**Department of Electrical Engineering and   
Computer Science**

**Faculty Member:** Dr. Hafsa Iqbal **Date:** 12/02/2024

**Semester:** 8th **Group:** GP-1

**Lab Engineer:** Munadi Ahmad Sial

**EE-381 Robotics**

Lab 2: Data Structures and Modules in Python

**Group Members**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | **PLO5 – CLO4** | | **PLO9 –**  **CLO4** |  |
| **Name** | **Reg. No** | **Analysis of Data in Lab Report** | **Modern Tool Usage** | **Individual and Team Work** | **Total Marks** |
|  |  | **10 Marks** | **5 Marks** | **5 Marks** | **20 Marks** |
| Muhammad Abdullah Sohail | 343642 |  |  |  |  |
| Muhammad Ahmed Mohsin | 333060 |  |  |  |  |
| Muhammad Umer | 345834 |  |  |  |  |
| Hassan Rizwan | 335753 |  |  |  |  |

**Table of Contents**

[2 Data Structures and Modules in Python 3](#_Toc159268875)

[2.1 Introduction 3](#_Toc159268876)

[2.2 Objectives 3](#_Toc159268877)

[2.3 Theory 3](#_Toc159268878)

[3 Lab Tasks 4](#_Toc159268879)

[3.1 Task 1 – Lists 4](#_Toc159268880)

[3.2 Task 2 – Adding Items to List 5](#_Toc159268881)

[3.3 Task 3 – Checking Items in List 6](#_Toc159268882)

[3.4 Task 4 – Importing From Modules 6](#_Toc159268883)

[3.5 Task 5 – List Slicing 7](#_Toc159268884)

[3.6 Task 6 – Selection Sort with Lists 8](#_Toc159268885)

[3.7 Task 7 – 2D Lists 9](#_Toc159268886)

[3.8 Task 8 – Dictionaries 12](#_Toc159268887)

[4 Conclusion 13](#_Toc159268888)

# Data Structures and Modules in Python

## Introduction

This laboratory exercise is meant to introduce the data structures and importing of modules in python language which will be very important when learning the Robot Operating System (ROS) for programming robots.

## Objectives

The following are the main objectives of this lab:

* Implement data structures such as lists and dictionaries in python
* Create, alter and loop through lists
* Use slicing to access range of items in a list
* Utilize various list methods such as append, insert, extend, remove, pop etc.
* Implement 2-D lists
* Create and implement a dictionary

## Theory

Data structures are an important part of python. The 4 main data structures are lists, tuples, sets and dictionaries. Lists are most commonly used so they will be the major part of the lab tasks. Dictionaries are also used at times. Tuples and sets are very similar to lists and are not very commonly used in robotics. In this lab, mainly lists will be considered. To write python scripts (.py files), the SublimeText application will be used which is a text editor with syntax highlighting. Once the code is written, the script is saved and then executed by using the Linux terminal.

The terminal commands are given as:

**cd <directory>**  change directory

**cd..**  go back to previous directory

**pwd**  print the current directory

**ls** list the contents of the current directory

**python <script.py>** execute python script

# Lab Tasks

## Task 1 – Lists

Create a simple list containing the characters of your names. Loop through the list and display each character on a new line. Provide the code and all relevant screenshots.

### Task 1 Code Starts Here ###

name\_list = ["UMER", "ABDULLAH", "AHMED", "HASSAN"]

for name in name\_list:

    for char in name:

