ÖZYEĞİN UNIVERSITY

EE393 Python for Engineers

Dr. Orhan Gökçöl

orhan.gokcol@ozyegin.edu.tr

online

09.11.2020

2020-2021 Fall Semester

1

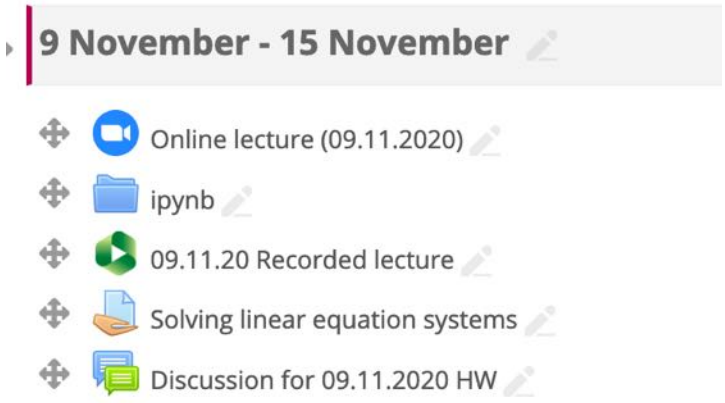
Agenda

- About mini projects
- Short review
- References to variables
- Generators
- Important python libs
  - sys
  - os
  - datetime
  - random
- Special methods in python with __
- Intr to scientific python
- (Extra)Intr to OOP

(Download to your local)

2

09.11.2020 | LMS resources



3

MIDTERM EXAM

SCHEDULED ON DECEMBER 21, 2020; 08:40

Online
(Details will be announced later)

4

EE393 Project

You are required to prepare a semester project work which contains an extensive use of Python and external libraries to solve an engineering problem **of your choice**.

Possible scenarios:

- Big Data –pattern matching, statistics, decision making (?)
- Machine Learning
- Image Recognition
- Web Apps development using Django or Flask
- IoT (Raspberry Pi is OK)
- Android app development
- AI (?)
- Engineering applications from your field
- **Use of Jupyter is encouraged**

- Group project –groups are to be formed up to 5 students
- At the end of the semester you need to submit your report and code(s) through LMS. In the report, each student's roles in the project and the differences of your code from the internet-resources must be given
- Not a copy-paste work of someone else's on the internet!! You may use some open-source codes/resources but there must be your contribution to the project. The level of your contribution will make a difference on your project grade.

5

EE

- Think of what you could do with Matlab. You can do in Python as well.
- DSP, FFT, Wavelets, RF Modulation, plotting signals
- Device interfacing – CANBus, Bluetooth, serial over USB, RS-485, RS-232
- Raspberry Pi
- Arduino
- Automating Test Equipment (like Oscilloscope, Digital Multimeters, Power supplies) – capturing data, presenting data, analysing data, managing data
- PCB CAD things
- Data visualization
- Network simülatör
- <…>

6

CS

- Cryptography
- Data Science
- Machine Learning
- Shell Programming (Linux mostly)
- Web things (with Djongo or Flask frameworks)
- UI development
- Mobile app development (with Kivy framework; IOS and Android OK)
- <…>

7

Other disciplines

You need to consider your departmental topics and areas where you need to take care of

- Data
- Algorithm
- Formulas
- Math

CE, ME => Core courses like Statics, Dynamics, Vibrations, Strength of Materials <…>

IE => Optimization, Operations Research, Linear/Nonlinear Programming, Decision Mating <…>

8

Review — Exception Handling

An exception is an **error** that happens during the execution of a program (Very similar to Java). Look at the code. Let's analyse:

```
x = float(input("Your number: "))
inverse = 1.0 / x
print("The inverse: ", inverse)
```

exception_handling.ipynb

if x=0, normally it should raise an error. However, using the following modification:

```
#exception handling
try:
    x = float(input("Your number: "))
    inverse = 1.0 / x
    print("The inverse: ", inverse)
except ZeroDivisionError:
    print("No zero in denominator ...")
finally:
    print("Execution completed.")
```

```
Your number:  0
No zero in denominator ...
Execution completed.
```

To catch all errors, do not specify exception name!

