```
#-------------------------------------------------
# 1111111111111111111111111111111111111111111111
# Returns a string containing all the last three
# letters from the parameter list of strings.
#-------------------------------------------------
```
Define the get_last_three_letters() function which is passed a list of strings as a parameter. The function returns a string made up of the concatenation of the last three letters of each word from the parameter list which has at least three characters. The string returned by the function should be in lowercase characters. If the parameter list is an empty list, the function should return an empty string. For example, the following code:

```
print("1.", get_last_three_letters(["Jess", "Cain", "Amity", "Raeann"]))
print("2.", get_last_three_letters(["CAIn", "JessiE", "O", "ROBERT", "Geoffrey",
"Li", "B"]))
print("3.", "***" + get_last_three_letters([]) + "***")
print("4.", "***" + get_last_three_letters(["A", "E", "O"]) + "***")
```

prints:

```
1. essainityann
2. ainsieertrey
3. ******
4. ******
```

```
#-------------------------------------------------
# 2222222222222222222222222222222222222222222222
# Returns the average of a list of numbers (excluding
# all zeroes, all negative numbers and excluding the
# minimum and maximum positive numbers)
#-------------------------------------------------
```
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 been excluded from the parameter list of numbers (if they exist in the list):

- all zeroes,
- all negative numbers,
- the two smallest positive numbers

For example, the following code:

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

```
print("7.  []:                                ", get_funny_average([]))

prints:

1.  [ 3, 2, 0, 25, 1]:                 14.0
2.  [-6, -32, 2, 0, -51, 1, 0, 0]:     0.0
3.  [56, 32, 2, 22, 22]:               36.7
4.  [-56, -3, 0, -21, 0, 6, 5]:        0.0
5.  [56, 3, 2, 0, 251, 1, 41, 22]:     74.6
6.  [-56, -3, 2, 0, -251, 1, -41, 0]:  0.0
7.  []:                                0.0
#-----------------------------------------------
# 333333333333333333333333333333333333333333333333
# Returns a new list containing all the numbers
# from the parameter list which do not contain
# the digit 9
#-----------------------------------------------
```
Define the get_list_nums_without_9() function which is passed a list of numbers as a parameter. The function returns a new list containing all the numbers (integers) from the parameter list which do not contain the digit 9. If the parameter list is empty or if all of the numbers in the parameter list contain the digit 9, the function should return an empty list. For example, the following code:

```
print("1.", get_list_nums_without_9([589775, 677017, 34439, 48731548, 782295632,
181967909]))
print("2.", get_list_nums_without_9([6162, 29657355, 5485406, 422862350, 74452,
480506, 2881]))
print("3.", get_list_nums_without_9([292069010, 73980, 8980155, 921545108,
75841309, 6899644]))
print("4.", get_list_nums_without_9([]))

nums = [292069010, 73980, 8980155, 21545108, 7584130, 688644, 644908219, 44281,
3259, 8527361, 2816279, 985462264, 904259, 3869, 609436333, 36915, 83705, 405576,
4333000, 79386997]
print("5.", get_list_nums_without_9(nums))

prints:

1. [677017, 48731548]
2. [6162, 5485406, 422862350, 74452, 480506, 2881]
3. []
4. []
5. [21545108, 7584130, 688644, 44281, 8527361, 83705, 405576, 4333000]
#-----------------------------------------------
# 444444444444444444444444444444444444444444444444
# Returns the score from a memory game.
# The strategy is to remove the number that has
# been in the memory the longest time.
#-----------------------------------------------
```
A memory game is played (and scored) as follows:

Random numbers 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
#-------------------------------------------------
# 55555555555555555555555555555555555555555555555
# Remove from the parameter list of words all words
# which are in the parameter sentence as well as
# in the list of words
#-------------------------------------------------
```

Define the alter_the_list() function which is passed a string of text and a list of words as parameters. The function removes any word from the parameter list of words which is a separate word in the parameter string of text. The string of text should be converted to lower case before you do any checking as the elements of the parameter list of words are all in lower case. Note that there is no punctuation in the parameter string of text. Note that each word in the parameter list of words is unique, i.e., it occurs only once. For example, the following code:

```
word_list =  ["a", "is", "bus", "on", "the"]
```

```
alter_the_list("A bus station is where a bus stops  A train station is where a
train stops  On my desk I have a work station", word_list)
print("1.", word_list)

word_list =  ["a", 'up', "you", "it", "on", "the", 'is']
alter_the_list("It is up to YOU", word_list)
print("2.", word_list)

word_list =  ["easy", "come", "go"]
alter_the_list("Easy come easy go go go", word_list)
print("3.", word_list)

word_list =  ["a", "is", "i", "on"]
alter_the_list("", word_list)
print("4.", word_list)

word_list =  ["a", "is", "i", "on", "the"]
alter_the_list("May your coffee be strong and your Monday be short", word_list)
print("5.", word_list)

prints:

1. ['the']
2. ['a', 'on', 'the']
3. []
4. ['a', 'is', 'i', 'on']
5. ['a', 'is', 'i', 'on', 'the']
#-------------------------------------------------
# 6666666666666666666666666666666666666666666666
# Returns a list of numbers with all the odd numbers
# at the front of the list (sorted) and all the even
# numbers at the back (sorted)
#-------------------------------------------------
Define the get_evens_at_back() function which is passed a list of integers as a
parameter. The function returns a new list with all the odd numbers from the
parameter list (in sorted order) at the front and all the even numbers from the
parameter list (in sorted order) at the back of the list.  If the parameter list
is empty, the function should return an empty list. For example, the following
code:

print("1.", get_evens_at_back([-1, 2, -3, 4, -2, 3, 5]))
print("2.", get_evens_at_back([1, 2, -3, 4, 7, 4, -6, 3, -1]))
print("3.", get_evens_at_back([-4, -2, 6, 8, 6, 2]))
print("4.", get_evens_at_back([-3, -1, 3, 1, 7, 9]))
print("5.", get_evens_at_back([]))

prints:

1. [-3, -1, 3, 5, -2, 2, 4]
2. [-3, -1, 1, 3, 7, -6, 2, 4, 4]
3. [-4, -2, 2, 6, 6, 8]
4. [-3, -1, 1, 3, 7, 9]
5. []
#-------------------------------------------------
```

```
# 77777777777777777777777777777777777777777777
# Returns True if the parameter string is a
# valid code, the function returns
# False otherwise
#-----------------------------------------------
```

Define the is_a_valid_code() function which is passed a string as a parameter. The function returns a boolean indicating whether the parameter string is a valid code or not. A valid 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 valid for the first letter of the code string,

- min_for_each_letter is a list which contains the minimum possible number (inclusive) for each digit following that letter,

- max_for_each_letter is a list which contains the maximum possible 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 follows the letter 'N' can be any number made up of the digits 4, 5 or 6. The number part of a valid 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 valid. You will find it useful to use the string method, isdigit(), which returns True if a string is a digit, False otherwise.

For example, the following code:

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

prints:

1. True
2. True
3. False
4. False
5. False

6. True
```
#-----------------------------------------------
# 88888888888888888888888888888888888888888888
# Score a hand of random dice throws
#-----------------------------------------------
```
In a dice rolling game a player's hand is made up of any number of random dice rolls 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 score for the hand.
• 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_dice_score() function which is passed a list of dice rolls and returns the value of the hand according to the rules described above.
```
#-----------------------------------------------
# The end :-)
#-----------------------------------------------
```