```
"""Assignment 3 functions S1, 2020."""

#--------------------------------------------------
# 11111111111111111111111111111111111111111111111111
# Returns the list of numbers which are not exactly
# divisible by 3 from the parameter list of
# numbers
#--------------------------------------------------
"""
Define the get_numbers() function which is passed a list of integers
as a parameter. The function returns a new list which contains all the
odd numbers which are not multiples of three (i.e. NOT exactly divisible
by 3). The numbers  should be in the same order as they appear in the
parameter list.

Note: if the parameter list is empty the function should
return an empty list. For example,

    list2 = get_numbers([23, 3, 6, 5, 12, 9, 7, 4])
    print(list2)

    print(get_numbers([87, 77, 49, 21, 4, 80, 51]))

prints:

[23, 5, 7]
[77, 49]
"""
#--------------------------------------------------
# 22222222222222222222222222222222222222222222222222
# Returns the average of a list of numbers (excluding
# all zeroes and the minimum and maximum 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 the nearest whole number) after the following have been
excluded from the parameter list of numbers (if they exist in the list):

• all zeroes,
• all smallest number,
• the largest number.

Note: if there are no valid numbers in the parameter list, the function returns 0.

For example, the following code:

    print(get_funny_average([ 3, 12, 0, 25, 1]))
    print(get_funny_average([-6, -32, 2, 0, -51, 1, 0, 0]))
    print(get_funny_average([56, 0, 2, 0, 22]))

prints:
```

```
8
-12
22
"""
#--------------------------------------------------
# 333333333333333333333333333333333333333333333333
# Returns a Python list containing the average fail
# mark and the average pass mark.
#--------------------------------------------------
"""
Define the get_fail_pass_average() function which is passed a list of integers as
a parameter where each
integer represents a mark out of 100. The function returns a Python list made up
of the average of all the
marks which are less than 50, followed by the average of all the marks which are
50 or more (both averages
are rounded to the nearest whole number). If there are no fail marks then the
average should be set to -1
and if there are no pass marks then the average pass mark should be set to -1.

For example, the following code:

    print("1.", get_fail_pass_average([63, 65, 33]))
    print("2.", get_fail_pass_average([63, 62, 100, 100]))
    print("3.", get_fail_pass_average([33, 42, 20, 10]))

prints:

1. [33, 64]
2. [-1, 81]
3. [26, -1]
"""
#--------------------------------------------------
# 444444444444444444444444444444444444444444444444
# Returns a list containing every third even number
# element of the parameter list starting
# from the last element.
#--------------------------------------------------
"""
Complete the get_every_third_even_element() function which is passed a list of
integers as a parameter. The function returns a list which contains every third
element from the parameter list which is an even number, starting from the last
element of the parameter list down to the first element.

Notes
• the original list should not be changed in any way,
• if the parameter list is empty, the function returns an empty list.

For example:

    list1 = [23, 12, 6, 5, 8, 9, 7, 4]
    print(get_every_third_even_element(list1))
    print(list1)
```

```
prints:

[4, 8, 12]
[23, 12, 6, 5, 8, 9, 7, 4]
"""
#--------------------------------------------------
# 55555555555555555555555555555555555555555555555
# Returns a list with the totals of each pair of
# of elements of the parameter list.
#--------------------------------------------------
"""
Define the get_sequencial_nums_sums() function which is passed a list of integers
as a parameter.  The function returns a new list where the first element is the
sum of the first two elements of the parameter list, the second element is the sum
of the next two elements of the parameter list, and so on.

Notes
• if the parameter list has an odd number of elements, the last element of the
returned list is the same as the last element of the parameter list,
• if the list passed to the function is empty, the function returns an empty list.


For example:

    list1 = [3, 3, 2, 3, 4, 3, 5]
    print(list1, "=>", get_sequencial_nums_sums(list1))

prints:

[3, 3, 2, 3, 4, 3, 5] => [6, 5, 7, 5]
"""
#--------------------------------------------------
# 66666666666666666666666666666666666666666666666
# Remove triplets made up of three sequential
# identical elements
#--------------------------------------------------
"""
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
sequential elements in the list which are exactly the same).  For example:

    a_list = [6, 6, 6, 7, 6, 6, 6, 3, 3, 3, 8, 8, 8, 3]
    print(a_list, end = " => ")
    remove_triplets(a_list)
    print(a_list)

    a_list = [6, 6, 6, 7, 6, 6, 6, 6, 6]
    print(a_list, end = " => ")
    remove_triplets(a_list)
    print(a_list)

prints:

[6, 6, 6, 7, 6, 6, 6, 3, 3, 3, 8, 8, 8, 3] => [7, 3]
```

```
[6, 6, 6, 7,   6, 6] => [7, 6, 6]
"""
#---------------------------------------------------
# 77777777777777777777777777777777777777777777777777
# Returns True if the parameter string is a
# valid date, otherwise returns False
#---------------------------------------------------
"""
Define the is_a_valid_date() function which is passed a string as a parameter. The
function returns a boolean indicating whether the parameter string is a valid date
or not. The first two lines of the function are:

month_names = ["January", "February", "March", "April", "May", "June", "July",
"August", "September", "October", "November", "December"]
days_in_month = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]

where month_names is a list of valid month names, and days_in_month is a list
which contains the maximum day number for each month (note that February is set to
28 days), e.g. the third element of the month_names list is the month of March and
the corresponding third element of the days_in_month list indicates that there are
31 days in March.

Notes
• The parameter string can contain any number of spaces but the month name must
always start at the first non-space character from the beginning of the string.

• The day number part of the date string to be tested could contain alphabetic
characters thus making the date string invalid. You will find it useful to use the
string method, isdigit(), which returns True if a string is made up of digits,
False otherwise.

 For example:

    print("1.", is_a_valid_date("January     21"))
    print("2.", is_a_valid_date("Auust 3"))
    print("3.", is_a_valid_date(" June   15B "))
    print("4.", is_a_valid_date("February 0"))
    print("5.", is_a_valid_date(" December 3K1"))

prints:

1. True
2. False
3. False
4. False
5. False
"""
#---------------------------------------------------
# 88888888888888888888888888888888888888888888888888
# 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 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 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])

Define 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.

IMPORTANT:  your code should not change the parameter list (i.e. you need to make
a copy of the parameter list and manipulate the copy).
"""
#---------------------------------------------------
# The end :-)
#---------------------------------------------------
```