9

Tuples — Review

- A tuple is a sequence data type that is similar to the **list**. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.
- The main differences between lists and tuples are:
  - Lists are enclosed in brackets ([ ]), and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated.
  - Tuples can be thought of as **read-only** lists.

mytuple = (11, 22, 33)
mytuple[0] → **11**
mytuple[-1] → **33**
mytuple[0:1] → **(11,)**
The comma is required!

For more info on Tuples:
https://www.tutorialspoint.com/python/pdf/python_tuples.pdf

10

Creating a dictionary:

Review

There are several methods for creating a dictionary:

1- Standard way

```
myDict = {}     # empty dictionar
ymyDict = {1: 'ali', 2: 'veli'}       # dictionary with integer keys
myDict = {'name': 'Jon', 1: [2, 4, 3]} # dictionary with mixed keys
```

2- Using dict() function

```
myDict = dict({1:'OzU', 2:'Ozyegin'})
```

A=dict(m=8, n=9) ➔ {'m': 8, 'n': 9}

3- Using sequence having each item as a pair in a dict()

```
myDict = dict([(1,'Ozu'), (2,'Ozyegin')])
```

dict.ipynb

4- Comprehensions

```
mySquares = {x: x*x for x in range(6)}
print(mySquares) # Output: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

which is equivalent to:
```

mySquares = {}for x in range(6): mySquares[x] = x*x

11

Dictionary content methods

Review

- `my_dict.items()` – all the key/value pairs
- `my_dict.keys()` – all the keys
- `my_dict.values()` – all the values

```
#items, keys and values
person = dict()
person['fname'] = 'Joe'
person['lname'] = 'Jones'
person['age'] = 50
person['spouse'] = 'Natalie'
person['children'] = ['Ralph', 'Betty', 'Joey']
person['pets'] = {'dog': 'Lucky', 'cat': 'Boni'}
print (person.items()) #returns all items
for x,y in person.items(): access keys and values via items
    print (x,y)
```

```
dict_items([('fname', 'Joe'), ('lname', 'Jones'), ('age', 50
pouse', 'Natalie'), ('children', ['Ralph', 'Betty', 'Joey'])
ts', {'dog': 'Lucky', 'cat': 'Boni'})])
fname Joe
lname Jones
age 50
spouse Natalie
```

12

Dictionary views are iterable using for

```
for key in my_dict:
    print(key)
```

- prints all the keys

```
for key,value in my_dict.items():
    print (key,value)
```

- prints all the key/value pairs

```
for value in my_dict.values():
    print (value)
```

- prints all the values

Review

13

Zipping iterables

Review

- The **zip()** function takes iterables (can be zero or more), aggregates them in a tuple, and return it.

```
#zipping iterables
number_list = [1, 2, 3]
str_list = ['one', 'two', 'three']

# No iterables are passed
result = zip(number_list, str_list)

# Converting iterator to list
result_list = list(result)
print(result_list)
```

```
[(1, 'one'), (2, 'two'), (3, 'three')]
```

14

Functions –Extra Material

Review

Functions may have a variable number of arguments

- We can have both normal and keyword variable number of arguments.

```
def myFun(*argv):
    for x in argv:
        print (x)
myFun('Hello', 'OzU', 'and', 'World')
```

```
Hello
OzU
and
World
```

functions2.ipynb

```
def myFun2(arg1, *argv):
    print ("First argument :", arg1)
    for x in argv:
        print("Next argument through *argv :", x)
myFun2('Hey', 'Hello', 'OzU', 'to', 'World', 'TR')
```

```
First argument : Hey
Next argument through *argv : Hello
Next argument through *argv : OzU
Next argument through *argv : to
Next argument through *argv : World
Next argument through *argv : TR
```

