ÖZYEĞİN UNIVERSITY

EE393 Python for Engineers

Dr. Orhan Gökçöl

orhan.gokcol@ozyegin.edu.tr

online

WEEK #3&4

2020-2021 Fall Semester

1

Agenda

- Short review
- ASCII and UTF-8 encoding/decoding in Python
- Formatted **print** output using string methods
- Common operations on lists
- List comprehensions
- Input and output using Files
- An introduction to user defined functions
- Dot notation
- File I/O

2

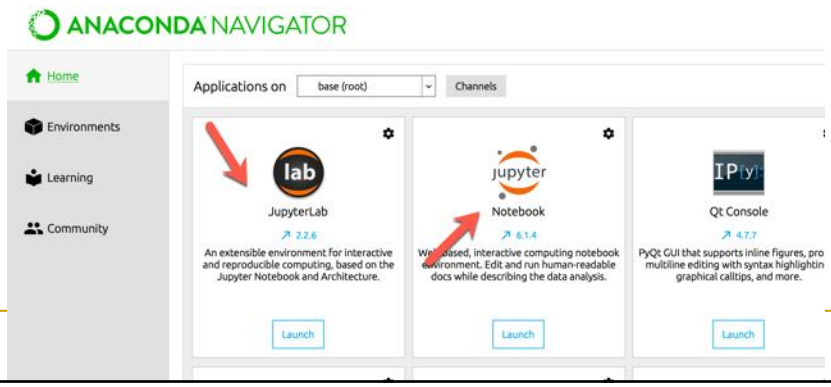
PREVIOUSLY:

- We use Anaconda distribution for python environment:
  - **conda** package manager and environment management system
  - **spyder** python editor (with **Ipython** support)
- Python libraries: math, numpy, scipy, matplotlib, pandas
  - import math
  - from math import *
  - from math import sqrt
- Data Types : int, float, complex, bool, str
- Logical operators: < , >, <=, >=, !=, ==
- Arithmetic operators: +, -, *, /, %, **, +=, -=, *=, /=
- Quiz1 & self-study

3

Jupyter Notebooks

- We use Jupyter to write Python code
- Jupyter is a free, open-source, interactive web tool known as a computational notebook, which data scientists can use to combine software code, computational output, explanatory text and multimedia resources in a single document.
- Computational notebooks have been around for decades, but Jupyter in particular has exploded in popularity over the past couple of years.



4

Logic

PREVIOUSLY:

- Many logical expressions use *relational operators*:

Operator	Meaning	Example	Result
==	equals	1 + 1 == 2	True
!=	does not equal	3.2 != 2.5	True
<	less than	10 < 5	False
>	greater than	10 > 5	True
<=	less than or equal to	126 <= 100	False
>=	greater than or equal to	5.0 >= 5.0	True

- Logical expressions can be combined with *logical operators*:

Operator	Example	Result
and	9 != 6 and 2 < 3	True
or	2 == 3 or -1 < 5	True
not	not 7 > 0	False

5

PREVIOUSLY:

- Sequences-Strings
  - Definition ➔s="Hello\nworld"
  - Many methods in use with strings such as:
    - s.lower(), s.upper(), s.startswith('..'), s.endswith('..'), s.isdigit(), s.find('..'), s.replace('..','..'), s.strip() ….
    - String concentation using +
    - String slicing ➔s[:6], s[3:], s[2:3], s[-1] etc.
    - First index is zero
- Sequences-Lists
  - Ordered set of objects. Very much like strings, except each member can be of any type. For example:
    - a=[1,2.0,'ali',"OzU",3+2j, None, NaN, math.pi]
  - Modifications and iterations on lists is possible using powerful loop structures
- Sequences-Tuples : They are immutable lists. For ex:
  - z= (21, 1.78, True, 'age','height', 'is a student?')

READ ONLY ACCESS

6

PREVIOUSLY:

- Sets
  - Mathematical sets – unordered collection of objects; indexing has no meanings – set functions (union, difference) are possible
  - Sets do not permit duplicity in the occurrence of an element, that is, an element either exist 0 or 1 times. Example:
  - my_set = {1,2,3,4,3,2} #last two members have no effect
- Frozen sets : They are immutable sets. For ex:
  - mylist = ['apple', 'banana', 'cherry']
    x = frozenset(mylist) → x became frozen set

7

PREVIOUSLY:

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**.

```
In [87]: person={'age':21,'height':1.79, 'location':'Istanbul'}

In [88]: person
Out[88]: {'age': 21, 'height': 1.79, 'location': 'Istanbul'}

In [89]: person['age']
Out[89]: 21

In [90]: type(person)
Out[90]: dict
```

8

Membership operator: **in** *very important*

- The membership operator checks if a value(s) of variables is a member of a specified **sequence**. If the member is found, it returns the boolean value **True**; otherwise, it returns **False**:

```
In [101]: 'hello' in 'hello world'
Out[101]: True

In [102]: 'name' in 'hello world'
Out[102]: False

In [103]: a=3

In [104]: b=[1,2,3,4,5]

In [105]: a in b
Out[105]: True

In [106]: 10 in b
Out[106]: False
```

A **sequence** could be a string, a list, a tuple, a dictionary etc.

