ÖZYEĞİN UNIVERSITY

EE393 Python for Engineers

Dr. Orhan Gökçöl
orhan.gokcol@ozyegin.edu.tr

online

WEEK #2

2020-2021 Fall Semester

1

There will be no ZOOM lecture for next week, 19.10.2020.

I'll put a recorded video.

2

LAST WEEK:

- Course logistics –**important:** you are required to follow the online lectures using your Win/Mac/Linux/iPad with a proper Python environment
- Python intro –history, motivation etc.
- Various environments (local & cloud) to develop in Python
- Anaconda navigator
  - **conda** package manager and environment management system
  - **spyder** python editor (with **Ipython** support)
  - **jupyter** notebooks
- You are assumed to have some background in one of the programming languages (C, Java, Matlab, C#, PHP etc. all fine)
- OzU LMS is used for course-related activities:
  - Announcements
  - Forum (Support + activity)
  - Handouts
  - Resources
  - HW/class work/project submissions
  - Scores

3

Installing Python :

- If you are on Mac or Linux, it is installed by default (most of the time!)

You also need to install a package manager for python : **pip** to have a hassle-free and up-to-date environment
- **Library updates**
- **Binary updates etc.**

www.python.org/downloads

- ****RECOMMENDED****Alternatively, you may use a python distribution packaged with engineering libraries.
One of the most important distribution is Anaconda.
This option is recommended.

ANACONDA

https://www.anaconda.com/

- Another option is to use Cloud environments. We may use one of them time to time : Google colab.

4

Rules for online classes

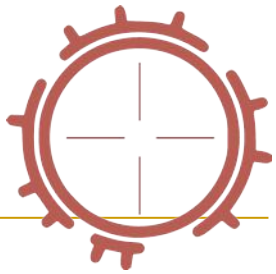
- Be ready before the ZOOM class starts! (i.e. before 08:40)
- Microphones are muted by default. Do not unmute unless you are permitted.
- **It is not permitted to**
  - record the class video or audio
  - stream the class through social media or web
  - take pictures of the screens

- If you have a question, please type your question in the chat area. While I answer your question, you may unmute your mic & talk to me.
- You may prefer not to turn your camera on. It is OK for me. However, I prefer you turn it on.
- Respect the privacy of your friends and me.
- OzU Distance learning regulations are applied!!

5

EE393 aims at

- Make you comfortable in using Python to solve problems occurred in your engineering disciplines.
- Review important engineering libraries such as numpy, scipy, pandas and matplotlib.
- Semester mini projects -using python in engineering problem solving.



6

Course Logistics

- **Weekly Schedule**
  - **MONDAY, 08:40 (via Zoom Meetings)**
- **Course web support and distance learning**
  - I'll use OzU LMS
- **Course e-mail (use this email for communication)**
  - ***orhan.gokcol@ozyegin.edu.tr***
- **Communication with course instructor and TAs**
  - We'll have an active communication channel through LMS
  - Details will be available through support forum

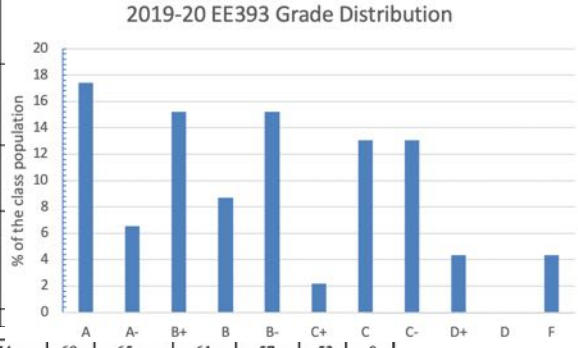
(email) orhan.gokcol@ozyegin.edu.tr

(business phone) (532)483-4545

7

ASSESSMENT METHODS, WEIGHTS AND RULES

Type	Weight	Assessment Method	Implementation Rule	Makeup Rule
Final Exam	25	Online with Respondus proctoring	Final exam will be closed books and notes. No calculation or communication devices will be allowed during the exam.	BÜT exam will serve as the makeup for the final exam
Midterm Exam	20	Online with Respondus proctoring	Midterm exam will be closed books and notes. No calculation or communication devices will be allowed during the exam.	Only valid excuses with an official report are accepted to qualify for a midterm makeup. At most one makeup will be given in the course due to health reports.
Quiz	5	Offline	Take-home quiz. Students will have a limited amount of time (e.g. 6 hours) to complete a task. Copying the work of others is not permitted.	---
Homework	25	Offline		
Project	15	Offline		
Other	10	Online collaboration - such as using LMS discussion boards effectively		
Total	100%			

