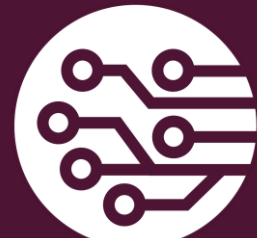


# MACHINE LEARNING LAB 2

Data Structures, NumPy Arrays,  
SciPy Functions



MUNADI SIAL



SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE  
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

# Data Structures

Python offers 4 built-in data types for storing collections of data:

1. **Lists:** A collection of data which is **ordered**, **changeable** and **allows duplicate members**
2. **Tuples:** A collection of data which is **ordered**, **unchangeable** and **allows duplicate members**
3. **Sets:** A collection of data which is **unordered**, **unchangeable** and **does not allow duplicate members**
4. **Dictionaries:** A collection of data which is stored in *key:value pairs* and is **ordered**, **changeable** and **does not allow duplicate members** (Dictionaries were unordered before Python 3.7)

# Data Structures

Examples of the collection data are given below. Notice the bracket types.

```
x_list = [1, 2, 3, 4, 5]
```

```
x_tuple = (1, 2, 3, 4, 5)
```

```
x_set = {1, 2, 3, 4, 5}
```

```
x_dictionary = {"name" : "Ali"  
                 "age"  : 31  
                 "result": False}
```

# Lists

- A List is a collection of data that is
  - Ordered
  - Changeable (Mutable)
  - Allows Duplicate Members
- A List is a sequence of values (called items or elements)
- Whereas a string is a sequence of characters, a list can be a sequence of any data type
- Lists are among the commonly used data types in python
- Lists are used extensively in ROS such as in reading laser data
- A List works similar to the arrays in C++

# Lists - Creation

- To create a list, the simplest way is to use square brackets to enclose the items and use commas to separate the items

```
x = [1, 2, 3, 4, 5]
```

```
print(x)
```

Output:

```
[1, 2, 3, 4, 5]
```

# Lists

- Lists can hold different types of data:

```
a = [11, 2, 93, 401, 560]
b = [1.5, 6.6, 7.3, 8.9]
c = ["apple", "banana", "cherry"]
d = [True, True, False, True, False]
e = []
f = list("ROBOT")
```

```
print(a)
print(b)
print(c)
print(d)
print(e)
print(f)
```

Output:

```
[11, 2, 93, 401, 560]
[1.5, 6.6, 7.3, 8.9]
['apple', 'banana', 'cherry']
[True, True, False, True, False]
[]
['R', 'O', 'B', 'O', 'T']
```

# Lists

- The same list can also hold different types of data:

```
g = [31.5, "Robotics", 4, True, 9]
```

```
print(g)
```

Output:

```
[31.5, 'Robotics', 4, True, 9]
```

# Lists

- Lists can nest other lists:

```
h = ["wheels", 35, 5.2, [10, 20], 50]
```

```
print(h)  
print(h[3])  
print(h[3][0])  
print(h[3][1])
```

Output:

```
['wheels', 35, 5.2, [10, 20], 50]  
[10, 20]  
10  
20
```



# Lists

- Lists allow duplicates of data:

```
j = ["apple", "banana", "cherry", "apple", "watermelon"]  
  
print(j)
```

Output:

```
['apple', 'banana', 'cherry',  
'apple', 'watermelon']
```

# Accessing Items in a List

- To access individual items in a list, square brackets are used which contain the index number of the item
- The index of the first item starts at zero
- The index of the last item is (number of items – 1)

0	1	2	3	4	5	6	7
---	---	---	---	---	---	---	---

```
my_list = [14, 20, 93, 41, 56, 77, 38, 62]
```

```
print(my_list[0])  
print(my_list[1])  
print(my_list[5])  
print(my_list[7])
```

Output:

```
14  
20  
77  
62
```

# Changing Items in a List

- The value of an item in a list can be changed by using its index

0	1	2	3	4	5	6	7
---	---	---	---	---	---	---	---

```
my_list = [14, 20, 93, 41, 56, 77, 38, 62]  
print(my_list)
```

```
my_list[2] = 37  
print(my_list)
```

Output:

```
[14, 20, 93, 41, 56, 77, 38, 62]  
[14, 20, 37, 41, 56, 77, 38, 62]
```