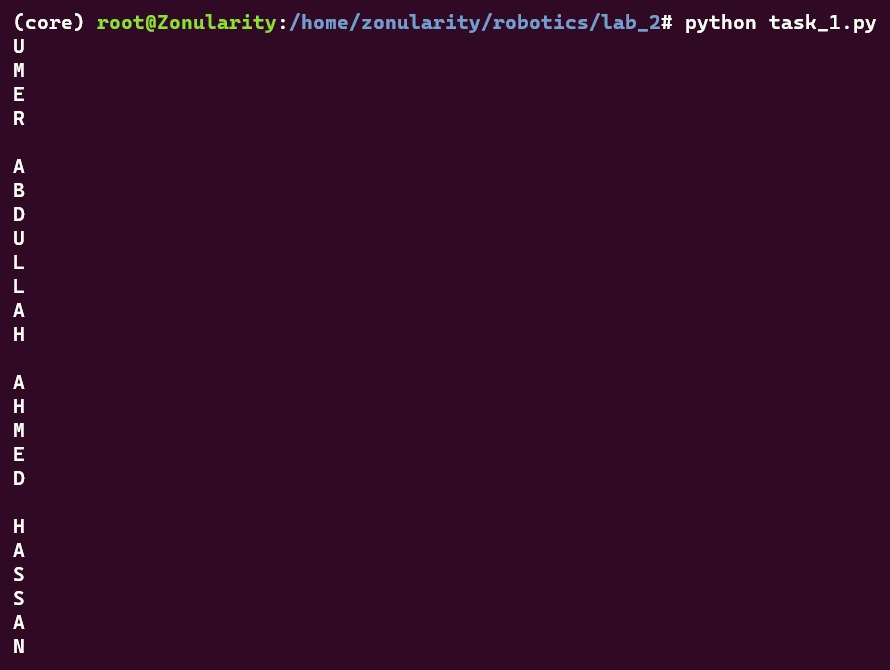
        print(char)

*# Separate each name with a new line*

    print()

### Task 1 Code Ends Here ###

### Task 1 Screenshot Starts Here ###



### Task 1 Screenshot Ends Here ###

## Task 2 – Adding Items to List

Write a program which first prompts the user for an integer which will be the size of a list. Then, the program must repeatedly prompt the user to input the items (integers or floats) of the list. Each time a number is added to the list, it must be placed in such a way that the list items are always in ascending order. Additionally, each time a number is input, the list is to be printed showing the newly added number. The inputs continue until the number of items reaches the size of the list. The final list is then printed. The syntax for making an empty list and appending function is given below:

my\_list = [ ]

my\_list.append(item)

### Task 2 Code Starts Here ###

number\_list = []

size = *int*(input("Enter the size of the list: "))

while len(number\_list) < size:

    number = *int*(input("\nEnter an input: "))

    for index, item in enumerate(number\_list):

        if number < item:

            number\_list.insert(index, number)

            break

    else:

        number\_list.append(number)

    print(number\_list)

### Task 2 Code Ends Here ###

### Task 2 Screenshot Starts Here ###



### Task 2 Screenshot Ends Here ###

## Task 3 – Checking Items in List

Write a function that takes in a list input and returns true if the items of the list make a palindrome. A palindrome is a word/number that is written the same way forward and backward. Examples of palindromes include “radar”, “level”, “5445”, “8395938”, “racecar”. To use a list in a function, use the following syntax:

def my\_function(my\_list):

statement1

statement2

### Task 3 Code Starts Here ###

word = input("Enter a word to check if it is a palindrome:\n")

palindrome\_list = *list*(word)

print(*f*"\n{palindrome\_list}")

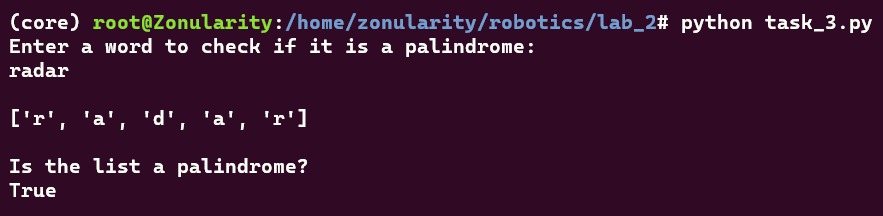
*def* is\_palindrome(*my\_list*):

    return my\_list == my\_list[::-1]

print(*f*"\nIs the list a palindrome?\n{is\_palindrome(palindrome\_list)}")

### Task 3 Code Ends Here ###

### Task 3 Screenshot Starts Here ###



### Task 3 Screenshot Ends Here ###

## Task 4 – Importing From Modules

Write a function that takes two arguments that will mark the start and end of range of integer values from which a random number is generated. Use your function to return random numbers which are then fed into a sine function. Provide the code and all relevant screenshots.