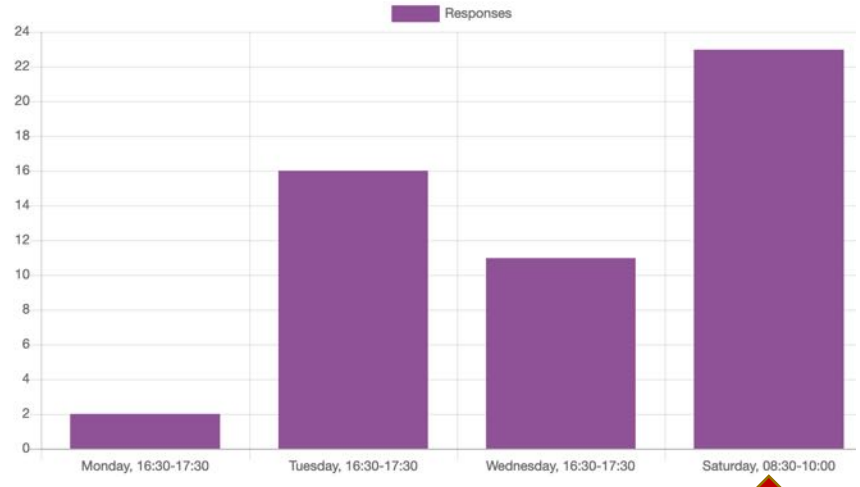


Points	100-90	89-85	84-80	79-75	74-70	69-66	65-62	61-58	57-54	53-50	0-49
Grade	A	A-	B+	B	B-	C+	C	C-	D+	D	F

Academic honesty is important. You will face with disciplinary actions for the cases against academic honesty

8

Course support – Virtual Office Hour



(ZOOM)

9

Python For Engineers

Python for Scientists and Engineers

Python for Scientists and Engineers is now free to read online. The table of contents is below, but please read this important info before.

Python for Scientists and Engineers was the first book I wrote, and the one I still get queries about. It was out of print for a long time, till now, and has been updated with help from the community.

There are a few new sections, using the highly technical name of *New Stuff*. The biggest change has been to the Machine Learning section. I have added some of my best articles here, and some new stuff.

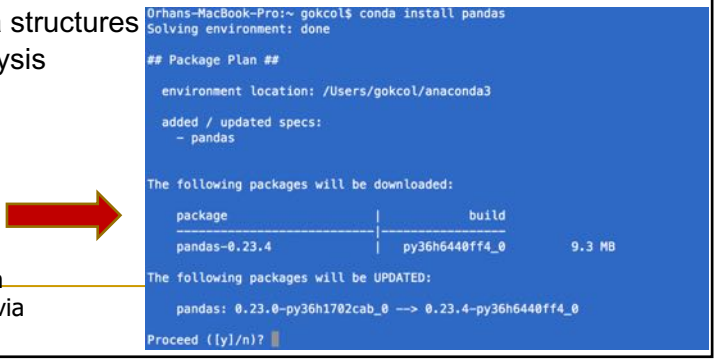
10

Important Python libraries

- **math** –math functions (trigonometry etc)
- **numpy** –fundamental library for scientific computing which defines data types and data operations on them (linear algebra, random numbers, n-dimensional arrays …)
- **scipy** –fundamental library for scientific computing
- **matplotlib** –plotting library
- **pandas** –data structures and data analysis

conda package manager

(can be used from command line or via anaconda GUI)



11

Importing modules

- The procedure to use modules is as follows (example module: **math**):

import math

- We need to mention the name of the module before using any function defined within itself. For example, if we wish to use the sqrt() function, we need to write **math.sqrt(2)**, which will calculate 2 using the math module.

from math import *

- "*" signifies everything (in the language of regular expressions). Hence, it means that we can use all the functions without the need to write the name of the module separated with a "." (dot operator) before the name of the function.

from math import sqrt

- This statement only imports the **sqrt()** function being used. Using this style of Python programming is essential when you have limited memory and are energy-conscious because you don't need to import unnecessary objects. You still need to write the name of module separated with a "." (dot operator) before the name of the function.

import numpy as np

- We can rename the module/library name to make a short hand.

