Assignment 3 – PYTHON LISTS

Due: 4:30pm, May 8
Worth: 2% of your final mark
Topic: Python lists
This assignment is marked out of 20

For Assignment 3, a program containing the skeleton and testing code for the 7 Assignment 3 questions has been posted on the CompSci 101 Assignments website. Download this program from the CompSci 101 assignments website:

https://www.cs.auckland.ac.nz/courses/compsci101s1c/assignments/

Develop the solution to each function in the program. Once you are happy that your function executes correctly, submit the whole function to CodeRunner:

https://coderunner2.auckland.ac.nz/moodle/

When you press the SUBMIT button in CodeRunner, you will receive immediate feedback telling you if you have passed all the tests for the question. You can submit as many times as you like. You need to submit one function at a time.

#Assignment 3 Questions -

```
#-------------------------------------------------
# 1111111111111111111111111111111111111111111111
# Returns the average of a list of numbers (excluding
# all zeroes, all negative numbers and excluding the
# minimum and maximum positive numbers) - 3 marks
#-------------------------------------------------

"""
Define the get_funny_average() function which is passed a list of numbers as
a parameter and returns the average of some of the numbers in the parameter
list. The function returns the average of the remaining numbers (rounded to
1 decimal place) after all the following have beeen excluded from the
parameter list of numbers (if they exist in the list):

   • all zeroes,
   • all negative numbers,
   • the smallest positive number
and
   • the largest positive number,

For example, the following code:

print("1.  Funny average: ", get_funny_average([ 3, 2, 0, 25, 1]))
print("2.  Funny average: ", get_funny_average([-6, -32, 2, 0, -51, 1, 0,
0]))
print("3.  Funny average: ", get_funny_average([56, 32, 2, 22, 22]))
print("4.  Funny average: ", get_funny_average([-56, -3, 0, -21, 0, 0, 5]))
print("5.  Funny average: ", get_funny_average([56, 3, 2, 0, 251, 1, 41,
22]))
print("6.  Funny average: ", get_funny_average([-56, -3, 2, 0, -251, 1, -41,
0]))
print("7.  Funny average: ", get_funny_average([]))

prints:

1.  Funny average:  2.5
2.  Funny average:  0
3.  Funny average:  25.3
4.  Funny average:  0
5.  Funny average:  24.8
6.  Funny average:  0
7.  Funny average:  0
"""
```

```
def get_funny_average(numbers):
   return 0
```

```
#-------------------------------------------------
# 2222222222222222222222222222222222222222222222
# Returns the score from a memory game.
# The strategy is to remove the number that has
# been in the memory the longest time. - 3 marks
#-------------------------------------------------
"""
A memory game  is played (and scored) as follows:
Random numbers between 0 and 10 (zero inclusive) are called out one at a
time. In this memory game the player can remember a maximum of 5 previously
called out numbers. If the called number is already in the player's memory,
a point is added to the player's score.  If the called number is not in the
player's memory, the player adds the called number to his memory, first
removing another number if his memory is full.  In our simulation of this
game, the number which is removed from the player's memory is the number
that has been in the player's memory the longest time. For example, if the
random numbers are [3, 4, 3, 0, 7, 4, 5, 2, 1, 3], the game proceeds as
