# MIS4311
# Machine Learning Applications

Fall 2025

Lecture #3

# Data Types

Text Type: str

Numeric Types: int, float, complex

Sequence Types: list, tuple, range

Mapping Type: dict

Set Types: set, frozenset

Boolean Type: bool

# Data Types

| | |
|---|---|
| `x = str("Hello World")` | **str** |
| `x = int(20)` | **int** |
| `x = float(20.5)` | **float** |
| `x = complex(1j)` | **complex** |
| `x = list(("apple", "banana", "cherry"))` | **list** |
| `x = tuple(("apple", "banana", "cherry"))` | **tuple** |
| `x = range(6)` | **range** |
| `x = dict(name="John", age=36)` | **dict** |
| `x = set(("apple", "banana", "cherry"))` | **set** |
| `x = bool(5)` | **bool** |

# Lists

List items are ordered, changeable, and allow duplicate values.
list1 = ["apple", "banana", "cherry", "apple", "cherry"]

list1.append("orange")

list1.insert(1, "orange")  #add specified index

list1.extend(list2)

list1.remove("banana")

list1.pop(1)  #remove specified index

del list1[0]  or del list1

list1.sort()  #Sort List Alphanumerically  list1.sort(reverse = True)

list1.reverse()

x = list1.copy()

# Tuples

Tuples are used to store multiple items in a single variable.

*A tuple is a collection which is ordered and **unchangeable**.

*Tuples allow duplicate values:

tuple1 = ("apple", "banana", "cherry", "apple", "cherry")

tuple2 = ("abc", 34, True, 40, "male")

Once a tuple is created:

*you cannot add items to it.
*You cannot remove items in a tuple
*The del tuple1 can delete the tuple completely

# Sets

A set is a collection which is both *unordered* and *unindexed*

- Sets are unordered, so you cannot be sure in which order the items will appear.
- Duplicate values will be ignored

```
set1 = {"apple", "banana", "cherry"}
set2 = {"apple", "banana", "cherry", "apple”, ”melon”}
```

Once a set is created, you cannot change its items, but you can add new items.

```
set1.add("orange")

set1.update(set2)      #add sets

set1.remove("banana")
```

# Dictionaries

Dictionaries are used to store data values in key:value pairs.
A dictionary is a collection which is ordered, changeable and does not allow duplicates (with same key)

```
dict1 = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
```

Select by index: dict1["brand"],

x = dict1.keys(),  x = dict1.values(),  x = dict1.items()

dict1.pop("model"),  dict1.popitem() #remove last item

del dict1["model"]  or  del dict1

dict1.clear()  #not delete but empties the dict1

# Summarizing

There are four collection data types in the Python programming language:

**List** is a collection which is ordered and changeable. Allows duplicate members.

**Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.

**Set** is a collection which is unordered and unindexed. No duplicate members.

**Dictionary** is a collection which is unordered and changeable. No duplicate members.

# If – Else Statements

Equals: a == b
Not Equals: a != b
Less than: a < b
Less than or equal to: a <= b
Greater than: a > b
Greater than or equal to: a >= b

# If – Else Statements

```
a = 3
b = 5
if b > a:
  print("b is greater than a")
```

```
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

# If – Else Statements

```
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

# Logical operators (and - or)

```
a = 4
b = 3
c = 5
if a > b and c > a:
  print("Both conditions are True")
```

```
if a > b or a > c:
  print("At least one of the conditions is True")
```

# Nested If

*You can have if statements inside if statements

```
a = 22
if a > 10:
  print("Above ten,")
  if a > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```

# For Loops

* A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

```
fruits = ["apple", "banana", "cherry"]        # Looping Through a list
for x in fruits:
  print(x)


for x in "banana":      # Looping Through a String
  print(x)
```

# The break Statement

- break statement can stop the loop

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":     # Exit the loop when x is "banana"
    break
```

# The continue Statement

- continue statement can stop the current iteration of the loop, and continue with the next:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":                    #do not print “banana”
    continue
  print(x)
```

# The range() Function

```
for x in range(6):
  print(x)
```

*range(6) is not the values of 0 to 6, but the values 0 to 5.