15

kwargs

Review

- The special syntax ****kwargs*** in function definitions in python is used to pass a keyworded, variable-length argument list. We use the name *kwargs* with the double star. The reason is because the double star allows us to pass through keyword arguments (and any number of them).
  - You can use **kwargs to let your functions take an arbitrary number of keyword arguments ("kwargs" means "keyword arguments")
  - A keyword argument is where you provide a name to the variable as you pass it into the function.

```
def myFun3(**kwargs):
    for key, value in kwargs.items():
        print ("%s == %s" %(key, value))

myFun3(first ='OzU', mid ='is', last='Great')
```

```
first == OzU
mid == is
last == Great
```

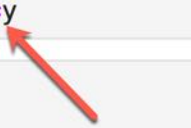
16

Global Variables and Global Constants

- Global variable: created by assignment statement written outside all the functions
  - Can be accessed by any statement in the program file, including from within a function
  - If a function needs to assign a value to the global variable, the global variable must be redeclared within the function
    - General format: `global variable_name`
- Reasons to avoid using global variables:
  - Global variables making debugging difficult
    - Many locations in the code could be causing a wrong variable value
  - Functions that use global variables are usually dependent on them
    - Makes function hard to transfer to another program
  - Global variables make a program hard to understand

```
global x,y
def ff():
    return x*y

x=10
y=21
print (ff())
```




```
210
```

Review

17

Inline functions - Lambda

Review

x = lambda a : a**2
t=x(5) ➔ the value of t is 5^2=25

x = lambda a, b : a * b
t=x(5, 6) ➔ the value of t is 5*6=30

There are popular and trickier uses ıf lambda.
See functions2.ipynb !!!

(Lambda may be removed from Python in the future)

18

Rules for coding!

Review

1. Think before you program!
2. A program is a human-readable essay on problem solving that also happens to execute on a computer. So, make it human readable!
3. The best way to improve your programming and problem solving skills is to practice!
4. A foolish consistency is the hobgoblin of little minds
5. Test your code, often and thoroughly
6. If it was hard to write, it is probably hard to read. Add a comment.
7. **All input is evil**, unless proven otherwise.
8. **A function should do one thing.**

19

VARIABLES -REVISITED

Variables are just **named references to objects**. Objects have types and categories and may be mutable, but names don't have these properties. Thus, all the following are true:

x = 42 # binds the name "x" to an integer object
x = "pc204" # binds the name "x" to the string object "pc204"
x = [1, 2, 3] # binds the name "x" to a list
y = ['a', x, 'c'] # binds the name "y" to a list which includes an embeded reference to another list which is ➔ y ['a', [1, 2, 3], 'c']

x[1] = 'b' # modify the second item in the original list. y is now ➔ ['a', [1, 'b', 3], 'c']

```
x = 42 # binds the name "x" to
x = "pc204" # binds the name "
x = [1, 2, 3] # binds the name
y = ['a', x, 'c'] # binds the
```

```
y
```

```
['a', [1, 2, 3], 'c']
```

```
x[1] = 'b'
```

```
y
```

```
['a', [1, 'b', 3], 'c']
```

(week6.ipynb)

20

A reference assigned to another reference is still just a reference…

```
x = [1, 2, 3]
z = x # both x and z reference the *SAME OBJECT*
y = ['a', z, 'c']
print(y)
x[1] = 'b' # this still changes the object that "y" references
print(y)
```

Output:

```
['a', [1, 2, 3], 'c']
['a', [1, 'b', 3], 'c']
```

(week6.ipynb)

21

Creating Copies of Objects

If you don't want x and y to share the same object, you need to create an explicit copy of the object...

x = [1, 2, 3]
y = x[:] # assigning a slice of the entire list creates a copy of the object
y gives ➔[1, 2, 3] # so now if I modify the original object, like this
x[1] = 'b' # so now if I modify the original object

Of course x is changed... to ➔ [1, 'b', 3]

But y # is unchanged.

this works for embeded references as well...
x = [1, 2, 3]
y = ['a', x[:], 'c']

(week6.ipynb)

22

Iterables

When we create a list, we can read its items one by one. Reading its items one by one is called **iteration**:

```
mylist = [1, 2, 3]
for i in mylist:
    print(i)
```

- Here, **mylist** is an ***iterable***. When we use a list comprehension, we create a list, and so an iterable:

```
mylist = [x*x for x in range(3)]
for i in mylist:
    print(i)
```

Handy, but comes with a price!!!