9

Encodings for strings

- As we saw before, strings are just a list of characters
- In most cases, characters are just single bytes.
  - The ASCII encoding standard maps between single byte values and the corresponding characters
- More recently, characters are two bytes.
  - **Unicode** uses two bytes per characters so that there are encodings for glyphs (characters) of other languages
  - Python uses unicode

10

ASCII

Code Point Range	Class
0 through 31	Control/non-printable characters
32 through 64	Punctuation, symbols, numbers, and space
65 through 90	Uppercase English alphabet letters
91 through 96	Additional graphemes, such as [and \
97 through 122	Lowercase English alphabet letters
123 through 126	Additional graphemes, such as { and |
127	Control/non-printable character (DEL)

11

ASCII encoding through **ord()** and **chr()**

```
#Character encoding
str = "Hello"
for char in str:
    print (ord(char))
```

```
72
101
108
108
111
```

```
for i in range(33,140):
    print (i, "==>", chr(i))
```

```
33 ==> !
34 ==> "
35 ==> #
36 ==> $
37 ==> %
38 ==> &
39 ==> '
40 ==> (
```

```
letters="abcdefghijklmnopqrstuvwxyz"
Letters="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
for x in letters:
    print (ord(x),end="/")

print ("\n")
for x in Letters:
    print (ord(x),end="/")
```

encoding.ipynb

```
97/98/99/100/101/102/103/104/105/106/107/108/109/110/111/112/113/114/115/116/117/118/119/120/121/122/

65/66/67/68/69/70/71/72/73/74/75/76/77/78/79/80/81/82/83/84/85/86/87/88/89/90/
```

12

There are more characters than we can type

Note that,

- Our keyboards don't have all the characters available to us, and it's hard to type others into strings.
  - Backspace?
  - Return?
- We use **backslash** escapes to get other characters into strings
- "\b" is backspace
- "\n" is a newline (pressing the Enter key)
- "\t" is a tab
- "\uXXXX" is a Unicode character, where XXXX is a code and each X can be 0-9 or A-F.
  - http://www.unicode.org/charts/
  - **Must precede the string with "u" for Unicode to work**

13

UTF-8 Encode and Decode

- See examples

Each UTF-8 character is 2 bytes in length

```
#UTF-8
print ("Orhan Gökçöl".encode("utf-8"))
print ("résumé".encode("utf-8"))
```

```
b'Orhan G\xc3\xb6k\xc3\xa7\xc3\xb6l'
b'r\xc3\xa9sum\xc3\xa9'
```

encoding.ipynb

```
s = u'ö'
print(s.encode('utf-8'))
euroSign = u'€'
print ("Euro sign utf-8 code is ", euroSign.encode('utf-8'))
```

```
b'\xc3\xb6'
Euro sign utf-8 code is  b'\xe2\x82\xac'
```

```
print (b'\xc3\xb6'.decode('utf-8')) #c3b6 in Hexadecimal
```

```
ö
```

```
print ("Euro Sign : ", b'\xe2\x82\xac'.decode('utf-8'))
```

```
Euro Sign :  €
```

14

A few words about print( ) function

- `print` function uses a space to separate items in print statement. To suppress the print between items printed use
  - Special argument `sep='`*`delimiter`*`'`
    - Causes `print` to use *`delimiter`* as item separator
    - Delimiter is usually no space in this case ‘’, but can be anything

```
>>> print ('One','Two','Three', sep='')
```

Output:

```
OneTwoThree
```

15

print( )

formatted_io.ipynb

```
#special delimeter in print function
print ('One','Two','Three', sep='')

#it is possible to change end of line (\n) character with
#something else.
for i in range(4):
    print(i, end=" :-) ")
print ("\n")

print ('12'.zfill(5)) #will print '00012'
print ('12'.ljust(5)) #will print '12 '
print ('12'.rjust(5)) #will print ' 12'
print ('12'.zfill(5).ljust(10)) #will print '00012 '
print ('12'.zfill(5).rjust(10)) #will print ' 00012'
print(192,168,178,42,sep=".") #will print 192.168.178.42
```

16

Formatted output for numbers

formatted_io.ipynb

- `format` function can format floating point numbers up to 12 significant digits
  - **`format(numeric_value, format_specifier)`**
  - ***Returns string*** containing formatted number
  - Format specifier typically includes **precision** and **data type**
    - Format: 'decimal_width(type)'
    - Ex: '2f' for 2 digits after decimal point
    - Numbers >= 5 are rounded up
  - Comma separator is possible for numbers greater than 999 using
    - ",.2f" ➔2 digits after decimal point and separation is applied

```
#format function
value = 123.4567
x = format(value, ".2f")
print (x)
```

```
123.46
```

```
big_n = 91234562.1678342
x = format(big_n, ",.2f")
print (x)
```

```
91,234,562.17
```

```
monthly_pay = 5000.0
annual_pay = monthly_pay * 12
#line continuation symbol is \
print('Your annual pay is ', \
        format(annual_pay, ',.2f'), " TRY", sep='')
```

