Lecture 18 – Last exercise from Lecture 17, is operator vs. "==", list objects are mutable, Assignment 3
Learning outcomes

- understand the difference between '==' and 'is'
- understand that list objects are mutable
- Assignment 3 Questions
From Lecture 17, Slide 18 - Exercise
Complete the `remove_multiples()` function which removes all the elements in the parameter list, `number_list`, which are multiples of the parameter, `multiples_of`.

```python
def remove_multiples(number_list, multiples_of):

def main():
    numbers = [25, 5, 9, 10, 15, 8]
    print(numbers)
    remove_multiples(numbers, 5)  # remove multiples of 5
    print("Numbers left", numbers)

main()
```

```
[25, 5, 9, 10, 15, 8]
Numbers left [9, 8]
```
The `==` operator is used to test if two objects contain the same information.

The `is` operator is used to test if two variables reference (point to) the same object.

```
word1 = "sweet"
word2 = word1
print("1.", word1 == word2)
print("2.", word1 is word2)
word2 = word2.upper()
word2 = word2.lower()
print("3.", word1 == word2)
print("4.", word1 is word2)
```
Strings are "immutable", i.e., the characters in a string object cannot be changed. Whenever a string is changed in some way, a new string object (with a new memory address) is created.

```python
word1 = "sweet"
word2 = word1
print("1.", word1, word2)
print("2.", word1 is word2)

word2 = word2 + " dumpling"
print("3.", word1, word2)
print("4.", word1 is word2)
```

1. sweet sweet
2. True
3. sweet sweet dumpling
4. False
Lists are "mutable", i.e., the elements in a list object can be updated and adjusted.

```python
list1 = [10, 20, 30, 40, 50]
list2 = [1, 5]  # A
print("1.", list1)
print("2.", list2)
print("3.", list1 is list2)

list2 = list1  # B
print("4.", list1 is list2)

list1[3] = 99
list2[1] = 3  # C
print("5.", list1)
print("6.", list2)
print("7.", list1 is list2)
```

1. [10, 20, 30, 40, 50]
2. [1, 5]
3. False
4. True
5. [10, 3, 30, 99, 50]
6. [10, 3, 30, 99, 50]
7. True
CompSci 101 Assignment 3

Due: 4:30pm, May 16

Worth: 3% of your final mark

Topic: lists

This assignment is marked out of 30
Coderunner2 Assignments

CompSci 101 has 5 assignments, in total worth 16.5% of your final mark. For two of these five assignments (a total of 6%), you are required to use the CodeRunner2 tool.

The CodeRunner2 tool is designed to help you practise, by presenting you with a set of coding exercises. CodeRunner2 is part of the Moodle learning system:

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

Information about using CodeRunner2 is available on the CompSci 101 assignments web page:
https://www.cs.auckland.ac.nz/courses/compsci101s1c/assignments/

Step 1: make sure you can log into CodeRunner2
Assignment 3 – Complete 7 functions

For Assignment 3, I have posted a program containing the skeleton and testing code for the 7 assignment questions. 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 your program.

Once you are happy that your function executes correctly, submit the whole function to Coderunner2 and press the "CHECK" button. You will receive immediate feedback from Coderunner2 telling you if you have passed the tests for that question. You can submit as many times as you like. You need to submit one function at a time.
A3 Q1 - get_funny_average ()

**parameter** - a list of numbers

**returns** – the average (to one decimal place) of just the positive elements excluding the smallest two positive numbers.

```python
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([]))
```