# Accessing Items with Negative Index

- Items can also be indexed from the end. This is done by using negative numbers for the index
- The negative indexing starts from -1 (not zero)

0	1	2	3	4	5	6	7
<code>my_list = [14, 20, 93, 41, 56, 77, 38, 62]</code>							
-8	-7	-6	-5	-4	-3	-2	-1

```
print(my_list[-1])  
print(my_list[-2])  
print(my_list[-3])  
print(my_list[-8])
```

Output:

```
62  
38  
77  
14
```

# Changing Items with Negative Index

- Negative indexing can also be used to change the value of items

-8	-7	-6	-5	-4	-3	-2	-1
----	----	----	----	----	----	----	----

```
my_list = [14, 20, 93, 41, 56, 77, 38, 62]  
print(my_list)
```

```
my_list[-5] = 500  
print(my_list)
```

Output:

```
[14, 20, 93, 41, 56, 77, 38, 62]  
[14, 20, 93, 500, 56, 77, 38, 62]
```

# Accessing Range of Items

- The access a range of items, the colon (:) is used (slice operation)
- Note the post-colon number is NOT included in the index

0	1	2	3	4	5	6	7
---	---	---	---	---	---	---	---

```
my_list = [14, 20, 93, 41, 56, 77, 38, 62]
```

```
print( my_list[2:5] )
```

0	1	2	3	4	5	6	7
---	---	---	---	---	---	---	---

```
print( my_list[2:] )
```

0	1	2	3	4	5	6	7
---	---	---	---	---	---	---	---

```
print( my_list[:5] )
```

0	1	2	3	4	5	6	7
---	---	---	---	---	---	---	---

Output:

```
[93, 41, 56]
```

```
[93, 41, 56, 77, 38, 62]
```

```
[14, 20, 93, 41, 56]
```

# Changing Range of Items

- The range of items, accessed with the colon (:), can be changed

0	1	2	3	4
---	---	---	---	---

```
my_list = ["apple", "banana", "cherry", "orange", "mango"]  
print( my_list)
```

```
my_list[2:4] = ["grapes", "melon"]  
print( my_list)
```

Output:

```
['apple', 'banana', 'cherry', 'orange', 'mango']  
['apple', 'banana', 'grapes', 'melon', 'mango']
```

# Changing Range of Items

- If you insert more items than you replace, the list will increase in length
- In the example below, banana and cherry are replaced by 3 items



```
my_list = ["apple", "banana", "cherry", "orange"]  
print( my_list)
```

```
my_list[1:3] = ["carrot", "potato", "turnip"]  
print( my_list)
```

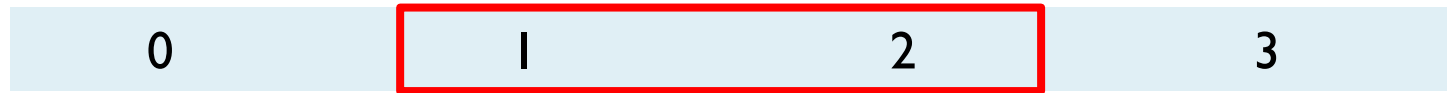
Output:

```
['apple', 'banana', 'cherry', 'orange']  
['apple', 'carrot', 'potato', 'turnip', 'orange']
```



# Changing Range of Items

- If you insert less items than you replace, the list will decrease in length
- In the example below, banana and cherry are replaced by 1 item



```
my_list = ["apple", "banana", "cherry", "orange"]  
print( my_list)
```

```
my_list[1:3] = ["strawberry"]  
print( my_list)
```

Output:

```
['apple', 'banana', 'cherry', 'orange']  
['apple', 'strawberry', 'orange']
```

# Checking Items in List

- The **in** and **not in** keywords can be used to check for items in a list:

```
my_list = ["apple", "banana", "cherry", "orange"]
```

```
print( "orange" in my_list )
```

```
print( "mango" in my_list )
```

```
print( "orange" not in my_list )
```

```
print( "mango" not in my_list )
```

Output:

```
True  
False  
False  
True
```

# Looping a List

- The **for** loop can be used to iterate through the items of a list
- The number of times the loop executes is equal to the number of items
- The iterator (fruit) takes the value of each item of every iteration

```
my_list = ["apple", "banana", "cherry", "orange"]
```

```
for fruit in my_list:  
    print(fruit)
```