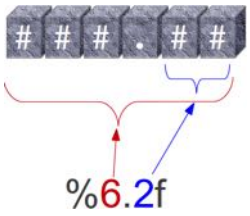
```
Your annual pay is 60,000.00 TRY
```

17

Formatted output for numbers

formatted_io.ipynb

- We can also set the width of number:
  - '[field_width] decimal_width(type)'
- Ex: `'12.2f'` will create a 12 character wide floating point number with two digits after the decimal




```
big_n = 91234562.1678342
x = format(big_n, "20.5f")
print (x)
x = format(big_n, "20,.5f")
print (x)
```

```
      91234562.16783
 91,234,562.16783
```

20 character wide

```
num1 = 127.899
num2 = 3465.148
num3 = 3.776
num4 = 264.821
# Display each number in a field of 7 spaces
# with 2 decimal places.
print(format(num1, '7.2f'))
print(format(num2, '7.2f'))
print(format(num3, '7.2f'))
print(format(num4, '7.2f'))
```

```
 127.90
3465.15
   3.78
 264.82
```

18

formatted_io.ipynb

Formatting Integers

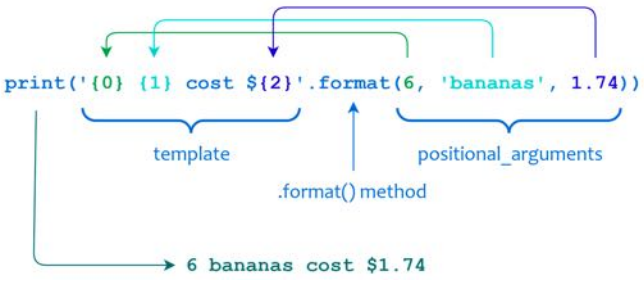
- To format an integer using `format` function:
  - Use **`d`** as the type designator
  - Do not specify precision
  - Can set field width or comma separator

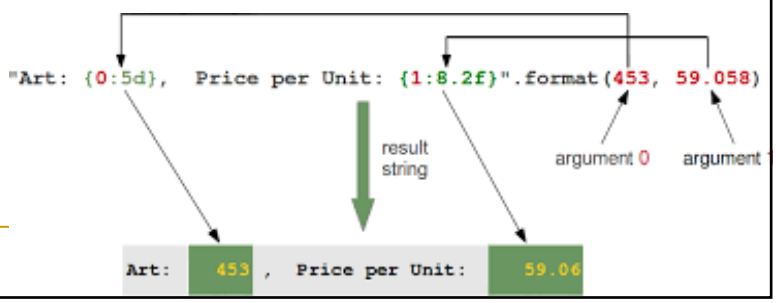
```
number = 123456
print(format(number,"d"))
print(format(number,",d"))
print(format(number,"10d"))
print(format(number,"10,d"))
```

```
123456
123,456
    123456
   123,456
```

19

The String .format() Method (simple use)





20

formatted_io.ipynb

```
#advanced uses of format in outputs
N = 61
print('Hello Ozu. There are {0} {1}, (all online :)) {other}.'
      .format(N, ' students ', other ='in the class'))

# using format() method with number
print("No. of Students :{0:3d}, average gpa :{1:6.2f}".
       format(N, 090.875))

# Changing positional argument
print("Second argument: {1:3d}, first one: {0:7.2f}".
       format(47.42, 11))

print("Students: {a:5d},  Average: {p:8.2f}".
      format(a = N, p = 90.875))
```

```
Hello Ozu. There are 61  students , (all online :)) in the class.
No. of Students : 61, average gpa : 90.88
Second argument:  11, first one:   47.42
Students:    61,  Average:    90.88
```

21

format identifiers

Escape	Description
%d	Decimal integers (not floating point)
%i	Same as %d
%o	Octal number
%u	Unsigned decimal
%x	Hexadecimal lowercase
%X	Hexadecimal uppercase
%e	Exponential notation, lowercase "e"
%E	Exponential notation, uppercase "E"
%f	Floating point real number
%F	Same as %f
%g	Either %f or %e, whichever is shorter
%G	Same as %g but uppercase
%c	Character format
%r	Repr format (debugging format)
%s	String format
%%	A percent sign

22

String Interpolation / f-Strings

- It lets us use embedded Python expressions inside string constants.

```
#String interpolation
name = "Jon Snow"
f'Hello, {name}!'
```

```
'Hello, Jon Snow!'
```

```
#String interpolation
a = 5
b = 10
result = f'Five plus ten is {a + b} and not {2 * (a + b)}.'
print (result)
```

```
Five plus ten is 15 and not 30.
```

formatted_io.ipynb

23

Review : While Loop

```
while expression:
    statement(s)
```

- **break** and **continue** is used to break and to iterate a loop
- **Case 1: counter**

```
count = 0
while (count < 9):
   print ('The count is:', count)
   count = count + 1
print ("Good bye!")
```

