Lecture 15 – Lists are mutable objects, making changes to the elements of lists, the split() method
At the end of this lecture, students should be able to:

- use the index number to access individual elements of a list
- make changes to the elements of a list
- copy the values of a list
- use the split() method on a string to obtain a list of string objects
- lists are mutable objects
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
    
    ```python
    list1 = []
    or
    list1 = list()
    ```
  
  - a list of ints
    
    ```python
    list2 = [2, 3, 4]
    ```
  
  - a list of strings
    
    ```python
    list3 = ["red", "blue"]
    ```
  
  - an integer list using the range function
    
    ```python
    list4 = list(range(3, 6))
    ```
  
  - a list can include mixed types
    
    ```python
    [4, True, "Test", 34.8]
    ```
    
    ```python
    [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.

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

- 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!

```python
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

![Index Error Diagram]
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.

```python
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
```
Mutable Objects, Immutable Objects

```python
def main():
    list1 = [6, 4, 7]
    value1 = list1[0]
    value1 = value1 + 3
    print("1.", list1, value1)

    list2 = ["a", "b", "c"]
    value2 = list2[0]
    value2 = value2.upper()
    print("2.", list2, value2)

    list3 = ["a", "b", "c"]
    value3 = list3
    value3[0] = value3[0].upper()
    print("3.", list3, value3)
    main()
```

- String and int objects are **immutable** (look at outputs 1. and 2.).

- List objects are **mutable** (look at output 3.).

1. [6, 4, 7] 9
2. ['a', 'b', 'c'] A
3. ['A', 'b', 'c'] ['A', 'b', 'c']

Why does the following not work as intended?

- In the following for...in loop, each element of the list is accessed but ...

```
def main():
a_list = [10, 8, 6, 4, 7]
print("1.", a_list)
    for number in a_list:
        number = number * 2
        print(number, end=" ")
print()
print("3.", a_list)
main()
```

1. [10, 8, 6, 4, 7]
20 16 12 8 14
3. [10, 8, 6, 4, 7]

Note that in the above example, the values of the elements in the list have not changed in any way.

What if the intention was to change the element values in the list?
Changing the values of elements in the list

- The elements in a list can be changed if we assign to each element of the list using the `index` of the element, e.g.,

```python
def main():
    a_list = [10, 8, 6, 4, 7]
    print("1.", a_list)
    number_of_elements = len(a_list)

    for index in range(number_of_elements):
        a_list[index] = a_list[index] * 2

    print("2.", a_list)
main()
```

1. `[10, 8, 6, 4, 7]`
2. `[20, 16, 12, 8, 14]`

Changing a value at an index location updates the element of the list.
def main():
    my_list = [10, 8, 6, 4, 7]
    for index in range(len(my_list)):
        print(index, my_list[index])
main()
Complete the main() function

- Complete the code in the main() function which adds 1 to each list element which has odd value.

```python
import random
def main():
    a_list = []
    for index in range(10):
        a_list = a_list + [random.randrange(1, 100)]

    print("1.", a_list)
    print("2.", a_list)

main()
```

1. [69, 98, 7, 92, 13, 9, 27, 36, 96, 46]
2. [70, 98, 8, 92, 14, 10, 28, 36, 96, 46]
Complete the main() function

- Complete the code in the main() function which changes the elements **starting from index 1** so that each element is the accumulative total of the previous elements (i.e., element 1 is the sum of the element 0 and element 1, element 2 is the sum of element 1 and element 2, etc.).

```python
import random

def main():
    a_list = []
    for num in range(10):
        a_list = a_list + [random.randrange(1, 10)]
    print("1.", a_list)
    print("2.", a_list)

main()
```

1. [8, 1, 9, 5, 6, 3, 6, 4, 5, 6]
2. [8, 9, 18, 23, 29, 32, 38, 42, 47, 53]
The split() method

- The split() method separates a **single string** into a **list of the parts** (the tokens) **of the string** using the separator defined (within the parentheses). Each element of the list is a string object. This method can be applied to any string object.

- If no separator is defined (as in the code below), whitespace is the default separator, e.g.,

```python
def main():
    phrase = 'The best cure for insomnia is to get a lot of sleep'
    list_of_words = phrase.split()
    print(list_of_words[0], list_of_words[4], list_of_words[7])

main()
```

The insomnia get
The split() method - example

```python
1 def main():
2     prompt = "Enter a line of numbers: "
3     line_of_nums = input(prompt)
4     list_of_nums = line_of_nums.split()
5     for index in range(len(list_of_nums)):
6         list_of_nums[index] = int(list_of_nums[index])
7     total = 0
8     for number in list_of_nums:
9         total = total + number
10    print("Total: ", total)
11 main()
```

Enter a line of numbers: 5 -3 6 8 1
Total: 17

Enter a line of numbers: 4 6 12 13 9
Total: 44

Note that split() function breaks a string up into a list of strings.
def split_message(message):
    words = message.split()
    num = int(words[1])
    num = num + 4
    words[2] = num
    a_word = words[0]
    words[0] = a_word[0:3]
    print(words[1], words[0], words[2], sep = "")

def main():
    phrase = 'tuna 4 lunch'
    split_message(phrase)

main()
Assigning a list object to a variable

- Python lists are objects. When an object is assigned to a variable, the reference (the address) is copied and stored in the variable.

```
list1 = [1, 2, 3]
list2 = list1
list3 = [1, 2, 3]
print(list1)
print(list2)
print(list3)
print()
print(list1)
print(list2)
print(list3)
```

```
[1, 2, 3]
[1, 2, 3]
[1, 2, 3]
[1, 2, 5]
[1, 2, 5]
[1, 2, 4]
```
Do the following two sections of code give the same output? If not, what is the difference in output?

**Code A**

```python
list1 = [1, 2, 3]
list2 = list1

for index in range(len(list1)):
    list2[index] = list1[index] * 2

print("1.", list1)
print("2.", list2)
```

**Code B**

```python
list1 = [1, 2, 3]
list2 = [1, 2, 3]

for index in range(len(list1)):
    list2[index] = list1[index] * 2

print("1.", list1)
print("2.", list2)
```
**Summary**

- **In a Python program:**
  - A `for ... in` loop can be used to access each individual element of a list.
  - A `for ... in range()` loop can be used to make changes to individual element of a list.
  - A list is an object. Assigning a list to a variable makes a copy of the reference (not a copy of the list).
  - Lists are mutable objects.
  - We use the `split()` method to break a string into a list of strings. The default separator for the `split()` method is whitespace.

---

21

CompSci 101 - Principles of Programming
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

def split_message(message):
    words = message.split()
    print(words[2], words[0])