follows:

Called number 3:  Score: 0, Numbers in memory: [3]
Called number 4:  Score: 0, Numbers in memory: [3, 4]
Called number 3:  Score: 1, Numbers in memory: [3, 4]
Called number 0:  Score: 1, Numbers in memory: [3, 4, 0]
Called number 7:  Score: 1, Numbers in memory: [3, 4, 0, 7]
Called number 4:  Score: 2, Numbers in memory: [3, 4, 0, 7]
Called number 5:  Score: 2, Numbers in memory: [3, 4, 0, 7, 5]
Called number 2:  Score: 2, Numbers in memory: [4, 0, 7, 5, 2]
Called number 1:  Score: 2, Numbers in memory: [0, 7, 5, 2, 1]
Called number 3:  Score: 2, Numbers in memory: [7, 5, 2, 1, 3]

Complete the get_memory_score() function which is passed a list of random
numbers as a parameter and returns the final score using the algorithm
described above.  For example, the following code:

print("1.  Score:", get_memory_score([3, 4, 1, 6, 3, 3, 9, 0, 0, 0]))
print("2.  Score:", get_memory_score([1, 2, 2, 2, 2, 3, 1, 1, 8, 2]))
print("3.  Score:", get_memory_score([2, 2, 2, 2, 2, 2, 2, 2, 2]))
print("4.  Score:", get_memory_score([1, 2, 3, 4, 5, 6, 7, 8, 9]))
random_nums5 = [7, 5, 8, 6, 3, 5, 9, 7, 9, 7, 5, 6, 4, 1, 7, 4, 6, 5, 8, 9,
                           4, 8, 3, 0, 3]
print("5.  Score:", get_memory_score(random_nums5))

prints:

1.  Score: 4
2.  Score: 6
3.  Score: 8
4.  Score: 0
5.  Score: 10
"""
```

```
def get_memory_score(numbers):
   return 0
```

```
#-------------------------------------------------
# 3333333333333333333333333333333333333333333333
# Returns the number from a list of numbers to test
# which is the most recent number in a separate
# list of numbers. - 3 marks
#-------------------------------------------------
"""
Define the get_most_recent() function which is passed two lists of numbers
as parameters:

  • a list of numbers which are in order from the least recent to the most
    recent, i.e., the number at the end of the list is the most recent,
and
  • a list of numbers to test - the numbers of this list may or may not be
    elements of the first parameter list.

This function returns the number from the "list of numbers to test" which
occurred most recently in the first parameter list (i.e., is closer to the
end of the list).  If none of the numbers in the "numbers to test" list
occurred in the first parameter list, the function should return -1.  For
example, the following code:

print("1.", get_most_recent([0, 1, 2, 0, 3, 4, 1], [2, 0, 3]))
print("2.", get_most_recent([0, 1, 2, 0, 3, 4, 1], [0, 7, 2]))
print("3.", get_most_recent([0, 1, 2, 8, 9, 0, 3, 4, 6], [1, 9, 2, 8]))
print("4.", get_most_recent([4, 1, 4, 5, 4, 1], [0, 7, 3]))
print("5.", get_most_recent([8, 1, 2, 0, 8, 4, 1], [8, 7, 3]))
print("6.", get_most_recent([], [8, 1, 0, 3]))
numbers_in_order = [1, 1, 1, 0, 1, 0, 2, 2, 1, 2, 0, 1, 2, 0, 3, 4, 1, 2, 4,
                                                 0, 3, 8, 8, 5, 5]
print("7.", get_most_recent(numbers_in_order, [1, 0, 3, 4] ))
numbers_in_order = [1, 2, 2, 2, 2, 3, 1, 3, 8, 0]
print("8.", get_most_recent(numbers_in_order, [1, 8, 2, 3, 4, 2]))

prints:

1. 3
2. 0
3. 9
4. -1
5. 8
6. -1
7. 3
8. 8
"""
```

```
def get_most_recent(numbers_in_order, numbers_to_test):
    return 0
```

```
#-------------------------------------------------
# 4444444444444444444444444444444444444444444444
# Returns True if the parameter string is a
# legitimate code, the function returns
# False otherwise - 3 marks
#-------------------------------------------------
"""
```