- **Case 2: infinite loop**

```
var = 1
while var == 1 : #infinite loop
   num = input("Enter a number :")
   print ("You entered: ", num)
print ("Good bye!")
```

Alternative is **while True:**

Case 3: else in while statement

```
count = 0
while count < 5:
   print (count, " is less than 5")
   count += 1
else:
   print (count, " is not less than 5")
```

The else clause is only executed when your while condition becomes false. If you break out of the loop, or if an exception is raised, it won't be executed.

break breaks the loop and execution is continued with the first statement after the while loop

24

Review : for

- They are used for iterating over a sequence
- It works as an iterator method, rather than a loop structure
- With the for loop we can execute a set of statements, once for each item in a list, tuple, set, dictionary etc.

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

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

break breaks the loop and execution is continued with the first statement after the while loop

25

For loops

```
Case 1:
for letter in 'Python':     # First Example
   print ('Current Letter :', letter)

Case 2:
for x in range(0, 3):
    print ("We're on time %d" % (x))   ← Old style formatting!

for num in range(10,20):  #to iterate between 10 to 20
   for i in range(2,num): #to iterate on the factors of the number
      if num%i == 0:      #to determine the first factor
         j=num/i          #to calculate the second factor
         print ('%d equals %d * %d' % (num,i,j))
         break #to move to the next number, the #first FOR
   else:                  # else part of the loop
      print (num, 'is a prime number')
```

26

For loop behaves as an iterating schema

```
#BAD implementation
for i in range(len(colors)):
    print(colors[i])

#PREFERRABLE
for color in colors:
    print(color)
```

lists.ipynb

```
#create a list
colors = ["red","green","blue","yellow","black","white"]
#Bad implementation
for i in range(len(colors)):
    print (colors[i],end="/")

#Python way
print("\n")
for cc in colors:
    print (cc,end="/")
```

```
red/green/blue/yellow/black/white/

red/green/blue/yellow/black/white/
```

27

Range

- Range generates a sequence of numbers as a "list"
- Format : range(start, stop, step size)
- Frequently used in **for** loops

```
In [87]: a=range(-100,30,27)

In [88]: list (a)
Out[88]: [-100, -73, -46, -19, 8]
```

28

Lists

- There are several key methods on lists which manipulates the list itself or retrieve part of a list.
  - Adding/appending a new element to the list
  - Deleting an existing element from the list
  - Searching through a list
  - Slicing a list

29

Lists – Common operations

lists.ipynb

```
# create an empty list (two ways)
empty_list = []
empty_list = list()

# create a list
friends = ['ali', 'veli', 'ayse']

# examine a list
friends[0]     # print element 0 ('ali')
len(friends)   # returns the length (3)

# modify a list (does not return the list)
friends.append('lisa')                  # append element to end
friends.extend(['smith', 'john'])   #append multiple elements to end
friends.insert(0, 'can')   # insert elem at index 0 (shifts everything right)
friends.remove('veli')     # search for first instance and remove it
friends.pop(0)             # remove element 0 and return it
del friends[0]             # remove element 0 (does not return it)
friends[0] = 'mehmet'      # replace element 0
```

30

Lists – Common operations

```
# find elements in a list
friends.count('ali')      # counts the number of instances
allfriends.index('ned')   # returns index of first instance

# list slicing [start:end:step]
weekdays = ['mon', 'tues', 'wed', 'thurs', 'fri']
weekdays[0]          # element 0
weekdays[0:3]        # elements 0, 1, 2
weekdays[:3]         # elements 0, 1, 2
weekdays[3:]         # elements 3, 4
weekdays[-1]         # last element (element 4)
weekdays[::2]        # every 2nd element (0, 2, 4)
weekdays[::-1]       # backwards (4, 3, 2, 1, 0)

# alternative method for returning the list backwards
list(reversed(weekdays))

# sort a list in place (modifies but does not return the list)
allfriends.sort()
allfriends.sort(reverse=True)     # sort in reverse
allfriends.sort(key=len)          # sort by a key
```

lists.ipynb

31

Lists – Common operations

```
# insert into an already sorted list, and keep it sorted
num = [10, 20, 40, 50]
from bisect import insort   #binary search algorithm
insort(num, 30)

# create a second reference to the same list
same_num = num
same_num[0] = 0          # modifies both 'num' and 'same_num'

# copy a list (two ways)
new_num = num[:]
new_num = list(num)

# examine objects
num is same_num   #returns True (checks if they are the same object
num is new_num    # returns False
num == same_num   # returns True (checks if they've same contents)
num == new_num    # returns True

# split a string into a list of substrings separated by a delimiter
s='I like you'
s.split(' ')         # returns ['I', 'like', 'you']
s.split()            # equivalent (since space is the default
delimiter)
s2 = 'a, an, the'
s2.split(',')        # returns ['a', ' an', ' the']

# join a list of strings into one string using a delimiter
fav = ['game', 'of', 'thrones']
' '.join(fav) # returns game of thrones'
```

