 6, 0, 2, 9, 2, 1, 2, 2, 9, 7, 3, 6, 1, 6, 4, 0, 8, 'not',
0, 4, 4, 7, 10, 2, 8, 8, 6, 4, 'to']
```