CMP4266: Computer Programming
Lab Session 5: Lists, Tuples, Dictionaries

Objectives

a. Experiment with the range function to generate lists.
b. Experiment with list operations, creating, inserting, appending and slicing lists.
c. Process every item in a list using a for loop.
d. Carry out programming exercises involving functions, lists and dictionaries.

1. Tuples, Collection Types, Lists, Dictionaries

1.1 Tuples, type evaluation and comparison

We've used tuples as an ordered collection of items which can't be changed e.g. (10.5,"hello"). These are usually surrounded by round () brackets, and items are separated using commas (,). We can also have empty tuples, or tuples with just one item, but in that case if we put a comma after the item we'll avoid confusing the interpreter. If we want to know, we can find out how an object's type will be evaluated, by passing any object as a parameter to the type() function:

```
print(type(("a",10.5)))
```
Output
```
<class 'tuple'>
```

```
print (type((0)))
```
Output
```
<class 'int'>
```

```
print(type((0,)))
```
Output
```
<class 'tuple'>
```

```
print(type(()))
```
Output
```
<class 'tuple'>
```

```
print(type("hello"))
```
Output
```
<class 'str'>
```

```
print(type(10.5))
```
Output
```
<class 'float'>
```

In Python it sometimes makes sense to compare types for equality:

```
print(type(math.pi) == type(10.5))
```
Output
```
True
```

```
print(type(10.5) == type("hello"))
```
Output
```
False
```

So applying what we learned last week, we can then branch code within a function, based upon the type of a function parameter.

1.2 Lists

A list is an ordered collection of objects which we can change, surrounded by square brackets.

```
print(type([]))
```
Output
```
<class 'list'>
```

```
print(type([2,4]))
```
Output
```
<class 'list'>
```

```
print(type(['hello', 10.5]))
```
Output
```
<class 'list'>
```

A list can be empty too. Sometimes we can construct a list using a loop, starting with the empty list: []. Mostly we'll want lists where all the items are of the same type, but they don't have to be.

1.2.1 Manipulating a list

```
l=[2,3]
l.append(5)
print(l)
```
Output
```
[2, 3, 5]
```

Append method inserts an item at the end of a list.

```
l.insert(0,1)
print(l)
```
Output
```
[1, 2, 3, 5]
```

```
l.insert(2,4)
print(l)
```
Output
```
[1, 2, 4, 3, 5]
```

The insert method inserts its 2nd parameter at the index position given by the first parameter. Index positions start counting at 0, so a list with 5 items will have indices in the range 0 to 4 inclusive.

```
print(l[1])
```
Output
```
2
```

```
print(l[4])
```
Output
```
5
```

1.2.2 Measuring length, sorting and reversing a list

We can get the length of any sequence object (e.g. list or tuple) using the len() function:

```
print(len(l))
```
Output
```
5
```
```
print(len("this"))
```
Output
```
4
```

```
print(len((0,)))
```
Output
```
1
```

Lists can be reversed and sorted. These operations don't return the changed list, but they do change the order of the items within it.

```
l.reverse()
print(l)
```
Output
```
[5, 3, 4, 2, 1]
```

```
l.sort()
print(l)
```
Output
```
[1, 2, 3, 4, 5]
```

The del command removes the indexed item, and shifts higher indices down by one.

```
del l[1]
print(l)
```
Output
```
[1, 3, 4, 5]
```

```
del l[3]
print(l)
```
Output
```
[1, 3, 4]
```

1.2.3 Generating a list using the list() and range() function

```
print(list(range(1,10)))
```
Output
```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

The first parameter to range() is the inclusive start of the range. The second parameter is the exclusive end of the range. We can subtract the start parameter from the end parameter to get the length.

We can go up in 2's or 3's, or any gap or increment using a 3rd parameter.

```
print(list(range(1,20,2)))
```
Output
```
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
```

```
print(list(range(3,20,3)))
```
Output
```
[3, 6, 9, 12, 15, 18]
```

Sometimes we want to count down so we use a negative increment.