lists.ipynb

32

+ and * operations on lists

- See below:

```
In [89]: a=[1,2,3,4]

In [90]: b=[5,6,7,8]

In [91]: c=a+b

In [92]: d=a*2+b

In [93]: c
Out[93]: [1, 2, 3, 4, 5, 6, 7, 8]

In [94]: d
Out[94]: [1, 2, 3, 4, 1, 2, 3, 4, 5, 6, 7, 8]
```

lists.ipynb

33

Built-in list functions

- cmp(list1, list2) Compares elements of both lists.
- len(list) Gives the total length of the list.
- max(list) Returns item from the list with max value.
- min(list) Returns item from the list with min value.
- list(a) Converts a given variable (tuple, string etc) to a list

34

Simple iteration over lists

```
#simple iteration over lists
# Program to find the sum of all numbers stored in a list
# List of numbers
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]
tot = 0
# iterate over the list
for val in numbers:
    tot = tot+val
print("The sum is", tot)
print("Or, sum(numbers) gives", sum(numbers))
```

```
The sum is 48
Or, sum(numbers) gives 48
```

35

More on sequence types

- Lists in lists

```
In [56]: lst1=[2,4,6,-1,True,'a']

In [57]: lst2=['aa','bb',40]

In [58]: lst3=[lst1,lst2]

In [59]: lst3
Out[59]: [[2, 4, 6, -1, True, 'a'], ['aa', 'bb', 40]]

In [60]: lst3[0]
Out[60]: [2, 4, 6, -1, True, 'a']

In [61]: lst3[0][2]
Out[61]: 6

In [62]: lst3[1][0]
Out[62]: 'aa'
```

lst3 has two elements and each of them is another list

MATRIX!!!!

Reaching individual elements
In each of the sub-lists

36

Enumeration over lists

- **enumerate** is a built-in function in Python. It allows us to loop over something and have an automatic counter. Here is an example:

```
In [69]: a=['red','green','blue','maroon','orange']

In [70]: b=enumerate(a)

In [71]: list(b)
Out[71]: [(0, 'red'), (1, 'green'), (2, 'blue'), (3, 'maroon'), (4, 'orange')]
```

```
#enumeration over lists
my_list = ['apple', 'banana', 'grapes', 'pear']
for c, value in enumerate(my_list, 1):
            print(c, value)
```

```
1 apple
2 banana
3 grapes
4 pear
```

37

List comprehensions

- List comprehensions provide a **short and concise** way to **create lists**.
- It consists of square brackets containing **an expression** followed by a **for clause**, then zero or more **for or if clauses**.
- The expressions can be anything, meaning you can put in all kinds of objects in lists.
- The result would be a **new list** made after the evaluation of the expression in context of the if and for clauses.

```
#list comprehensions
multiples = [i for i in range(30) if i % 3 == 0]
print(multiples)
```

```
[0, 3, 6, 9, 12, 15, 18, 21, 24, 27]
```

IMPORTANT!!!!!

38

List comprehensions

```
squared = []
for x in range(10):
        squared.append(x**2)
```

```
#list comprehensions
squared = [x**2 for x in range(10)]
print(squared)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

More examples:

```
#list comprehensions
num_list = [y for y in range(100) if y % 2 == 0 if y % 5 == 0]
print(num_list)
```

```
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
```

What is the output for each of the following list comprehensions??

```
obj = ["Even" if i%2==0 else "Odd" for i in range(10)]
print(obj)