Everything we can use "for... in..." on is an iterable; lists, strings, files... These iterables are handy because we can read them as much as we wish, **but we store all the values in memory and this is not always what we want when you have a lot of values.**

23

Generators

- Generators are iterators, a kind of iterable **we can only iterate over once**. Generators do not store all the values in memory, **they generate the values on the fly**:

```
mygenerator = (x*x for x in range(3))
for i in mygenerator:
    print(i)
```

It is just the same as before. we used ( ) instead of [ ]. BUT, we **cannot** perform for i in mygenerator a second time since generators can only be used once: they calculate 0, then forget about it and calculate 1, and end calculating 4, one by one.

24

Returning from a function through a generator

yield is a keyword that is used like return, except the function will return a generator.

```
def createGenerator():
    mylist = range(3)
    for i in mylist:
        yield i*i
```

The first time the for calls the generator object created from your function, it will run the code in your function from the beginning until it hits yield, then it'll return the first value of the loop. Then, each other call will run the loop you have written in the function one more time, and return the next value, until there is no value to return.

```
mygenerator = createGenerator() # create a generator
print(mygenerator) # mygenerator is an object!
for i in mygenerator:
    print(i)
```

(week6.ipynb)

Here we assume our function will return a huge set of values that we will only need to read once. **when we call the function, the code we wrote in the function body does not run.** The function only returns the generator object, this is a bit tricky!!

25

Random (number) generators

There are many choices

See **random.ipynb**

```
import random
#randint to get an integer in an interval
guess = random.randint(1,100)
print (guess)

#random() itself gives a float between 0 and 1
print (random.random())

#use random.choice to get a random value in an iterator
dinner = random.choice(["meatball", "pizza", "pasta"])
print (dinner)

#it is possible to get random choices; more than one value
#in an iterator
dist=[3,5,7,-8,0,'a1','b']
distget=random.choices(dist,k=3)
print(distget)

#shuffle a list
random.shuffle(distget)
print(distget)
```

```
83
0.8995629580152779
pasta
['a1', -8, 3]
[3, 'a1', -8]
```

26

Previously- Modules

- Module is an organizational tool
- We put related functions together into the same file
- Avoid having multiple copies of the same code
- Functional decomposition (divide and conquer !!)
- Modules are files containing Python definitions and statements (for ex. **name.py**)
- A module's definitions can be imported into other modules by using "import *name*"
- The module's name is available as a global variable value

27

Important built-in modules

- **sys** Information about Python itself (path, etc.)
- **os** Operating system functions
- **shutil** Utilities for copying files and directory trees
- **cmp** Utilities for comparing files and directories
- **glob** Finds files matching wildcard pattern
- **re** Regular expression string matching
- **time** Time and date handling
- **datetime** Fast implementation of date and time handling
- **doctest, unittest** Modules that facilitate unit test

28

More….

- **pdb** Debugger
- **hotshot** Code profiling
- **pickle, cpickle, marshal, shelve** Used to save objects and code to files
- **getopt, optparse** Utilities to handle shell-level argument parsing
- **math, cmath** Math functions (real and complex) faster for scalars
- **random** Random generators (likewise)
- **gzip** read and write gzipped files
- **struct** Functions to pack and unpack binary data structures
- **StringIO, cStringIO** String-like objects that can be read and written as files (e.g., in-memory files)
- **types** Names for all the standard Python type