### Task 4 Code Starts Here ###

import math

import random

low = *int*(input("Enter the low end of the range: "))

high = *int*(input("Enter the high end of the range: "))

*def* random\_number(*start*, *end*):

    return round(random.uniform(start, end), 3)

random\_num = random\_number(low, high)

print(*f*"\nRandom number : rand = {random\_num}")

print(*f*"sin(rand)) = {round(math.sin(random\_num), 3)}")

### Task 4 Code Ends Here ###

### Task 4 Screenshot Starts Here ###

### Task 4 Screenshot Ends Here ###

## Task 5 – List Slicing

Create a list with the sequence 1, 2, 3… 20. Then using the slice operation (:) on this list, print the following sub-lists:

5, 6, 7… 20

1, 2, 3… 12

7, 8, 9 … 16

4, 5

11, 12, 13, 14

### Task 5 Code Starts Here ###

main\_list = *list*(range(1, 20 + 1))

print("Main list: ", main\_list)

print("\nSub-list 1: ", main\_list[4:])

print("Sub-list 2: ", main\_list[0:12])

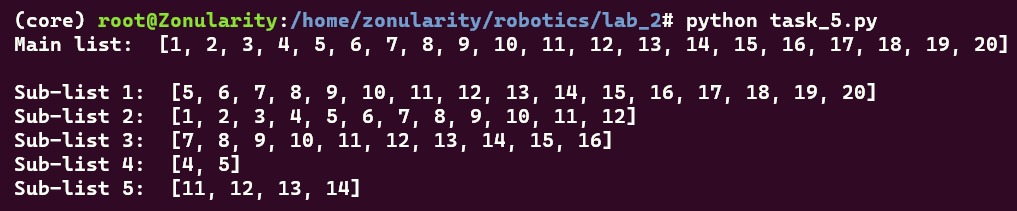
print("Sub-list 3: ", main\_list[6:16])

print("Sub-list 4: ", main\_list[3:5])

print("Sub-list 5: ", main\_list[10:14])

### Task 5 Code Ends Here ###

### Task 5 Screenshot Starts Here ###



### Task 5 Screenshot Ends Here ###

## Task 6 – Selection Sort with Lists

In this task, you will implement the Selection Sort algorithm using lists. A selection sort searches a list looking for the smallest element. Then, the smallest element is swapped with the first element of the list. The process is repeated for the sub-list beginning with the second element of the list. Each pass of the list results in one element being placed in its proper location. When the sub-list being processed contains one element, the list is sorted. Create a function which takes a list as input and then implements the selection sort on it. You need to print the list each time a swap is made.

### Task 6 Code Starts Here ###

*def* selection\_sort(*arr*):

    for i in range(len(arr)):

        min\_index = i

        for j in range(i + 1, len(arr)):

            if arr[min\_index] > arr[j]:  *# index of the smallest element*

                min\_index = j

        arr[i], arr[min\_index] = arr[min\_index], arr[i] *# swap*

*# print*

        if i == len(arr) - 1:

            print("Sorted list: ", arr)

        else:

            print(arr)

    return arr

in\_list = input("Enter a list of numbers separated by spaces: ").split()

in\_list = [*int*(x) for x in in\_list]

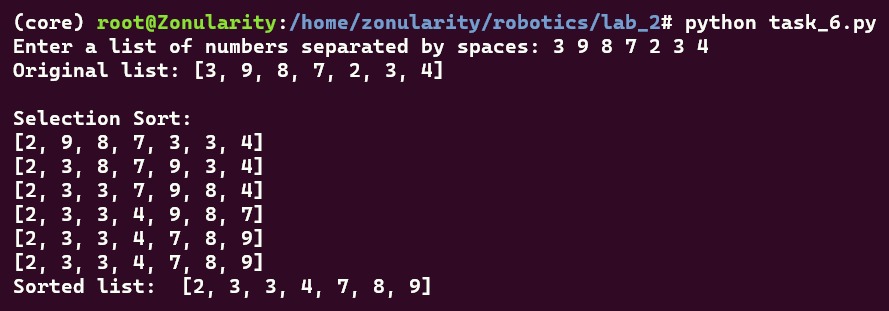
print(*f*"Original list: {in\_list}")

print("\nSelection Sort:")

selection\_sort(in\_list)