matrix = [[1, 2], [3,4], [5,6], [7,8]]
t = [[row[i] for row in matrix] for i in range(2)]
print (t)
```

39

Class study (Quiz):

- Define two lists. The first list (list1) must contain the integer values 1, 2 and 3 and the second list (list2) must contain the string values a, b and c.

a) Iterate through both lists to create another list that contains all the combinations of the (list1) and (list2) elements. The final list should **"exactly"** look like:
[1a, 1b, 1c, 2a, 2b, 2c, 3a, 3b, 3c] or [a1, a2, a3, b1, b2, b3, c1, c2, c3]

b) Make the final list contains all possible combinations :
[1a, a1, 1b, b1, 1c, c1, 2a, a2, 2b, b2, 2c, c2, 3a, a3, 3b, b3, 3c, c3]

- Use only the constructions we have seen so far
- Use formatted output techniques to create the answer
- Use core python. You are not allowed to use any python libraries
- Output new lists to terminal (console) using print ( )
- Submit your source code/ipynb notebook via LMS
- Commenting and writing a meaningful code will be matter!!!!

40

Other ways to deal with inputs and outputs

- Console (input and print)
- Files
- Databases
- Online (similar to files)

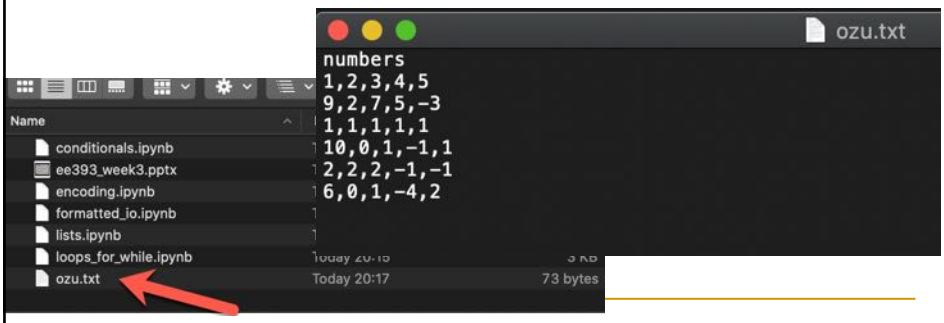
file_io.ipynb

41

Files: Places to put strings and other stuff

- Files are these named large collections of bytes.
- Files typically have a *base* name and a *suffix*
  - **ozu.txt** has a base name of "**ozu**" and a suffix of "**.txt**"
- Files exist in *directories* (sometimes called *folders*)

In Python, we can read from a file, write to a file or create a new file for writing



42

In Python:

- We can read inputs from a file
- We can write outputs to a file

- Use of a file in Python is done by following the sequence given below :
  - Open the file using "**open**" function
  - If needed, read from file using **read()** method
  - If needed, write to file using **write()** method
- Note that, everything we read from a file and everything we write to a file is "string". It is programmer's responsibility to convert it into numbers.
- String format method can be used to control what's written to a file

43

Working with files

Sample data: ozu.txt

file_io.ipynb

Opening a file:

```
f = open("ozu.txt")          # equivalent to 'r' or 'rt'
f = open("ozu.txt",'w')      # write in text mode
f = open("ozu.png",'r+b') # read and write in binary mode
f = open("C:/Python3.7/README.txt")  # specifying full path
#specify more parameters
f = open("ozu.txt", mode = 'r', encoding = 'utf-8')
```

You can specify full path for file access

44

Modes:

- 'r' : Open a file for reading. (default)
- 'w' : Open a file for writing. Creates a new file if it does not exist or truncates the file if it exists.
- 'x' : Open a file for exclusive creation. If the file already exists, the operation fails.
- 'a' : Open for appending at the end of the file without truncating it. Creates a new file if it does not exist.
- 't' : Open in text mode. (default)
- 'b' : Open in binary mode.
- '+' : Open a file for updating (read & write)

45

How to write to a file:

```
with open("test.txt",'w',encoding = 'utf-8') as f:
        f.write("Hello world!!\n")
        f.write("This file\n\n")
        f.write("contains three lines\n")
```

How to read from a file:

```
f = open("test.txt", 'r', encoding = 'utf-8')
f.read(4)    # read the first 4 UTF8 data => 'Hell'
f.read(4)    # read the next 4 data => 'o wo'
f.read()     # read the rest => 'rld!!\nThis file\ncontains three lines\n'
f.read()     # further reading returns empty sting => ''
```

46

file_io.ipynb

```
#writing to a file
f = open("testOZU.txt",'w',encoding = 'utf-8')
f.write("Hello world!!\n")
f.write("This file\n\n")
f.write("contains several lines\n")
f.write("even some utf-8 chars Ğ İ € \n")
f.close()
```

testOZU.txt

```
Hello world!!
This file

contains several lines
even some utf-8 chars Ğ İ €
```

47

```
#reading from testOZU.txt
f = open("testOZU.txt", 'r', encoding = 'utf-8')
x1 = f.read(4)     # read the first 4 UTF8 data => 'Hell'
x2 = f.read(4)     # read the next 4 data => 'o wo'
x3 = f.read()       # read the rest => 'rld!!\nThis file\n\ncontains ...'
x4 = f.read()      # further reading returns empty sting => ''
f.close()
print ("first read: ", x1)
print ("second read: ", x2)
print ("third read: ", x3)
print ("last read: ", x4)
```

file_io.ipynb

```
first read:  Hell
second read:  o wo
third read:  rld!!
This file

contains several lines
even some utf-8 chars Ğ İ €

last read:
```

testOZU.txt

```
Hello world!!
This file

contains several lines
even some utf-8 chars Ğ İ €
```

48

We can read a file line by line until it ends using for loop:

```
#We can read a file line by line
#until it ends using for loop:
f = open("testOZU.txt", 'r', encoding = 'utf-8')
for line in f:
    print ("Line is read =>", line)
close(f)
```

```
Line is read => Hello world!!