```
print(list( range(20,0,-1)))
```
Output
```
[20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

1.2.4 Slicing a list

```
a=list(range(1,11))
print(a)
```
Output
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
print(a[3:5])
```
Output
```
[4, 5]
```

```
print(a[:])
```
Output
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
print(a[:6])
```
Output
```
[1, 2, 3, 4, 5, 6]
```

```
print(a[6:])
```
Output
```
[7, 8, 9, 10]
```

If you do these experiments, you'll understand the rest of this explanation better. We can obtain a slice from the list, or a part of it, by using the slice operator. This is a pair of square brackets, with a colon, with indices before and after the colon, the first being inclusive and the second exclusive. If a start or end index isn't present, the start or end of the slice will be at the start or end of the list. A slice of part or of the whole list will take a copy of the list, and make this into a new list. Change the old list, and the cloned copy won't be changed. However, if you assign a list to another reference variable, these 2 references will still point to the same list.

```
b=a
c=a[:]
a.remove(3)
print(a)
```
Output
```
[1, 2, 4, 5, 6, 7, 8, 9, 10]
```

```
print(b)
```
Output
```
[1, 2, 4, 5, 6, 7, 8, 9, 10]
```

```
print(c)
```
Output

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
Enter a number or * to quit: 2
Enter a number or * to quit: 5
Enter a number or * to quit: 7
Enter a number or * to quit: *
14.0
```

1.3 Dictionaries

1.3.1 Dictionaries are declared as a list of comma separated key/value pairs between curly braces. Key and value are separated by a colon. An empty dictionary is created with just a pair of curly braces. You can use len(· · ·) to get the number of key/value pairs in a dictionary.

```
len({})
len({ "key1" : "value1", "key2" : "value2" })
```

1.3.2 Keys are associated with values in the dictionary by indexing the dictionary with a key and assigning a value:

```
d = {}
d["key1"] = "value1"
d["key2"] = "value2"
```

1.3.3 We retrieve the value associated with a specific key by indexing the dictionary with the same key:

```
d["key1"]
```

1.3.4 The inclusion operator tests whether keys exist or not:

```
d = { "key1" : "value1", "key2" : "value2" }
"key1" in d
"key3" in d
```

1.3.5. Key/value pairs can be removed from a dictionary by using the del keyword

```
d = { "key1" : "value1", "key2" : "value2" }
del d["key1"]
```

1.3.6. When we iterate through a dictionary using a for loop, we actually iterate over the keys:

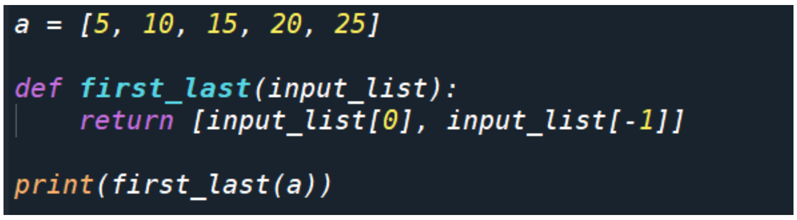
```
d = { "key1":1, "key2":2, "key3":1, "key4":3, "key5":1, "key6":4, "key7":2 }
for k in d :
        print("key=", k, " value=", d[k], sep="")
```

Output