```
for x in range(2, 6):       #the values from 2 to 5.
  print(x)
```

```
for x in range(2, 30, 3):   #Increment the sequence with 3 (default is 1)
  print(x)
```

# Nested Loops

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
    print(x, y)                #Print each adjective for every fruit:
```

# The while loop

```
i = 1
while i < 10:
    print(i)          #Print i as long as i is less than 10
    i += 1
```

*You can use break and continue statements in while loops

# Functions

*A function is defined using the **def** keyword:

```
def my_function():
  print("Hello from a function")
```

*To call a function, use the function name followed by parenthesis:

```
my_function()   #To use a function you should call it
```

# Arguments

*Information can be passed into functions as arguments

```
def my_function(name):
  print(" Hello”, name)
```

*Arguments are specified after the function name, inside the parentheses.

```
my_function("Erhan")
my_function("Tugba")
```

# Multiple Arguments

*You can add as many arguments as you want, just separate them with a comma.

```
def my_function(name, surname):
  print(name, surname)

my_function(”Sinan", ”Demir")
```

# Keyword Arguments

*You can also send arguments with the *key = value* syntax.

```
def my_function(student3, student2, student1):
  print("The youngest student is ”, student3)

my_function(student1 = ”Erhan", student2 = "Tugba", student3
= "Levent")
```

# Default parameter value

*If we call the function without argument, it uses the default value:

```
def my_function(country = ”Turkey"):
  print("I am from ”, country)

my_function("Sweden")
my_function()
my_function("Brazil")
```

# Passing a List as an Argument

- You can send any data types of argument to a function (string, number, list, dictionary etc.)

```
def my_function(food):
  for x in food:
    print(x)


fruits = ["apple", "banana", "cherry"]
my_function(fruits)
```

# Return Values

*To let a function return a value, use the return statement:

```
def calculate(y):
  return 4 * y

print(calculate(2))
print(calculate(6))
print(calculate(9))
```

# What is Module

*Modules refer to a file containing Python statements and definitions.

*A file containing a set of functions you want to include in your application.

*Consider a module to be the same as a code library.

*A file containing Python code, for example: example.py, is called a module, and its module name would be example.

# Create a Module

```
def add(a, b):
        result = a + b
        return result
```

*save this function as example.py

# Use a Module

```
import example # use module name
example.add(4,5) # call methods in the module
```

# Variables in a Module

```
person1 = {
"name": ”Elif",
 "age": 21,
"country": ”Turkey" }
```

*save this dictionary as module2.py

# Use an existed variable

```
import module2
a = module2.person1["age"]
print(a)
```

# Import with renaming

```
import example as ex        # use module name as ex
ex.add(4,5)                 # call methods in the module
```

---

```
import module2 as mdl2
a = mdl2.person1["age"]
print(a)
```

# from...import statement

```
def greeting(name):
        print("Hello, " + name)
person1 = {
        "name": ”Elif",
        "age": 21, "country":
        ”Turkey" }
```

*Save as module3.py

---

```
from module3 import person1    # only person1 variable can be used
print (person1["age"])    # not mymodule.person1["age"]
```

# Built-in Modules

```
import math        # import built-in module math
print("The value of pi is", math.pi)
```

# Using the dir() Function

```
import numpy
x=dir(numpy) # to list all the function names (or variable names)
print(x)
```

# Python Packages

- A package can contain one or more relevant modules.
- Physically, a package is actually a folder containing one or more module files.

| | |
|---|---|
| **tkinter** | (Graphical User Interface (GUI)) |
| **sqlite3** | (SQL database applications) |
| **json** | (JavaScript Object Notation) |
| **regex** | (Regular expressions) |
| | |
| **django** | (Web framework) |
| **kivy** | (IOS application) |

# Introduction to Machine Learning

```
import numpy                            #numeric python
import matplotlib.pyplot as plt         #matlab plot package

x = numpy.random.uniform(0.0, 5.0, 15)
#Create uniformly distributed numbers
#mean=0 and standard deviation=5 for 15 points

y = numpy.random.normal(5.0, 1.0, 100000)   #normal distribution

plt.plot(x)       #plot data itself
plt.show()

plt.hist(x, 5)    #plot histogram with 5 bars
plt.show()
```

# Introduction to Machine Learning

v = [99,86,87,88,111,86,103,87,94,78,77,85,87,86,87] **#speed of cars**

mean=numpy.**mean**(v) #average
median=numpy.**median**(v) #mid-point
deviation=numpy.**std**(v) #standard deviation
variance=numpy.**var**(v) #variance

mean= 89.4
median= 87.0
standard dev= 8.670255667126161
variance= 75.17333333333333

# Introduction to Machine Learning

```
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]                    #age of the cars

y = [99,86,87,88,111,86,103,87,94,78,77,85,86] # #speed of cars

plt.scatter(x,y)                    #show distribution
```

# Next Week

- Data Processing
- Logistic Regression

Thank you for your participation ☺