Line is read => This file

Line is read =>

Line is read => contains several lines

Line is read => even some utf-8 chars Ğ İ €
```

49

We can read a file line by line using readline()

```
#readline() reads a whole line at a time:
f = open("testOZU.txt", 'r', encoding = 'utf-8')
x = f.readline()
f.readline()
f.readline()
y = f.readline()
f.readline()
print (x); print (y)
f.close()
```

```
Hello world!!

contains several lines
```

50

Or, readlines() read everything and returns a list

```
#readlines() reads all the lines:
f = open("testOZU.txt", 'r', encoding = 'utf-8')
x = f.readlines()
print (x)
print (x[0])
print (x[1])
print (x[3])
```

```
['Hello world!!\n', 'This file\n', '\n', 'contains several lines\n', 'even some utf-8 chars Ğ İ €
Hello world!!

This file

contains several lines
```

51

IMPORTANT FILE METHODS (optional – see .ipynb file

- **read(n)** Read atmost n characters form the file. Reads till end of file if it is negative or None.
- **readable()** Returns True if the file stream can be read from.
- **readline(n=-1)** Read and return one line from the file. Reads in at most n bytes if specified.
- **readlines(n=-1)** Read and return a list of lines from the file. Reads in at most n bytes/characters if specified.
- **seek(offset,from=SEEK_SET)** Change the file position to offset bytes, in reference to from (start, current, end).
- **seekable()** True if the file stream supports random access.
- **tell()** Returns the current file location.
- **writable()** Returns True if the file stream can be written to.
- **write(s)** Write string s to the file and return the number of characters written.
- **writelines(lines)** Write a list of lines to the file.

52

Tasks for file i/o code –extra example

```
# Open a file for reading - ozu.txt is in the same folder as .pynb file
ff = open("ozu.txt", "r")
x =ff.readline()
print ("First line of the file :", x)
for line in ff:
    print("Read from file as a string =>", line.split(","))
    aa = [int(a) for a in line.split(",")]
    print ("converted to numbers, and a list is created =>", aa)
    print ("Sum of numbers in the list =>", sum(aa))
```

file_io.ipynb

```
First line of the file : numbers

Read from file as a string => ['1', '2', '3', '4', '5\n']
converted to numbers, and a list is created => [1, 2, 3, 4, 5]
Sum of numbers in the list => 15
Read from file as a string => ['9', '2', '7', '5', '-3\n']
converted to numbers, and a list is created => [9, 2, 7, 5, -3]
Sum of numbers in the list => 20
Read from file as a string => ['1', '1', '1', '1', '1\n']
converted to numbers, and a list is created => [1, 1, 1, 1, 1]
Sum of numbers in the list => 5
Read from file as a string => ['10', '0', '1', '-1', '1\n']
converted to numbers, and a list is created => [10, 0, 1, -1, 1]
Sum of numbers in the list => 11
Read from file as a string => ['2', '2', '2', '-1', '-1\n']
converted to numbers, and a list is created => [2, 2, 2, -1, -1]
Sum of numbers in the list => 4
Read from file as a string => ['6', '0', '1', '-4', '2']
converted to numbers, and a list is created => [6, 0, 1, -4, 2]
Sum of numbers in the list => 5
```

53

Introduction to Functions

- Function: group of statements within a program that perform as specific task
  - Usually one task of a large program
    - Functions can be executed in order to perform overall program task
  - Known as *divide and conquer* approach
- Modularized program: program wherein each task within the program is in its own function

54

Benefits of Modularizing a Program with Functions

- The benefits of using functions include:
  - Simpler code
  - Code reuse
    - write the code once and call it multiple times
  - Better testing and debugging
    - Can test and debug each function individually
  - Faster development
  - Easier facilitation of teamwork
    - Different team members can write different functions

55

Functions in Python

- A function is a set of statements that take inputs, do some specific computation and produces output.
- The idea is to put some commonly or repeatedly done task together and make a function, so that instead of writing the same code again and again for different inputs, we can call the function.
- Python provides built-in functions like print(), etc. but we can also create your own functions. These functions are called user-defined functions.

```
1  # A simple Python function to check
2  # whether x is even or odd
3  def evenOdd( x ):
4      if (x % 2 == 0):
5          print ("even")
6      else:
7          print ("odd")
8
9  # Driver code
10 evenOdd(2)
11 evenOdd(3)
12
```

FUNCTION

```
even
odd
```

Name of the function: **evenOdd**
Value passed to function : x

56

void Functions and Value-Returning Functions

- A void function:
  - Simply executes the statements it contains and then terminates. For ex. print( )
- A value-returning function:
  - Executes the statements it contains, and then it returns a value back to the statement that called it.
    - The `input`, `int`, and `float` functions are examples of value-returning functions.

Now, we will learn how to define our very own functions

57

void Python functions

- Defining a function:

```
def greet_user():
    print("Hello!")
```

- Passing information to a function:

```
def greet_user(username):
    print("Hello, " + username + "!")
```

invoke the function using => **greet_user('Jon Snow')**

- Multiple information is possible:

```
def f1(a,b,c):
    print(a,b,c)
```

invoke the function using: **f1(3,6,9)**

58

void function examples

functions.ipynb

```
#void function
def sayHello():
    print ("Hello")

def sayHello2(name):
    print ("Hello ", name)

def sayHello_n_times(n):
    for i in range(n):
        print ("Hello",end=" ")

def giveMyMsg(name,msg):
    print ("Hello ", name)
    print ("Here is your message: ", msg)
```

```
sayHello()
sayHello2("Özyeğin Univ.")
sayHello_n_times(4)
giveMyMsg("Jon Snow", "Winter is coming!")
```

```
Hello
Hello  Özyeğin Univ.
Hello Hello Hello Hello Hello  Jon Snow
Here is your message:  Winter is coming!
```

59

Defining Functions

Function definition begins with “def.”

Function name and its arguments.