### Task 6 Code Ends Here ###

### Task 6 Screenshot Starts Here ###



### Task 6 Screenshot Ends Here ###

## Task 7 – 2D Lists

In this task, you will implement a 2-D list. You will create a simple 5x5 grid using a 2-D list. The empty space on the grid is indicated by 1’s. The “player” 0 will start at the center of the grid and will move up, down, left, and right. The following gives the start position:

[1 1 1 1 1]

[1 1 1 1 1]

[1 1 0 1 1]

[1 1 1 1 1]

[1 1 1 1 1]

This task is divided into 2 parts:

1. **Display Function**

The grid is defined as a 2-D list. If you print the 2-D list, it will show all the rows on the same line. To prevent this, define a function to display the 2-D list in such a way that each row is printed on a new line. The above grid example is properly displayed. The function must take a list as input. Manually, create the above grid as a 2-D list and call it in your display function.

### Task 7A Code Starts Here ###

*# With color*

*def* display(*grid*):

    for row in grid:

        for cell in row:

            if cell == 0:

                print("\033[92m{}\033[00m".format(cell), *end*=" ")

            else:

                print(cell, *end*=" ")

        print()

*# Without color*

*def* simple\_display(*grid*):

    for row in grid:

        print(row)

grid = [

    [1, 1, 1, 1, 1],

    [1, 1, 1, 1, 1],

    [1, 1, 0, 1, 1],

    [1, 1, 1, 1, 1],

    [1, 1, 1, 1, 1],

]

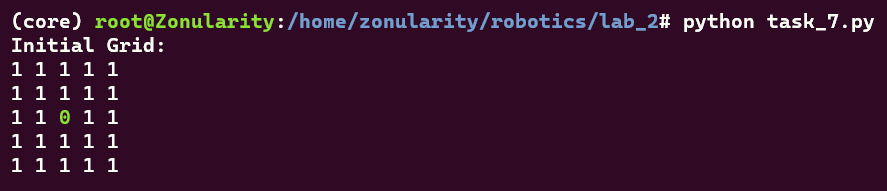
*# Display the grid*

print("Initial Grid:")

display(grid)

### Task 7A Code Ends Here ###

### Task 7A Screenshot Starts Here ###



### Task 7A Screenshot Ends Here ###

1. **Moving on the Grid**

Write a program that takes input from the user. The user must input “w”, “s”, “a”, or “d” for up, down, left or right movements respectively. When the input is given, the grid is updated so that the “player” 1 moves in the grid. At every input, the updated grid is displayed. You need to use the display function you created in the previous part for this. If the user input “q” then the program must end. If the user presses a button other than those mentioned, then it should be ignored.

*Note that you do not need to cater for the grid boundary in this task. Your program must only be to the point that you can move towards the boundary of the grid.*

### Task 7B Code Starts Here ###

*def* move\_player(*grid*, *move*):

    player\_row = 0

    player\_col = 0

*# Get the player's position*

    for i in range(len(grid)):

        for j in range(len(grid[i])):

            if grid[i][j] == 0:

                player\_row = i

                player\_col = j

                break

*# Move cases*

    if move == "w":

        if player\_row > 0:

            grid[player\_row][player\_col] = 1

            grid[player\_row - 1][player\_col] = 0

    elif move == "s":

        if player\_row < 4:

            grid[player\_row][player\_col] = 1

            grid[player\_row + 1][player\_col] = 0

    elif move == "a":

        if player\_col > 0:

            grid[player\_row][player\_col] = 1

            grid[player\_row][player\_col - 1] = 0

    elif move == "d":

        if player\_col < 4:

            grid[player\_row][player\_col] = 1

            grid[player\_row][player\_col + 1] = 0

    elif move == "q":

        return "q"

    return grid

grid = [

    [1, 1, 1, 1, 1],

    [1, 1, 1, 1, 1],

    [1, 1, 0, 1, 1],

    [1, 1, 1, 1, 1],

    [1, 1, 1, 1, 1],

]

*# Display the grid*

print("Initial Grid:")

display(grid)

*# Move the player*

while True:

    move = input("\nEnter a move (w, s, a, d, q): ")

    grid = move\_player(grid, move)

    if grid == "q":