12

Assignments to variables and simple arithmetics

```
# Assignments
val1 = 3
val2 = 7; val3=5;
val4 = 13
v1,v2,v3=3,8,10
vv = v1+v2+v3
print ("vv is ", vv)
vnew = vv**val1 #taking exponent
print (vv," to the power of ", val1, "is ", vnew)
remainder = vnew%val4
print (vnew, "divided by ", val4, " gives ", remainder, " as the remainder")
print (vnew, "/", val4, "=", vnew/val4, "(real division)")
print (vnew, "/", val4, "=", vnew//val4, "(integer division)")

val4 += val2  #val4 = val4 + val2; similarly -=, *=, /=, %=, //=, **=
val1 **= val3
print ("val4 :", val4)
print ("val1 :", val1)
```

```
vv is  21
21  to the power of  3 is  9261
9261 divided by  13  gives  5  as the remainder
9261 / 13 = 712.3846153846154 (real division)
9261 / 13 = 712 (integer division)
val4 : 20
val1 : 243
```

week2.ipynb

13

Data Types in Python

In Python, we can define new **data types** as and when required. There are some **built-in** data types for handling numbers and characters. Different data types occupy different amounts of memory.

- **Logical:** This type of data stores boolean values True or False and can be operated by boolean operators such as **and, or, not**. Logical operations are resultant of comparison operators (< , >, <=, >=, !=, ==)

Example

```
x = 2
y = -6
print ("Test 1: ", x>2)
print ("Test 2: ", x<=2)
print ("Test 3: ", -1<x<5)
print ("Test 4: ", not x)
print ("Test 5: ", 3<y<8 and x>0)
print ("Test 6: ", 3<y<8 or x>0)
myRes = (x+y)<0
print ("Test 7: ", myRes and 0<x<=2)
```

```
Test 1:  False
Test 2:  True
Test 3:  True
Test 4:  False
Test 5:  False
Test 6:  True
Test 7:  True
```

week2.ipynb

Bitwise operators: See
https://www.geeksforgeeks.org/python-bitwise-operators/

14

Logical operators as operated on data (1)

As you can see from the example, Python is very flexible in creating Boolean expressions.

The following operators are used to construct logical expressions :

<, <=, >, >=, ==, !=, not, and, or

The following named keywords are used to indicate true and false :

True, False

15

Identity Operators

- **is** : Returns True if both variables are the same object
- **is not** : Returns True if both variables are not the same object
- Note that, the **==** operator compares the values of both the operands and checks for value equality. Whereas **is** operator checks whether both the operands refer to the same object or not.

week2.ipynb

```
#Identity operators
name1 = "Jon Snow"
name2 = "Jon Snow"
iden1 = name1 is name2
iden2 = name1 == name2
iden3 = name1 is not name2
print (iden1, iden2, iden3)
```

```
False True True
```

16

Membership Operators

- **in** : Returns True if a sequence with the specified value is present in the object
- **not in** : Returns True if a sequence with the specified value is not present in the object
- Note that, membership operators are generally used for "lists" and "dictionaries". Below see a simple example with strings. A string is a special type of list

week2.ipynb

```
#Membership operators
name1 = "Jon Snow"
searchKey = "JON"
r1 = "Jon" in name1
r2 = "Dragon" not in name1
r3 = searchKey in name1
print (r1, r2, r3)
```

```
True True False
```

17

Data Types

- **Numeric:** There are four types of numeric data types. Python has arbitrary precision for them. As a result, the limit of the length of these numbers is subject to the availability of memory :
  - **int**: Integers
  - **float**: Floating point numbers
  - **complex**: Complex numbers

```
import sys
sys.float_info
```

```
sys.float_info(max=1.7976931348623157e+308,
max_exp=1024, max_10_exp=308, min=2.2250738
585072014e-308, min_exp=-1021, min_10_exp=-
307, dig=15, mant_dig=53, epsilon=2.2204460
49250313e-16, radix=2, rounds=1)
```