29

sys

- The sys module provides functions and variables used to manipulate different parts of the Python runtime environment. some of the important ones are in **week6.ipynb**.
- command line arguments ➔**argv**

```
1  import sys
2  for x in sys.argv:
3      print (x)
4
```

ozu_sys.py

```
(base) Orhans-MacBook-Pro:week6 orhang$ ls -la
total 2744
drwxr-xr-x   7 orhang  staff      224 Nov  8 07:56 .
drwxr-xr-x  22 orhang  staff      704 Nov  8 06:03 ..
drwxr-xr-x   3 orhang  staff       96 Nov  8 07:31 .ipynb_checkpoints
-rwxrwxrwx@  1 orhang  staff  1386361 Nov  8 07:53 ee393_week6.pptx
-rw-r--r--   1 orhang  staff       43 Nov  8 08:00 ozu_sys.py
-rw-r--r--   1 orhang  staff     4121 Nov  8 07:54 week6.ipynb
-rw-r--r--@  1 orhang  staff      165 Nov  8 06:45 ~$ee393_week6.pptx
(base) Orhans-MacBook-Pro:week6 orhang$ python ozu_sys.py 1 2 3 hello
ozu_sys.py
1
2
3
hello
(base) Orhans-MacBook-Pro:week6 orhang$
```

There are four arguments passed to the main

30

sys

- **sys.exit()** ➔causes the script to exit back to either the Python console or the command prompt. This is generally used to safely exit from the program in case of generation of an exception.
- **sys.maxsize** ➔ returns the largest integer a variable can take.
- **sys.path** ➔ This is an environment variable thatis a search path for all Python modules
- See **sys.ipynb** for some other uses

```
#sys
import sys
x = 150
y = 'Hello'
print ("max integer value a variable can take", sys.maxsize)
print ("Float info: ", sys.float_info)
print (sys.getdefaultencoding())
#Note that every string in Python takes additional 49-80 bytes of
#where it stores supplementary information, such as hash,
#length, length in bytes, encoding type and string flags
print ("Size of",y,"in bytes is :",sys.getsizeof(y))
#Note that numbers are 64 bit. An int will occupy 28 bytes
#Python numbers are limited only with memory
print ("Size of",x,"in bytes is :",sys.getsizeof(x))
print ("Current path is", sys.path)
print ("Python version :", sys.version)
print ("Int info :", sys.int_info)
print ("The platform you are running Python is : ", sys.platform)
print ("The directory prefix of python interpreter :", sys.prefix)
```

31

OS

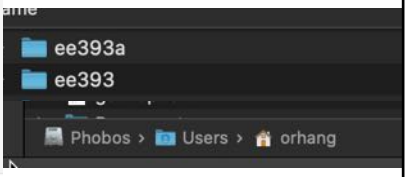
- The **OS module in python** provides functions for interacting with the operating system. **OS**, comes under **Python's** standard utility **modules**. This **module** provides a portable way of using operating system dependent functionality. The ***os*** and ***os**. path** **modules** include many functions to interact with the file system.
- See **os.ipynb** for some important other uses

In all the examples, you should change path and filenames to something appropriate to your computer!!!!! Note that paths in windows go like "C:\...\etc"

32

```
#miscellaneous os module methods
import os

directory = "ee393a"
parent_dir = "/Users/orhang/"
path = os.path.join(parent_dir, directory)
#create a directory
os.mkdir(path)
print("Directory {0:s} created".format(directory))
```




```
Directory ee393a created
```

```
#miscellaneous os module methods
import os

directory = "ee393a"
parent_dir = "/Users/orhang/"
path = os.path.join(parent_dir, directory)
#create a directory
os.mkdir(path)
print("Directory {0:s} created".format(directory))
```

Exception for already available object

```
FileExistsError                           Traceback (mo
<ipython-input-55-589abd829001> in <module>
      6 path = os.path.join(parent_dir, directory)
      7 #create a directory
```

33

datetime & time

- In Python, **date, time and datetime** classes provides a number of function to deal with dates, times and time intervals.
- Date and datetime are an object in Python, so when you manipulate them, you are actually manipulating objects and not string or timestamps. Whenever you manipulate dates or time, you need to import datetime function.
- The datetime classes in Python are categorized into main 5 classes.
  - **date** – Manipulate just date (Month, day, year)
  - **time** – Time independent of the day (Hour, minute, second, microsecond)
  - **datetime** – Combination of time and date (Month, day, year, hour, second, microsecond)
  - **timedelta**— A duration of time used for manipulating dates
  - **tzinfo**— An abstract class for dealing with time zones