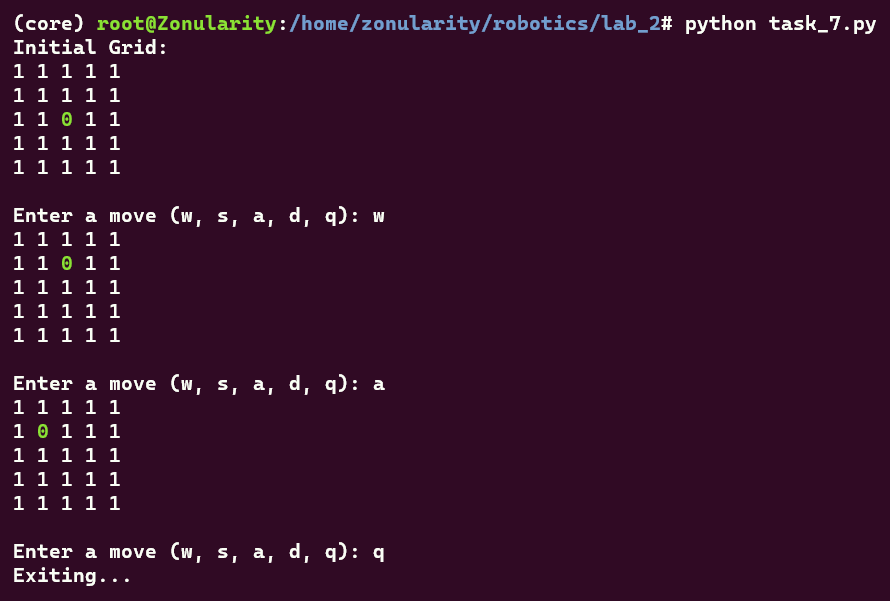
        print("Exiting...")

        break

    display(grid)

### Task 7B Code Ends Here ###

### Task 7B Screenshot Starts Here ###



### Task 7B Screenshot Ends Here ###

## Task 8 – Dictionaries

In this task, you will make use of dictionaries. Write a program that first prompts the user to input five strings which will be the keys of the dictionary. Then, the program must prompt the user to input the values of the respective keys. When entering the values, the user must be shown the key whose value is being input. Once all values are entered, display the dictionary.

### Task 8 Code Starts Here ###

task\_dict = {}

for i in range(5):

    key = input("Enter key " + *str*(i + 1) + ": ")

    task\_dict[key] = None

print()

for key in task\_dict:

    value = input("Enter value for " + key + ": ")

    task\_dict[key] = value

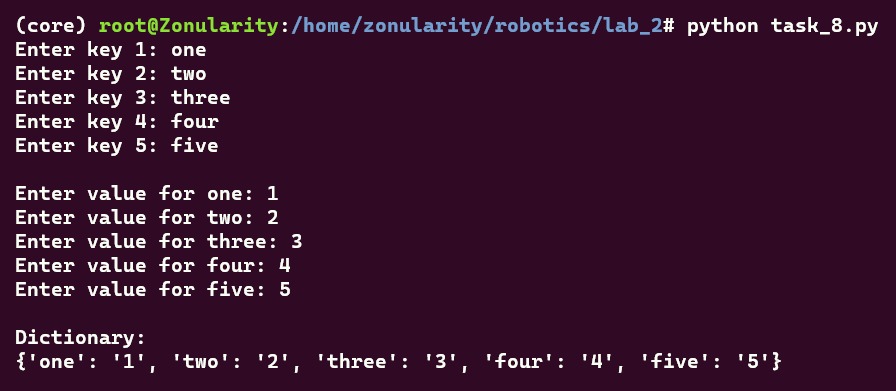
print("\nDictionary:")

print(task\_dict)

print()

### Task 8 Code Ends Here ###

### Task 8 Screenshot Starts Here ###



### Task 8 Screenshot Ends Here ###

# Conclusion

In this lab, we delved into the fundamentals of data structures and the intricacies of importing modules in Python, laying a solid foundation crucial for mastering the Robot Operating System (ROS) and its application in programming robots. Through hands-on experimentation and analysis, we gained invaluable insights into the efficiency and functionality of various data structures, honing our skills in harnessing Python's versatility for future robotic endeavors. By grasping these core concepts, we are better equipped to navigate the complexities of ROS and contribute meaningfully to the advancement of robotics technology.