Lecture 18 – Lists recap, Assignment 3
At the end of this lecture, students will be able to:

- understand the Assignment 3 requirements
Lists Recap

Lists
- Elements are separated by commas and enclosed in square brackets,
- Ordered sequence of items of any types

Create Lists
- an empty list: \texttt{list1 = []} or \texttt{list1 = list()}
- a list of ints: \texttt{list2 = [2, 3, 4]}
- a list of strings: \texttt{list3 = ["red", "blue"]}
- an integer list using the range function: \texttt{list4 = list(range(3, 6))}
- a list can include mixed types: [4, True, "Test", 34.8] [1, 2, "jim"]
Lists Recap - len() function

The elements of a list are the individual items in a list.
The len() function can be used to get the length of a list.

```
my_list = [10, 20, 30, 40, 50]
print(len(my_list))
```

5

Specific elements of a list can be accessed using an integer index which indicates the position of the element in the list (starting from position 0).
Lists Recap - accessing list elements

Specific elements in a list can be manipulated using square bracket notation with the index number of the element to be accessed.

```python
my_list = [10, 20, 30, 40, 50]
print(my_list[2])
my_list[3] = my_list[1] + my_list[len(my_list) - 1]
print(my_list[0], my_list[3])
```

This way of writing 'len(my_list) - ...' to access elements from the end of the lists can be avoided.
Lists Recap - accessing list elements

The elements of a list can be accessed from the end of the list by using a negative index value.

```python
my_list = [10, 20, 30, 40, 50]
print(my_list[-4])
my_list[-3] = my_list[-1] + my_list[-2]
print(my_list[-3], my_list[1], my_list[-5])
```

```
20
90 20, 10
```
Index out of Range - IndexError

Warning! If you try to access an element that does not exist, Python will throw an error!

```
my_list = [10, 20, 30, 40, 50]
print(my_list[5])       #NO! Element at index 5 does not exist
print(my_list[-6])      #NO! Element at index -6 does not exist
```

IndexError: list index out of range
Lists Recap - the 'in' Operator (membership)

The `in` operator returns a boolean. It returns True if the value (on the left hand side of the `in` operator) is an element of the list. Otherwise the `in` operator returns False.

```
my_list = [10, 20, 30, 40, 50]
result1 = 100 in my_list
print(result1)
print(30 in my_list)
print(40 not in my_list)
```

```
False
True
False
```
Lists Recap - visiting each element in a list (iteration)

We can iterate through all the elements of a list, in order, using a for ... in loop, e.g.,

```python
my_list = [30, 20, 10, 20, 40, 30]
count = 0
for element in my_list:
    if element > count:
        count = count + 10
print(count)
```

```
30
```

```python
my_list = [10, 20, 30, 40, 50]
total = 0
for element in my_list:
    if element % 4 == 0:
        total = total + element
print(total)
```

```
60
```
Updating the elements of a list

The values in the elements of a list can be visited and updated using a for … in range(...) loop, e.g.,

```python
my_list = [10, 20, 30, 40, 50]
for index in range(len(my_list)):
    if index % 2 == 0:
        my_list[index] = my_list[index] + 5
    else:
        my_list[index] = my_list[index] + 10
print(my_list)

[10, 21, 30, 41, 50]

my_list = [10, 20, 30, 40, 50]
for index in range(len(my_list)):
    if my_list[index] % 4 == 0:
        my_list[index] = my_list[index] + 1
print(my_list)

[10, 21, 30, 41, 50]
```
List methods

position = a_list.index(...)  
element = a_list.pop()  
element = a_list.pop(...)  
a_list.insert(..., ...)  
a_list.append(...)  
a_list.reverse()  
a_list.sort()

my_list = []
my_list.append(4)
my_list.append(3)
my_list.append(21)
my_list.insert(2, 1)
my_list.insert(0, 2)
value = my_list.pop(1)
my_list.append(value + 3)
my_list.sort()
my_list.reverse()
my_list.pop()
if 7 in my_list:
    pos = my_list.index(7)
    my_list.append(pos)
else:
    pos = my_list.pop()

print(my_list, "pos:", pos)
Complete the `remove_multiples()` function from Lecture 16, Slide 18.

```python
def remove_multiples(number_list, multiples_of):
    # Your implementation here

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

main()
```

Output:
```
[25, 5, 9, 10, 15, 8]
Numbers left [9, 8]
```
CodeRunner Assignments

CompSci 101 has 5 assignments, in total worth 11% of your final mark. For two of these five assignments (a total of 4%), you are required to use the CodeRunner tool.

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

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

Information about using CodeRunner 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
CompSci 101 Assignment 3

Due: 4:30pm, May 8

Worth: 2% of your final mark

Topic: lists

This assignment is marked out of 20
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. 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 can submit one function at a time.
A3 Q1 - get_funny_average ()

**parameter** - a list of numbers

**returns** – the average (to one decimal place) of **positive** non zero elements excluding the minimum and the maximum positive elements.

```
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([]))
```

<table>
<thead>
<tr>
<th></th>
<th>Funny average</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.5</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>25.3</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>24.8</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</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, 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))
```

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]
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 in the first list. Most recent is the last element of the list.

```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_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]))
```
A3 Q4 - is_legitimate_code()

**parameter** – a string

**returns** – a boolean indicating whether the parameter string denotes a legitimate 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_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 '))
```

1. True
2. True
3. False
4. False
5. False
6. True
A3 Q5 – get_longest_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 on the right.)

```
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([""]) + "***")
```

1. Melissa  
2. Pamelita  
3. Rosetta  
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

```
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)
```
A3 Q7 - get_hand_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 hand score.
- 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.

- \([5, 3, 2, 5, 4, 5, 6, 4, 3]\) 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])
In a Python program:

• a `for ... in` loop can be used to access each individual element of a list
• a `for index in range()` loop can be used to make changes to individual element of a list
Examples of Python features used in this lecture

def change_list(a_list):
    number_of_elements = len(a_list)
    for i in range(number_of_elements):
        a_list[i] = a_list[i] * 2

def use_lists(list1, list2):
    list3 = []
    for index in range(len(list1)):
        list3 = list3 + [list1[index] + list2[index]]
    return list3