- Plain **int** is unbounded. Limited by computer's memory

week2.ipynb

18

Data Types

- An important method : **type**

```
#type
a = 2
print (type(a))
print (type (2.0))
c = 2 + 3j
print (type(c))
msg = "Hello world\n"
print ("Type of ", msg, "is", type (msg))
print (type (True))
```

```
<class 'int'>
<class 'float'>
<class 'complex'>
Type of  Hello world
 is <class 'str'>
<class 'bool'>
```

week2.ipynb

- conversions

```
#conversions
print (int (2.5*7))
print (float(a))
print (complex(a))
print (str(c))
str2 = "3.45"
print (float(str2)*1.5)
print (complex(2,3))
```

```
17
2.0
(2+0j)
(2+3j)
5.175000000000001
(2+3j)
```

19

Data types

- To employ even more precision, we can use the decimal module that has the function **Decimal()**, which returns the number as stored by the computer:

```
In [18]: from decimal import Decimal

In [19]: Decimal(pi)
Out[19]: Decimal('3.14159265358979311599796346854418516159057617187 5')
```

- Extreme cases: **infinite number**, **NoneType** and **Not a Number**
- NoneType: **None** is frequently used to represent the absence of a value

week2.ipynb

```
import math
bound = math.inf
print (2*bound)

d=math.nan
e = None

print(d)
print (e, type(e))
```

```
inf
nan
None <class 'NoneType'>
```

20

Data Types

- Check if a variable is of a specific type (int, float or complex)

```
xx = 3.75
print (isinstance(xx,int))
print (isinstance(math.pi,float))
```

```
False
True
```

- (*optional content*)Fixed precision math

We'll see this notation later

```
In [51]: import decimal as dec

In [52]: dec.getcontext().prec=4

In [53]: dec.Decimal(4)/dec.Decimal(23)
Out[53]: Decimal('0.1739')

In [60]: dec.Decimal('-3.14').as_integer_ratio()
Out[60]: (-157, 50)
```

```
In [35]: 4/23
Out[35]: 0.17391304347826086
```

```
In [38]: a,b=dec.Decimal('3.14').as_integer_ratio()

In [39]: a
Out[39]: 157

In [40]: b
Out[40]: 50
```

21

Some built-in functions on data

- **divmod()** to find a quotient and remainder simultaneously
- **round()** to round a number to a **certain decimal point**
- **pow()** to raise a number to a certain power (*requires math library*)
- **abs()** for absolute value

```
#built-in functions
print (divmod(100,21)) #quotient is 4 and remainder is 14
print (abs(-5.89))
x = pow(4, 3, 5) #4^3 mod 5
print (x)
print (pow(3,-0.87))
print (round(pow(3,-0.87),5))
```

```
(4, 16)
5.89
4
0.38450722824313915
0.38451
```

22

Investigate **math** library

```
In [70]: import math

In [71]: dir(math)
Out[71]:
['__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 'acos',
 'acosh',
 'asin',
 'asinh',
 'atan',
 'atan2',
 'atanh',
 'ceil',
 'copysign',
 'cos',
 'cosh',
 'degrees',
 'e',
 'erf',
 'erfc',
 'exp',
 'expm1',
 'fabs',
 'factorial',
 'floor',
 'fmod',
 'frexp',
 'fsum',
 'gamma',
 'gcd',
 'hypot',
 'inf',
 'isclose',
 'isfinite',
 'isinf',
 'isnan',
 'ldexp',
 'lgamma',
 'log',
 'log10',
 'log1p',
 'log2',
 'modf',
 'nan',
 'pi',
 'pow',
 'radians',
 'sin',
 'sinh',
 'sqrt',
 'tan',
 'tanh',
 'tau',
 'trunc']
```

TWO constants:

pi
e

```
In [72]: math.__name__
Out[72]: 'math'

In [73]: math.__spec__
Out[73]: ModuleSpec(name='math',
loader=<_frozen_importlib_external.ExtensionFileLoader object at
0x10f57fef0>, origin='/Users/gokcol/anaconda3/lib/python3.6/lib-
dynload/math.cpython-36m-darwin.so')
```