See datetime.ipynb

34

Double underscore (__) in variable and method names

- The use of double underscore (__) in front of a name (specifically a method name) is not a convention; **it has a specific meaning to the interpreter**. Python manages these names and it is used to avoid name clashes

Double Underscore Before and After a Name (e.g. __init__ , __main( )__) :

- **These are special methods used by Python**. As far as one's concerned, this is just a convention, a way for the Python system to use names that won't conflict with user-defined names. You then typically override these methods and define the desired behaviour for when Python calls them.

35

__name__ and __main__

- When a file is run as a top-level program, it's **__name__** is set to **"__main__"** when it starts
- If a file is imported, __name__ is set to the **name of the module** as the importer sees it
- We can use this to package a module as a library, but allow it to run stand-alone also, by checking

```
if __name__ == '__main__':
# run in stand-alone mode do_whatever()
```

maintest.py

```
def f1():
    return "hello1"

def f2():
    return "hello2"

if __name__ == '__main__':
    print ("I am running in standalone")
    print (f1())
```

By importing the module:

```
import maintest
msg=maintest.f1()
print(msg)
```

36

Two interesting functions: eval and exec

eval: The *expression* argument is parsed and evaluated as a Python expression

x=1
eval('x+1') → outputs 2

- It may be useful for input processing.

exec : dynamic execution of Python code

```
#The following prints "hello"
exec('print("hello")')

program = '''
for i in range(3):
    print("Python is cool")
'''
exec(program)
```

37

You can use methods instead of functions by adding double underscore to the beginning and of the function name. For example:

```
x=3.78
print(int(x))
print(x.__int__())
```

or

instead of using `x==y`, try `x.__eq__(y)`
`__eq__, __ne__, __lt__, __gt__, __le__, __ge__` are all possibl

```
a=3
b=5
a.__add__(b)
```
→possible!!

38

Python Libraries for Science and Engineering

Many popular Python toolboxes/libraries:

- NumPy
- SciPy
- Pandas
- SciKit-Learn

All these libraries are installed on Anaconda

Visualization libraries

- matplotlib
- Seaborn (based on matplotlib)

and many more …

39

Python Libraries for Science and Engineering

NumPy:

- introduces objects for multidimensional arrays and matrices, as well as functions that allow to easily perform advanced mathematical and statistical operations on those objects
- provides vectorization of mathematical operations on arrays and matrices which significantly improves the performance

Link: http://www.numpy.org/

- many other python libraries are built on NumPy

40

Python Libraries for Science and Engineering

SciPy:

- collection of algorithms for linear algebra, differential equations, numerical integration, optimization, statistics and more
- part of SciPy Stack built on NumPy

Link: https://www.scipy.org/scipylib/

41

Python Libraries for Science and Engineering

Pandas: data manipulation and analysis

- adds data structures and tools designed to work with table-like data
- provides tools for data manipulation: reshaping, merging, sorting, slicing, aggregation etc.

Link: http://pandas.pydata.org/

- allows handling missing data

42

Python Libraries for Science and Engineering

SciKit-Learn:

- provides machine learning algorithms: classification, regression, clustering, model validation etc.
- built on NumPy, SciPy and matplotlib

Link: http://scikit-learn.org/

43

Python Libraries for Science and Engineering

matplotlib:

- python 2D plotting library which produces publication quality figures in a variety of hardcopy formats
- a set of functionalities similar to those of MATLAB
- line plots, scatter plots, barcharts, histograms, pie charts etc.