```
key=key1 value=1
key=key2 value=2
key=key3 value=1
key=key4 value=3
key=key5 value=1
key=key6 value=4
 key=key7 value=2
```

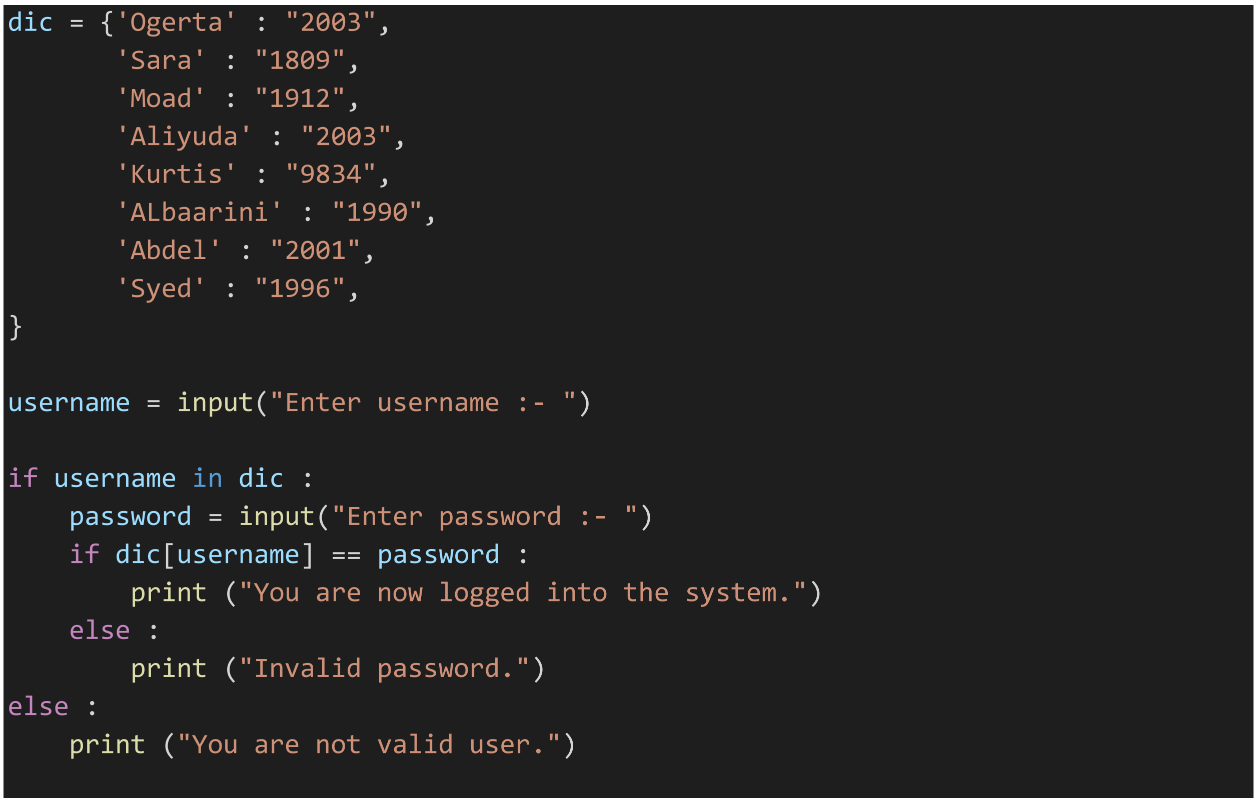
2. Getting started

2.1 Show the output from the following code:



```python
a = [5, 10, 15, 20, 25]

def first_last(input_list):
    return [input_list[0], input_list[-1]]

print(first_last(a))
```

2.2 Explain what the following program does.



```python
dic = {'Ogerta' : "2003",
       'Sara' : "1809",
       'Moad' : "1912",
       'Aliyuda' : "2003",
       'Kurtis' : "9834",
       'ALbaarini' : "1990",
       'Abdel' : "2001",
       'Syed' : "1996",
}

username = input("Enter username :- ")

if username in dic :
    password = input("Enter password :- ")
    if dic[username] == password :
        print ("You are now logged into the system.")
    else :
        print ("Invalid password.")
else :
    print ("You are not valid user.")
```

3. Lab 5 Submission Exercises

Exercise 3.1

- Create a script named lab5_ex1.py.
- Before attempting this task, make sure you have tried all the codes from the lecture notes if you are new to Python programming.
- In the script, create the following functions:

Function	Parameters	What does it do?
`create_list`	N/A	It returns a list with the following elements: **['PlayStation', 'Xbox', 'Steam', 'iOS', 'Google Play']**
`get_info`	`my_list` - list type	It returns the following information of my_list as a tuple: **the first element, the second last element, number of elements.**
`get_partial`	`my_list` - list type	It returns a new list which contains **the 2nd, 3rd, and 4th elements from my_list, in the original order**.
`get_last_three`	`my_list` - list type	It returns a new list which contains **the last three elements from my_list, in the reversed order**.
`double_list`	`my_list` - list type	It returns a new list which **concatenates two of my_list.**
`amend`	`my_list` - list type	It returns a new list which the following amendments: **change the 2nd element of my_ list to "None", add "Bye" to the end of my_list.**

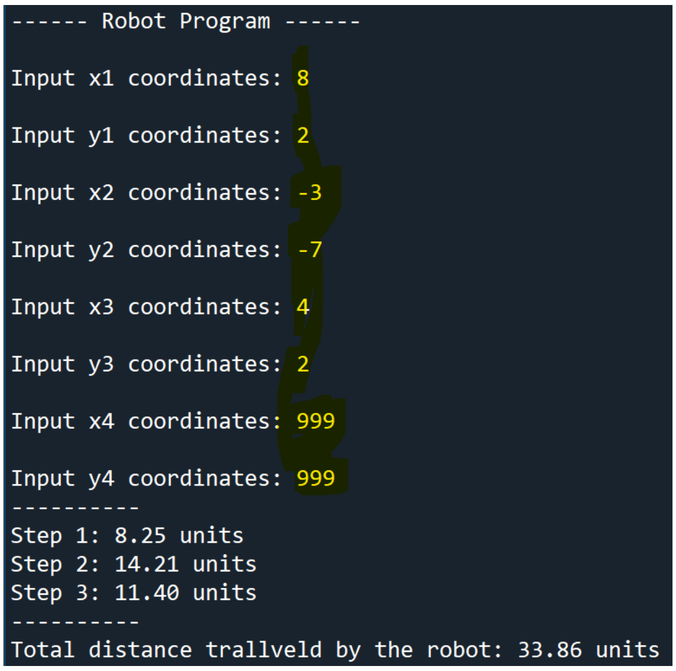
- You can check your solution with the follow:

```
if __name__ == "__main__":
    test_list = create_list()
    print(test_list)
    print(get_info(test_list))
    print(get_partial(test_list))
    print(get_last_three(test_list))
    print(double(test_list))
    print(amend(test_list))
```

```
['PlayStation', 'Xbox', 'Steam', 'iOS',
'Google Play']
('PlayStation', 'iOS', 5)
['Xbox', 'Steam', 'iOS']
['Google Play', 'iOS', 'Steam']
['PlayStation', 'Xbox', 'Steam', 'iOS',
'Google Play', 'PlayStation', 'Xbox',
'Steam', 'iOS', 'Google Play']
['PlayStation', 'None', 'Steam', 'iOS',
'Google Play', 'Bye']
```

Exercise 3.2

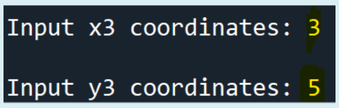
You are going to design a program which allows a robot to move in a 2D coordinate system starting from the original point (0, 0). The robot can move to the next point (x, y) specified by the user. If the next point is (999, 999), the program ends and prints out the distance moved at each step.
Your program can be implemented in various ways as long as it has the below behavior/display format - user inputs are highlighted in yellow:



```
------ Robot Program ------

Input x1 coordinates: 8

Input y1 coordinates: 2

Input x2 coordinates: -3

Input y2 coordinates: -7

Input x3 coordinates: 4

Input y3 coordinates: 2

Input x4 coordinates: 999

Input y4 coordinates: 999
----------
Step 1: 8.25 units
Step 2: 14.21 units
Step 3: 11.40 units
----------
Total distance trallveld by the robot: 33.86 units
```

However, your program must contain at least the following functions:

Function	Parameters	What does it do?

`get_next_point`	`step`
	- int type
		It returns a tuple (x, y) from the user input. The parameter step indicates the step number.
		For example:
		get_next_point(3) asks input for x3, y3 (user input in yellow)
		
		`Input x3 coordinates: 3`
		`Input y3 coordinates: 5`
		It returns (3, 5) - note the return is a tuple and has no step info.

`cal_distance`	`p1, p2` - tuple type	**It returns the distance (float) between the two points.** To calculate the distance between two points (x1, y2) and (x2, y2), use the formula below: $distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ **How to calculate the square root? Refer to Lab 1 - task 10.**
`main`	N/A	It runs the whole robot program. It uses **get_next_point()** to take user inputs. It uses **cal_distance()** to calculate the distance between two points. It could use any other functions created by you as well.

Further notes on output format:

- "------ Robot Program ------" has 6 "-" at each end.
- " ----------" before and after the steps has 10 "-".
- The units should be printed with 2 decimal places.

Use the following code to test your robot program:

```
if __name__ == "__main__":
    main()
```

4. Moodle submission

At the end of this lab session, zip the python files that you created in Sections 3 (3.1 and 3.2) and then upload the zip file in the Moodle. **Note:** It is important to complete the weekly labs in particular labs 1, 2, 3, 4, 5 and 6 because it contains questions that are part of the coursework. Failure to complete and submit the answers to these labs on the Moodle before the deadline may mean giving you **zero** on for this assessment component.

The zip file should be named using the following format StudetName_studentID.zip For example: OgertaElezaj_123456789.zip

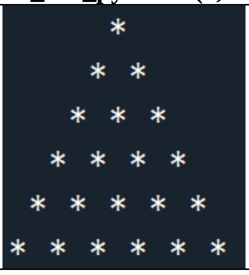
5. Additional Exercises

5.1 Create a function named **print_full_pyramid(level, symbol)**, which takes two parameters: level and symbol.

- level: an integer and it's greater than 0.
- symbol: a string to form the pyramid.

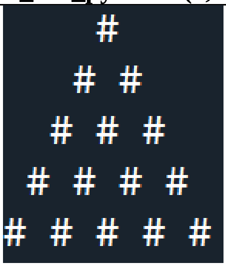
See blow for the print format of the pyramid:

print_full_pyramid(6,"*")



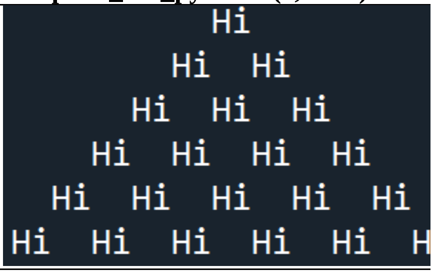
```
     *
    * *
   * * *
  * * * *
 * * * * *
* * * * * *
```

print_full_pyramid(6,"#")



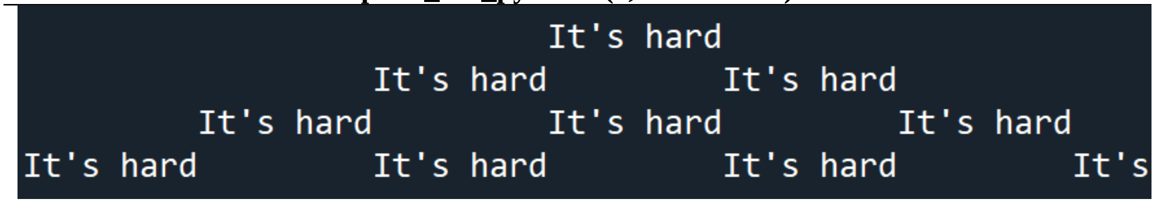
```
    #
   # #
  # # #
 # # # #
# # # # #
```

print_full_pyramid(9,"Hi")



```
                Hi
            Hi  Hi
        Hi  Hi  Hi
    Hi  Hi  Hi  Hi
Hi  Hi  Hi  Hi  Hi
Hi  Hi  Hi  Hi  Hi  H
```

print_full_pyramid(4, "It's hard")



```
                                                   It's hard
                                 It's hard                  It's hard
                It's hard                 It's hard                  It's hard
It's hard                It's hard                 It's hard                  It's
```

5.2 Write a program that get two lists as input and check if they have at least one common member.

5.3 Write a function named `find_greater(x_list, x_num)` that expects a list of numbers and a number. It returns a new list consisting of all the numbers in the list that are greater than the number x_num. For example:

```
        >>> find_greater( [11, 35, 46, 2, 104, 43, 41,8], 10 )
        [11, 35, 46, 104, 43, 41]
```

5.4 Write a function named `sum(list_x, list_y)` that expects two lists of numbers and returns a new list containing the sums of those lists' members. For example:

```
>>> sum( [2,4,7], [1,2,3] )
        [3, 6, 10]
```

5.5. Write a function called word frequencies(list_a) that accepts a list of strings called list_a and returns a dictionary where the keys are the words from list_a and the values are the number of times that word appears in list_a.

5.6 Modify exercise 3.2 to complete this exercise. Using the current

location (x1,y1) and the destination location (x2,y2) the software should also be able to identify direction of the movement. For example, if the robot is moving to the Bottom, Top, Right, Left or combination of them e.g. Top Right or Bottom Left.
As in exercise 3.2, when the user inputs a special number/character, the software should print all previous movements by showing a combination of the moved distance and the direction of each step. The trace function also displays the total distance travelled by the robot.
The trace of robot movement is shown as the following:

Output

```
please insert destination X value: 4
please insert destination Y value: 3
please insert destination X value: 7
please insert destination Y value: 4
please insert destination X value: 2
please insert destination Y value: 2
please insert destination X value: 90
please insert destination Y value: 43
please insert destination X value: 999 # special number to exit the loop
please insert destination Y value: 999 # special number to exit the loop
Step 1: 5 meter to Top Right
Step 2: 3 meter to Top Right
Step 3: 5 meter to Bottom Left
Step 4: 97 meter to Top Right
----------------------------------------
Total distance (in meters) the robot has moved is : 110
```