```
def get_final_answer(filename):
    """Documentation String"""
    line1
    line2
    return total_counter
```

Colon.

The indentation matters…
First line with less indentation is considered to be outside of the function definition.

The keyword ‘return’ indicates the value to be sent back to the caller.
A function may not contain return if it isn’t sending anything back

60

Functions returning a value

```
#finds root of ax+b=0
def findRoot(a,b):
    x = -b/a
    return x
```

```
c1, c2 = 5, 12
root = findRoot (c1, c2)
print ("Root is ", root)
```

```
Root is  -2.4
```

functions.ipynb

61

Different uses

```
#finds root of ax+b=0
#default parameters
def findRoot(a=1,b=-1):
    import math
    if (a==0):
        return math.nan
    x = -b/a
    return x
```

```
print (findRoot(b=-4, a=7)) #7x-4=0
```

```
0.5714285714285714
```

```
aa=10
print (findRoot(aa,5)) #10x-5=0
```

```
-0.5
```

```
print (findRoot(0,1)) #0x+5=0
```

```
nan
```

```
#finds root of ax+b=0
#if second parameter is not supplied, it is -1
def findRoot(a,b=-1):
    x = -b/a
    return x
```

```
print (findRoot(8)) #8x-1=0
```

```
0.125
```

functions.ipynb

62

- (name=value) pair in functions is possible:

def roots(…… a,b,c):

Then the following uses are OK

roots(a=1,b=2,c=3)

roots(c=4,a=2,b=10)

You may use default values for the information passed to a function. Default values are used if there is no value passed:

def roots(a=1,b,c=0):

……

Then,

roots(1,2,3) **roots(a=11,c=2,b=3)**

roots(b=-2) #now, a=1 and c=0 by default!!

roots(a=7,b=0) #now, c=0 by default!!

63

- A function may return many values. For ex:

```
def sumAll(a,b,c):
    d=a+b+c
    return d,a,b
```

```
mySum=sumAll(1,2,3)
d=sumAll(1,2,3)
print (d)
```

(TUPLE)

```
(6, 1, 2)
```

(Lists)

```
def combineLists(l1,l2):
    return l1+l2
```

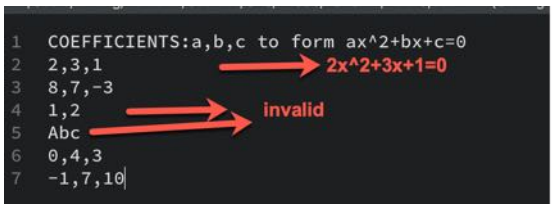
```
a=[1,2,3,"hello"]
b=[2+5j,3*"OzU",a]
c=combineLists(a,b)
print (c)
```

```
[1, 2, 3, 'hello', (2+5j), 'OzUOzUOzU', [1, 2, 3, 'hello']]
```

64

Homework

- The file "**coeff.txt**" contains coefficients for a series of 2nd order equations. Below is a sample data:



- Develop a Python program which reads all the coefficient inputs from coeff.txt and find the "real" roots of the equations.
  - If there is no real root, then it must print "No real root"
  - If the coefficients inputs are inappropriate, then it must print "equation is not valid"
- Roots are to be found using a function
- All the outputs are to be written to an output text file

65

Libraries

PREVIOUSLY:

To use a library, you can use a simple **import** statement, which should come at the beginning of your program. For an example, let's look at a standard Python library, **math**. To access the functions within this module start with:

import math

Then, we can use methods included within the module such as sqrt as **math.sqrt(5.8)**

Alternatively, **from math import sqrt**

Then, we can use methods included within the module such as sqrt as **sqrt(5.8)**

Use with caution: **from math import ***

66

Defining an object

Libraries contain objects that we can call after importing the relevant library

Objects know things.

- Data that is internal to the object.
- We often call those instance variables.

Objects can do things.

- Behavior that is internal to the object.
- We call functions that are specific to an object methods.
  - But you knew that one already.

We access both of these using **dot notation**

- object.variable
- object.method()

```
math.pi
math.sqrt()
```

67

Making your own libraries

- Importing a code that you have written as a library works the same as in the examples above, provided that you have saved your code as a **.py** file.
- For example, if you will be using a function or set of functions in multiple programs, you can save them in a separate file as **myfunctions.py**. Be sure to include in your module whatever import command are necessary for it to work. In another program, you can import it using:

```
import myfunctions
```

68

SEE YOU NEXT WEEK!!!

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

69