Output:

```
apple  
banana  
cherry  
orange
```

# Looping a List

- If the indices are needed in the loop, then the **range** and **length** functions can be used

```
my_list = [1,2,3,4,5]

for i in range(len(my_list)):
    my_list[i] = 2 * my_list[i]

print(my_list)
```

Output:

```
[2, 4, 6, 8, 10]
```

# List Functions

- List functions (also called methods) are useful functions that can be used with a list
- Python provides the following list functions:

- Append()
- Insert()
- Extend()
- Remove()
- Pop()
- Count()
- Index()
- Reverse()



Functions to Add Items



Functions to Remove Items

# Functions to Add Items

- To add items, Python provides 3 list functions:
  - Append
  - Insert
  - Extend
- The **append** function adds an item to the end of a list:

```
listA = ["apple", "banana", "cherry"]  
print(listA)
```

```
listA.append("mango")  
print(listA)
```

Output:

```
['apple', 'banana', 'cherry']  
['apple', 'banana', 'cherry', 'mango']
```

# Functions to Add Items

- The **insert** function adds an item to a certain index
- The items are readjusted and the list size increases

```
listA = ["apple", "banana", "cherry"]  
print(listA)
```

```
listA.insert(1, "mango")  
print(listA)
```

Output:

```
['apple', 'banana', 'cherry']  
['apple', 'mango', 'banana', 'cherry']
```

# Functions to Add Items

- The **extend** function adds a list to another list

```
listA = ["apple", "banana", "cherry"]  
print(listA)
```

```
listB = ["carrot", "onion"]  
listA.extend(listB)  
print(listA)
```

- Another way to do this is to use the addition operation to concatenate the lists  
(`listA = listA + listB`)

Output:

```
['apple', 'banana', 'cherry']  
['apple', 'banana', 'cherry', 'carrot', 'onion']
```



# Functions to Remove Items

- To remove items, Python provides 2 list functions:
  - Pop
  - Remove
- The **remove** function can remove a specified item:

```
listA = ["apple", "banana", "cherry", "mango"]  
print(listA)
```

```
listA.remove("cherry")  
print(listA)
```

Output:

```
['apple', 'banana', 'cherry', 'mango']  
['apple', 'banana', 'mango']
```

# Functions to Remove Items

- The **pop** function can remove an item from a specific index:

```
listB = ["apple", "banana", "cherry", "mango"]  
print(listB)
```

```
listB.pop(1)  
print(listB)
```

Output:

```
['apple', 'banana', 'cherry', 'mango']  
['apple', 'cherry', 'mango']
```

# Other List Functions

- The **count** function returns the total number of a specified item

```
listC = [1,1,3,4,3,5,7,4,3,7,8,3,2]  
number = listC.count(3)  
print(number)
```

Output:

4

- The **index** function returns the index of the first occurrence of a specified item

```
index_val = listC.index(3)  
print(index_val)
```

Output:

2

# Other List Functions

- The **reverse** function reverses the order of the items in the list

```
listA = ["apple", "banana", "cherry", "mango"]  
listA.reverse()  
print(listA)
```

Output:

```
['mango', 'cherry', 'banana', 'apple']
```

# 2-D Lists

- The following is a 2-D list (a list of lists)
- The element of such lists can be accessed by multiple indexes
- The first index selects among the “items” of the outer list
- The second index selects among the items of the inner sub-list
- Remember that indexing in python starts from zero

```
my_2d_list = [[11, 22, 33, 44, 55],  
              [71, 27, 42, 99, 58],  
              [61, 62, 63, 64, 65],  
              [83, 85, 19, 24, 21]]
```

```
print( my_2d_list[2][3] )
```

Output:

64

# Strings - Review

- Python supports the string data type which is an array of characters

```
h = "Manipulator"  
print(h)
```

Output:

**Manipulator**

- You can get individual characters with square brackets

```
print(h[0])  
print(h[1])  
print(h[8])
```

Output:

**M  
a  
p**

- You can get the number of characters with the len() function

```
g = len(h)  
print(g)
```

Output:

**11**

# Strings - Review

- You can concatenate strings easily in python

```
c = "Computer"  
d = "Vision"  
e = c + d  
print(e)
```

Output:

**ComputerVision**

```
f = c + " " + d  
print(f)
```

Output:

**Computer Vision**

- You can check if a character is present in the string with the "in" keyword

```
print("t" in c)  
print("s" in c)
```