<table>
<thead>
<tr>
<th></th>
<th>get_funny_average()</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[ 3, 2, 0, 25, 1] :</td>
</tr>
<tr>
<td>2</td>
<td>[-6, -32, 2, 0, -51, 1, 0, 0] :</td>
</tr>
<tr>
<td>3</td>
<td>[56, 32, 2, 22, 22] :</td>
</tr>
<tr>
<td>4</td>
<td>[-56, -3, 0, -21, 0, 6, 5] :</td>
</tr>
<tr>
<td>5</td>
<td>[56, 3, 2, 0, 251, 1, 41, 22] :</td>
</tr>
<tr>
<td>6</td>
<td>[-56, -3, 2, 0, -251, 1, -41, 0] :</td>
</tr>
<tr>
<td>7</td>
<td>[] :</td>
</tr>
</tbody>
</table>
A3 Q2 - get_memory_score()

**parameter** – a list of random numbers (0 – 10)

**returns** – the score.

```python
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, 3, 1, 1, 8, 2]))
print("3. Score:", get_memory_score([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))
```

The list of random numbers:

[3, 4, 3, 0, 7, 4, 5, 2, 1, 3]
A3 Q3 - get_most_recent()

parameters – a list of numbers and a list of numbers to test

returns – the number from the second list which occurs most recently (closest to the end of the list) in the first parameter list. (-1)

```python
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 = [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, [1, 0, 3, 4]))
```


1. 3
2. 0
3. 9
4. -1
5. 8
6. -1
7. 3
A3 Q4 - is_a_valid_code()

**parameter** – a string

**returns** – a boolean indicating whether the parameter string denotes a valid code or not.

The first three lines of the function are:

```python
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
```

```python
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 '))
```

1. True
2. True
3. False
4. False
5. False
6. True
A3 Q5 – get_longest_e_word()

**parameter** – a list of strings

**returns** – the longest word in the parameter list which has at least 6 letters. (If two or more are the longest then last word in the list).

```python
print("1.", get_longest_e_word(["Melissa", "Jessie", "Kath", "Amity", "Raeann"]))
print("2.", get_longest_e_word(["Jo", "Jessie", "Penelope", "Jin", "Raeann", "Pamelita"]))
print("3.", get_longest_e_word(["Alan", "Melita", "Amity", "Rosalia", "Rositta", "LeeAnne"]))
print("4.", "***", get_longest_e_word(["Jo", "Jai", "Jen", "Jing", "Joey", "Jess"]), "***", sep = ")
print("5.", "***", get_longest_e_word([]), "***", sep = ")
print("6.", "***" + get_longest_e_word([""]) + "***")
```

1. Melissa
2. Pamelita
3. LeeAnne
4. *****
5. *****
6. *****
A3 Q6 - remove_triplets()

parameters – a list of integers

returns – None

The function removes triplets made up of three sequential identical elements

```python
a_list = [6, 6, 6, 7, 6, 6, 6, 3, 3, 3, 8, 8, 8, 3]
remove_triples(a_list)
print("1.", a_list)

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

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

a_list = [1, 1, 1, 4, 4, 4, 1, 1, 1]
remove_triples(a_list)
print("3.", a_list)
```

1. [7, 3]
2. [7, 6, 6]
3. [7, 6, 6, 4, 3]
4. []
**A3 Q7 - get_dice_score()**

**parameter** – a list of dice throws

**returns** – the value of the hand according to the rules:

- 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 of the hand.
- If there is no 1 in a hand of dice then the whole hand scores 0.
- A hand of dice can contain more than one run.

<table>
<thead>
<tr>
<th>List of Dice Throws</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>[5, 3, 2, 5, 4, 5, 6, 4, 3]</td>
<td>0</td>
</tr>
<tr>
<td>[3, 4, 1, 5, 3, 1, 4, 6]</td>
<td>2</td>
</tr>
<tr>
<td>[5, 3, 2, 6, 4, 5, 1, 4]</td>
<td>21</td>
</tr>
<tr>
<td>[2, 1, 1, 2, 3, 3, 1, 3, 2]</td>
<td>19</td>
</tr>
<tr>
<td>[3, 4, 1, 5, 2, 1, 5, 1, 2, 3, 4, 6]</td>
<td>37</td>
</tr>
</tbody>
</table>