23

CONTAINER DATA TYPES
Sequences : String

- Python deals with three important types of sequences : Strings, lists and tuples
- A **string** is merely a sequence of characters. Lowercase and uppercase characters have different encoding; thus, strings are case-sensitive.
- String values are given in single or double quotation

  x='Hello world'

  y="Hello \n world" #escape characters are possible in " … "
- Try **dir(str)** from the iPython console or in jupyter to see all of the available string methods. Use __doc__ property to get an help on the usage. For ex:

```
In [75]: str.isdigit.__doc__
Out[75]: 'S.isdigit() -> bool\n\nReturn True if all characters in S are digits\nand there is at
least one character in S, False otherwise.'
```

24

Quick look to "Strings"

```
# create a string
ss = str(42) # convert another data type into a string
s = 'I like you'
print (ss, s)

# examine a string
print (s[0]) # returns 'I'
print (len(s)) # returns 10

# string slicing
print (s[:6]) # returns 'I like'
print (s[7:]) # returns 'you'
print (s[-1]) # returns 'u'
```

```
42 I like you
I
10
I like
you
u
```

week2.ipynb

25

Basic string methods

```
# basic string methods (does not modify the original)
print (s.lower()) # returns 'i like you'
print (s.upper()) # returns 'I LIKE YOU'
print (s.startswith('I')) # returns True
print (s.endswith('you')) # returns True
print (s.isdigit()) # returns False (returns True if every character in the string is a digit)
print (s.find('like')) # returns index of first occurrence (2)
print (s.find('hate')) # returns -1 since not found s.replace('like', 'love') # replaces all ins
# concatenate strings
s3 = 'The meaning of life is'
s4 = '42'
print(s3 + ' ' + s4) # returns 'The meaning of life is 42'
# remove whitespace from start and end of a string
s5 = ' steak and cheese '
print (s5.strip()) # returns 'steak and cheese'
```

```
i like you
I LIKE YOU
True
True
False
2
-1
The meaning of life is 42
steak and cheese
```

week2.ipynb

26

Sequences: Lists

- A list is an ordered set of objects, **irrespective of its data type**. It can be defined by enclosing a set of values in square brackets.

```
#Lists
#A list is an ordered set of objects, irrespective of its data type.
#It can be defined by enclosing a set of values in square brackets
myList = [1,2.0,'ali',"0zU",3+2j, None, math.nan, math.pi]
print (myList)
print (myList[2])
myList[3]="veli"
print (myList)
```

```
[1, 2.0, 'ali', '0zU', (3+2j), None, nan, 3.141592653589793]
ali
[1, 2.0, 'ali', 'veli', (3+2j), None, nan, 3.141592653589793]
```

- Modifications and iterations on lists is possible using powerful loop structures and it is the basis for many engineering computations related to using many data in matrices and arrays

We'll discuss lists in detail next week

27

Sequences: Tuples

- A tuple is an immutable list. They are defined using parentheses ( ) instead of brackets [ ] For example:
- Their elements, once defined, cannot be altered. Elements of a **list** can be altered using their indices, though.

```
#Tuples
myList = (1,2.0,'ali',"0zU",3+2j, None, math.nan, math.pi)
print (myList)
print (myList[2])
myList[3]="veli" #will give an error
print (myList)
```

```
(1, 2.0, 'ali', '0zU', (3+2j), None, nan, 3.141592653589793)
ali
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call la
<ipython-input-98-3cc5e43fe019> in <module>
      3 print (myList)
      4 print (myList[2])
----> 5 myList[3]="veli" #will give an error
      6 print (myList)

TypeError: 'tuple' object does not support item assignment
```

We'll discuss tuples in detail next week

28

Sets and Frozen Sets

*(*Optional content*)*

- The **set** data type is the implementation of a mathematical set. It is an unordered collection of objects. That is, indexing has no meaning.
- Unlike sequence objects, such as list and tuple, where elements are ordered, sets do not have such requirements.
- Sets do not permit duplicity in the occurrence of an element, that is, an element either exist 0 or 1 times. This is not the case with list and tuple objects.
- Frozen sets are immutable sets

We may discuss sets later

```
In [83]: mySet=set(['h','e','l','l','o',1,2.0,3+4j])

In [84]: mySet
Out[84]: {(3+4j), 1, 2.0, 'e', 'h', 'l', 'o'}

In [85]: mySet=frozenset(['h','e','l','l','o',1,2.0,3+4j])

In [86]: mySet
Out[86]: frozenset({(3+4j), 1, 2.0, 'e', 'h', 'l', 'o'})
```

Python sets are implemented using a hash table. The *order* will change across runs - or within the same run if you *insert* a lot.