Define the is_legitimate_code() function which is passed a string as a parameter. The function returns a boolean indicating whether the parameter string is a legitimate code or not. A legitimate code is a string made up of one letter followed by one or more digits (can also include spaces before, between or after the digits). The first three lines of code inside the function should be:

```
code_letters = ["S", "B", "N", "T", "P"]
min_for_each_letter = [1, 3, 4, 0, 3] #inclusive
max_for_each_letter = [7, 9, 6, 7, 5] #inclusive
```

where:

- code_letters is the list of code letters which are legitimate for the first letter of the code string,
- min_for_each_letter is a list which contains the minimum number (inclusive) for each digit following that letter,
- max_for_each_letter is a list which contains the maximum number (inclusive) for each digit following that letter.

For example, the third element of the code_letters list is the letter 'N', the corresponding third element of the min_for_each_letter list is 4 and the corresponding third element of the max_for_each_letter list is 6. This indicates that the code digits which follow the letter 'N' can be any number made up of the digits 4, 5 or 6. The number part of a legitimate code string can also contain any number of spaces.

Note: The number part of a parameter code string to be tested could contain an alphabetic character thus making the code not legitimate. You will find it useful to use the isdigit() method which returns True if a string is a digit, False otherwise.

For example, the following code:

```
print("1.", is_legitimate_code('B747346'))
print("2.", is_legitimate_code('N  444  454'))
print("3.", is_legitimate_code('T 400 4854'))
print("4.", is_legitimate_code('S  444S454'))
print("5.", is_legitimate_code('P  '))
print("6.", is_legitimate_code('T  0  '))
```

prints:

```
1. True
2. True
3. False
4. False
5. False
6. True
"""
```

def is_legitimate_code(code_string):

```
    code_letters = ["S", "B", "N", "T", "P"]
    min_for_each_letter = [1, 3, 4, 0, 3] #inclusive
    max_for_each_letter = [7, 9, 6, 7, 5] #inclusive

    return True
```

```
#-------------------------------------------------
# 55555555555555555555555555555555555555555555555
# Returns the longest word (which has six or more
# characters) from the parameter list - 3 marks
#-------------------------------------------------
"""
Define the get_longest_word() function which is passed a list of strings as
a parameter. The function returns the word in the list which has the most
characters (i.e., the longest word) BUT only words with six or more
characters are considered. If two or more words in the list have the same
number of characters as the longest word, the function should return the
last word from the start of the list which has the most characters. If the
parameter list is empty or if there are no words in the list with six or
more characters, the function should return the empty string.  For example,
the following code:

print("1.", get_longest_word(["Melissa", "Jessie", "Kath", "Amity",
                                                        "Raeann"]))
print("2.", get_longest_word(["Jo", "Jessie", "Penelope", "Jin", "Raeann",
                                                        "Pamelita"]))
print("3.", get_longest_word(["Alan", "Jess", "Amity", "Rosalie",
                                                        "Rosetta"]))
print("4. ", "***", get_longest_word(["Jo", "Jai", "Jen", "Jing", "Joey",
                                                    "Jess"]), "***", sep = "")
print("5. ", "***", get_longest_word([]), "***", sep = "")
print("6.", "***" + get_longest_word([""]) + "***")

prints:

1. Melissa
2. Pamelita
3. Rosetta
4. ******
5. ******
6. ******
"""
def get_longest_word(word_list):
   return ""
```

```
#---------------------------------------------------
# 6666666666666666666666666666666666666666666666666
# Remove triplets made up of three sequential
# identical elements - 3 marks
#---------------------------------------------------
"""
Define the remove_triplets() function which is passed a list of integers as
a parameter. The function removes all triplets from the list (i.e., removes
any three elements in the list which are exactly the same and are in
sequence).  For example, the following code:

a_list = [6, 6, 6, 7, 6, 6, 6, 3, 3, 3, 8, 8, 8, 3]
remove_triplets(a_list)
print("1.", a_list)

a_list = [6, 6, 6, 7, 6, 6, 6, 6, 6]
remove_triplets(a_list)
print("2.", a_list)

a_list = [6, 6, 6, 7, 6, 6, 4, 3, 3, 3, 8, 8, 8, 3]
remove_triplets(a_list)
print("3.", a_list)

a_list = [1, 1, 1, 4, 4, 4, 1, 1, 1]
remove_triplets(a_list)
print("4.", a_list)

a_list = [1, 1, 2, 1, 2, 2]
remove_triplets(a_list)
print("5.", a_list)

prints:

1. [7, 3]
2. [7, 6, 6]
3. [7, 6, 6, 4, 3]
4. []
5. [1, 1, 2, 1, 2, 2]
"""
```

```
def remove_triplets(a_list):
    pass
```

```
#-------------------------------------------------
# 77777777777777777777777777777777777777777777777
# Score a hand of random dice throws - 2 marks
#-------------------------------------------------
"""
In a dice rolling game a hand is made up of any number of random dice throws
and is valued in the following way:

  • In this game a run is a sequence of dice values starting from 1, e.g.,
    123, 12345, 1234, 1.
  • Each dice which is part of a run of dice starting from a 1 has a value
    which is equivalent to the dice number. The value of any dice which is
    part of a run is added to the hand score.
  • If there is no 1 in a hand of dice, the score for the whole hand is 0.
  • A hand of dice can contain more than one run.

Study the following five example hands of dice and their corresponding
valuation.  Make sure you understand how the hands are valued:

[5, 3, 2, 5, 4, 5, 6, 4, 3] has value 0
[3, 4, 1, 5, 3, 1, 4, 6] has value 2 (contains one run with just the dice
[1] and a second run with just [1])
[5, 3, 2, 2, 6, 4, 5, 1, 4] has value 21 (contains one run with the dice [1,
2, 3, 4, 5, 6])
[2, 1, 1, 1, 2, 3, 3, 1, 3, 2] has value 19 (contains three separate runs
with the dice [1, 2, 3] and a second run with the dice [1]
[3, 4, 1, 5, 2, 1, 5, 1, 2, 3, 4, 6] has value 37 (contains one run with the
dice [1, 2, 3, 4, 5, 6], a second run with [1, 2, 3, 4, 5] and a third run
with the dice [1])

Complete the get_hand_score() function which is passed a list of dice throws
and returns the value of the hand according to the rules described above.
"""

def get_hand_score(list_of_dice):
    return 0
```