Output:

**True**  
**False**

# Strings - Review

- The **for** loop can go through a sequence of characters
- The iterable object will be a string variable in this case

```
for i in "PYTHON":  
    print(i)
```

Output:

P  
Y  
T  
H  
O  
N



# Dictionaries

- A dictionary is another data structure
- A dictionary is like a list but somewhat more general
- In a list, index positions have to be integers; in a dictionary, index positions can be (almost) any type
- A Dictionary is a collection of data that is
  - Ordered (as of Python 3.7)
  - Changeable (Mutable)
  - Does NOT Allow Duplicate Members

# Dictionaries - Creation

- A dictionary is a mapping of a set of indices (called keys) to a set of values
- Each key maps to a value
- Each key-value pair is an item of a dictionary
- To create a dictionary, key-value pairs are enclosed in braces

```
eng2span = {'one': 'uno',  
            'two': 'dos',  
            'three': 'tres'}
```

```
print(eng2span)
```

Output:

```
{'one': 'uno', 'two': 'dos', 'three': 'tres'}
```

# Dictionaries - Accessing

- The following dictionary maps English (keys) to Spanish (values)

```
eng2span = {'one': 'uno',  
            'two': 'dos',  
            'three': 'tres'}
```

- The keys are used as indexes to look up the corresponding values

```
print (eng2span['one'])  
print (eng2span['two'])  
print (eng2span['three'])
```

Output:

```
uno  
dos  
tres
```

# Dictionaries - Accessing

- The following dictionary maps English (keys) to Numbers (values)

```
eng2num = {'one': 1,  
           'two': 2,  
           'three': 3}
```

- The keys are used as indexes to look up the corresponding values

```
print (eng2num['one'])  
print (eng2num['two'])  
print (eng2num['three'])
```

Output:

```
1  
2  
3
```

# Dictionaries – Changing Value

- Consider the following dictionary which maps prices

```
prices = {'burger': 250,  
          'sandwich': 150,  
          'pizza': 400}
```

- The value can be changed by using its key:

```
prices['sandwich'] = 180  
print(prices)
```

Output:

```
{'burger': 250, 'sandwich': 180, 'pizza': 400}
```

# Importing Modules

- In python, we use the **import** keyword to use libraries (modules) which contain functions and classes
- There are many libraries commonly used in python such as NumPy, Pandas, OpenCV, Matplotlib, SciPy, Tensorflow, Keras, PySerial, Math and Datetime etc
- We will use the Math library as an example:

```
import math  
var = math.sqrt(64)  
print(var)
```

8.0

- We can also assign an *alias* for the library:

```
import math as mt  
var = mt.sqrt(25)  
print(var)
```

5.0

# Importing Modules

- The math library contains some common mathematical functions:

```
print( math.sqrt(64) )
print( math.ceil(1.5) )
print( math.floor(1.5) )
print( math.pi )
print( math.inf )
print( math.sin(30) )
print( math.cos(30) )
print( math.tan(30) )
print( math.log(5) )
print( math.log10(5) )
print( math.radians(180) )
print( math.degrees(math.pi) )
print( math.exp(4) )
```

```
8.0
2
1
3.141592653589793
inf
-0.9880316240928618
0.15425144988758405
-6.405331196646276
1.6094379124341003
0.6989700043360189
3.141592653589793
180.0
54.598150033144236
```

# NumPy

- NumPy (Numerical Python) is a library used for creating computationally efficient arrays for matrix operations

```
import numpy as np
A = np.array([[1, 2, 3],
              [4, 5, 6]])
```

- Numpy offers wide range of functions on the arrays

$A + B$

$A - B$

$A * B$

$A / B$

`np.dot(A, B)`

`np.sum(A, axis=0)`

`np.mean(A, axis=0)`

`np.std(A, axis=1)`



# SciPy

- SciPy (Scientific Python) is a library used to implement different scientific functions and is compatible with numpy arrays

```
from scipy.optimize import root
def equation(x):
    return x + x*sin(x)
output = root(equation, 0)
print(output.x)
```

- In machine learning, SciPy is used mostly for minimization functions, sparse matrices, interpolation etc.

# Lab Tasks

- Download the materials from LMS
- Perform the Lab Tasks given in the manual
- Convert the completed manual into .pdf and submit on LMS