29

Mappings & Dictionaries

- Mapping is a scheme of defining **data** where each element is identified with a **key** called “hash tag.” Therefore, the element can be accessed by referring to the key. One of the data types in this category is a **dictionary**.

```
#Simple dictionary
person={'age':21,'height':1.79, 'location':'Istanbul'}
player={'nick':"ghost", "level":17, "guns":['arrow','laser','pistol']}
print (player)
print (player['guns'])
```

```
{'nick': 'ghost', 'level': 17, 'guns': ['arrow', 'laser', 'pistol']}
['arrow', 'laser', 'pistol']
```

30

More on Variables & Assignments

```
In [91]: a=5

In [92]: b,c,d=7,'hello',True

In [93]: [x,y,z]=[3,5,7]

In [94]: name="Ali Can\r"

In [95]: b
Out[95]: 7

In [96]: mylist=[x,y,z]=[3,5,7]

In [97]: x
Out[97]: 3

In [98]: mylist
Out[98]: [3, 5, 7]

In [99]: a = b = c = 15
```

Assignment Operators

Operator	Example
=	v = a+b
+=	v +=a ⇒ v = v + a
-=	v -=a ⇒ v = v - a
/=	v /=a ⇒ v = v / a
//=	v //=a ⇒ v = v // a
*=	v *=a ⇒ v = v * a
**=	v **=a ⇒ v = v ** a
%=	v %=a ⇒ v = v % a

// is used for floor division and can be used on both **int** and **float** numbers

31

Different number systems

```
In [127]: bin(198)
Out[127]: '0b11000110'

In [128]: hex(7985)
Out[128]: '0x1f31'

In [129]: oct(11467)
Out[129]: '0o26313'
```

See how they are given: 0b , 0x, 0o

int() bin()
oct() hex()

32

Console input

```
#Simple input
name = input("What's your name? ")
print("Nice to meet you " + name + "!")
age = input("Your age? ")

print("So, you are already " + age + " years old, " + name + "!")

age = int(input("Your age? "))
print (age*age)
```

```
What's your name?  Jon Snow
Nice to meet you Jon Snow!
Your age?  29
So, you are already 29 years old, Jon Snow!
Your age?  29
841
```

Remember: INPUTS ARE STRINGS

33

Python conditions and if statements

- The execution of statements can be done when a condition is met or not met. Hence, we define a scope which depends on the logical condition.
- If delta is greater than or equal to zero
  - Then calculate real roots
- If delta is less than zero
  - Then print “no real roots”
- Lets go with an example:

34

```
#----------------------------------------
#This program calculates real roots of ax^2 + bx + c = 0
#a,b,c are real numbers and taken from the console input
#(^C)O.Gökçöl, EE393 intro example
#12.10.2020 : Initial version, v1.0
#----------------------------------------
import math
a = input ("Enter coefficient a :")
b = input ("Enter coefficient b :")
c = input ("Enter constant term c :")
print ("The equation is: ", a,"x^2+",b,"x+",c,"=0",sep='')
a = float(a); b= float(b); c=float(c)
delta = b*b -4*a*c
if delta>=0:
    x1 = (-b + math.sqrt(delta))/2*a
    x2 = (-b - math.sqrt(delta))/2*a
    print ("Roots are; x1=",round(x1,4),"x2=",round(x2,4))
else:
    print ("No real root\n")
print ("End of execution. Thank you!")
```

```
Enter coefficient a : 2
Enter coefficient b : 7
Enter constant term c : 2
The equation is: 2x^2+7x+2=0
Roots are; x1= -1.2554 x2= -12.7446
End of execution. Thank you!
```

Be careful! Code is not perfect ☺ I'm sure you can do better

: is used to create a conditional scope

Pay attention to indentation

For statements which are executed if a condition is met, we put **:** at the end of condition and the statements are written with a constant indent

IMPORTANT indents must be arranged properly

35

Nested Decision Structures and the `if-elif-else` Statement

- A decision structure can be nested inside another decision structure
  - Commonly needed in programs
  - Example:
    - Determine if someone qualifies for a loan, they must meet two conditions:
      - Must earn at least $30,000/year
      - Must have been employed for at least two years
    - Check first condition, and if it is true, check second condition