Link: https://matplotlib.org/

- relatively low-level; some effort needed to create advanced visualization

44

44

Python Libraries for Science and Engineering

Seaborn:

- based on matplotlib
- provides **high level interface** for drawing attractive statistical graphics
- Similar (in style) to the popular **ggplot2** library in **R**

Link: https://seaborn.pydata.org/

45

45

Quiz: Solve linear equation systems

- Examine **numpy_simple.ipynb** (available @ LMS)

```
"""
Solution of the following linear equation using numpy
3*x0 + x1   + 2*x2 = 9
x0   + 2*x1 + 4*x2 = 8
-x0  - 2*x1 +2.5*x2= 1
"""

import numpy as np
a = np.array([[3,1,2], [1,2,4] , [-1,-2,2.5]])
b = np.array([9,8,1])
x = np.linalg.solve(a, b)
print (x)
```

```
[2.         0.23076923 1.38461538]
```

Modify the program to solve

3 * x0 + x1 = 9
x0 + 2 * x1 = 8

46

Exercise | HW

Download **nov09.txt** which contains a 4x4 matrix and a 4x1 vector to form a 4x4 linear equations system. Develop a Python program which performs the followings:

- Solve the equation using numpy library. You will read the coefficients from the file and construct numpy arrays properly.
- Note that you'll print the solution on the console as well as write to a file "output.txt" which contains the equations in a fancy format and gives the solution.
- Verify that solution is correct. Develop a solutionTest function which returns true if the solution satisfies the equation system and false if not.

47

Sample input file and sample output

```
COEFFICIENTS
SIZE: 4
3.50 2.77 -0.76 1.80
-1.80 2.68 3.44 -0.09
0.27 5.07 6.90 1.61
1.71 5.45 2.68 1.71

7.31 4.23 13.85 11.55
```

```
Number of Equations : 4
EQUATIONS
----------
+3.5x1+2.77x2-0.76x3+1.8x4=7.31
-1.8x1+2.68x2+3.44x3-0.09x4=4.23
+0.27x1+5.07x2+6.9x3+1.61x4=13.85
+1.71x1+5.45x2+2.68x3+1.71x4=11.55
SOLUTION:
----------
x1=1.000000000000083
x2=1.0000000000000173
x3=1.000000000000025
x4=0.9999999999998221

Check if the solution satisfies the equations:
Equation   1  : Error is  0.0
Equation   2  : Error is  -8.881784197001252e-16
Equation   3  : Error is  -1.7763568394002505e-15
Equation   4  : Error is  -1.7763568394002505e-15
SOLUTION IS CORRECT WITH A TOTAL ERROR OF  -4.440892098500626e-15
```

48

OOP

(Extra)

- In all the programs we developed so far, we have designed our program around functions i.e. blocks of statements which manipulate data. This is called the *procedural programming*.
- There is another way of organizing our program which is **to combine data and functionality and wrap it inside** something called an **object**.
- This is called the ***object oriented*** programming paradigm. Most of the time we can use procedural programming, but when writing large programs or have a problem that is better suited to this method, you can use object oriented programming techniques.

OOP is a different approach for designing our software products

49

(Extra)

Classes and objects are the two main aspects of object oriented programming:

oop_basics.ipynb

A **class** creates a new *type* where **objects** are **instances** of the class.
The simplest class possible is shown in the following example.

```
class Person:
    def say_hi(self):
        print('Hello, how are you?')

p = Person()
p.say_hi()
```

An instance of the object is CREATED!

self keyword is used for the class method. Class methods have only one specific difference from ordinary functions - they must have an extra first name that has to be added to the beginning of the parameter list, but you **do not** give a value for this parameter when you call the method, Python will provide it. This particular variable refers to the object *itself*, and by convention, it is given the name self
It is like "this" pointer in C++ or "this" reference in Java.

50

(Extra)

__init__ method

This method is run as soon as an object of a class is instantiated (i.e. created). The method is useful to do any *initialization (it is like C++ constructor)*

```
class Person:
    def __init__(self, name):
        self.name = name

    def say_hi(self):
        print('Hello, my name is', self.name)
p = Person('Ali Veli')
p.say_hi()
```

We'll continue digging OOP principles later on

51

SEE YOU NEXT WEEK!!!

DR. ORHAN GÖKÇÖL
gokcol@gmail.com
orhan.gokcol@ozyegin.edu.tr

52