36

The `if-elif-else` Statement

- `if-elif-else` statement: special version of a decision structure
  - Makes logic of nested decision structures simpler to write
    - Can include multiple **`elif`** statements
  - Syntax:

```
if condition1:
      statements
elif condition2:
      statements
else:
      statements
```

IMPORTANT indents must be arranged properly

37

While loop

- With the **while** loop we can execute a set of statements as long as a condition is true. See the example

```
#program to print all integers which are between 1 and 1000
#and dividable by 103 or 121
#EE393 Simple looping -exercise; 12.10.2020

number = 1
count = 0
while number<=1000:
    if number%203==0 or number%221==0:
        print ("Found.", number, "is dividable to 17 or 21")
        count += 1
    number += 1
print ("A total of ", count, " number found.")
print ("Program ended")
```

Pay attention To indentation

```
Found. 203 is dividable to 17 or 21
Found. 221 is dividable to 17 or 21
Found. 406 is dividable to 17 or 21
Found. 442 is dividable to 17 or 21
Found. 609 is dividable to 17 or 21
Found. 663 is dividable to 17 or 21
Found. 812 is dividable to 17 or 21
Found. 884 is dividable to 17 or 21
A total of  8  number found.
Program ended
```

38

Additional – **breaking the loop**

```
x = 0
while True:
        x = x + 2
        if x > 21:
                break
        print (x)
```

```
x = 0
while x < 21:
        x = x + 2
        print (x)
```

39

for – iteration on containers which have more than one value

- As opposed to what's known, **for** is not a loop, but an iterator.
- Iterators in python take all the values in a given container such as a list or a string.
- They are used for iterating over a sequence. It works as an iterator method, rather than a loop structure
- With **for** we can execute a set of statements, once for each item in a list, tuple, set, dictionary etc.
- See the examples:

40

```
#simple for loop examples
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)
```

```
apple
banana
cherry
```

```
for x in "banana":
    print (x)
```

```
b
a
n
a
n
a
```

- Pay careful attention to indentation
- See how **"in"** membership operator is used. As long as the condition given in the for-header is true, inside statements are executed.
- **"in"** membership operator is used at nearly all the for-constructions.
- We can "break" **for** as we do in while loop.

41

range

```
#for loop with range()
for x in range(0,4):
    print ("We're on time %d" % (x))
```

```
We're on time 0
We're on time 1
We're on time 2
We're on time 3
```

range() is a built-in function of Python. It is used when a user needs to perform an action for a specific number of times.
The **range()** function is used to generate a sequence of numbers.
range() is commonly used in for looping. Most common use of range() function in Python is to iterate sequence type (List, string etc..) with for and while loop.

range can be used with three arguments. See week2.ipynb:
•**start**: integer starting from which the sequence of integers is to be returned
•**stop:** integer before which the sequence of integers is to be returned.
The range of integers end at stop – 1.
•**step:** integer value which determines the increment between each integer in the sequence

42

Class exercise

1- A Fibonacci sequence is the integer sequence of 0, 1, 1, 2, 3, 5, 8....

The first two terms are 0 and 1. All other terms are obtained by adding the preceding two terms. This means to say the nth term is the sum of (n-1)th and (n-2)th term. #1 and #2 below are class activities (we'll do together).

- #1:List all Fibonacci numbers on console screen up to n terms (class activity)
- #2: List all Fibonacci numbers on console screen between two integers m,n

In all versions: Make sure the user inputs are **int** and properly given.

If you have questions during the activity, ask it using the chat area

43

About submitting hws and quizzes

- Minimum requirements:
  1. *Good commenting* – At the beginning of your program, do include your name, surname, class, department and a short description of the program
  2. Inside your code, do use comments when there is some complex task that is needed to be explained
  3. A separate *readme* file where you explain how your program is used with sample inputs and expected outputs. It is a text file – use Markdown language (MD) to prepare your readme
  4. Your .py program
  5. ALTERNATIVELY, send your .ipynb instead of (3. and 4.)

There will be penalty for the following cases : 1) Plagiarism (-100 pts), not using proper commenting (up to -30 pts), not providing a readme file (-50 pts), improper readme file (up to -50 pts), python code behaves unexpectedly (up to -70 pts)

44

SEE YOU NEXT WEEK!!!

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

45