بِسۡمِ ٱللَّهِ ٱلرَّحۡمَٰنِ ٱلرَّحِيمِ

Muhammad Haris

Road To Machine Learning ان شاء اللہ

"اللہ کے فضل سے ناممکن بھی ممکن ہو جاتا ہے"

بسم اللہ

20th-July-2025

"Python for Everybody"

Dr Charles Severance Ph.D.

(Course 1 - Lecture 1)

END

⇒ Functions → def never executes the code automatically we've to invoke it

- We should define function initially by def to use it throughout our code.
- We also put values in function parameter (...) to invoke values of our need

e.g:
```
def Islamabad (place):
    if place == 'Monal'
        print("good place")
    else:
        print ("simple")

place (fb)
print ("Hello")
```
→

Output:
Simple
Hello

- If we print a function f() we must return value inside function to show in the output section.
  It also ends function execution and "sends back" the result of function.
- Built in function returns automatically
- Only the function we make required to return inside def f():

- Only use variable in function parameter
  e.g (abc, ab2c, ab_cd) and not use numbers
  e.g (1abc, @bc) and keywords and quoted name
  e.g ("abc")

- Multiple parameters / Arguments allowed

```
def f(a,b): → (parameter → name of function
    {Add = a+b
    return add
x = f(3,5) → Argument (actual value)
Print(x)
output ⇒ 8
```

⇒ Loops & Iteration :

→ While loops are Indefinate
- While loops runs until condition will false
- Break → quits the loop
- Continue → goes to the next iteration and skips the current one
- True False are keywords
  - ↓ In while we break it
  - ↓ Dont use in while

→ for loops are definate.

→ Do it automatically in sequence.

```
for i in [0,1,2,3]:
```

variable ⇔ it prints every index value one time throughout in sequence

→ Large Number

↓

```
LN = None
for N in [1,3,7,6]
    if N > LN or LN is None :
        LN = N
Print (LN)
```

→ Similarly ~~for small number~~

→ Count = 0

Count += 1

→ Sum = 0

Sum += i

→ Average → Sum / Count

```
Count = 0
sum = 0
for i in [0,1,2]:
    count += 1
    sum += i
Average = sum/count
```

⇒ Files:

→ a = open (filename, mode)

↓

open ('haris.txt', 'r')

→ filename is a string

→ mode is optional (r → read, w → write)

→ \n → newline

→ open() are line iterables by default.

→ continue skips the current iteration.

→ break quits the loop

→ quit() stops the whole code to proceed

DATA STRUCTURE

⇒ Lists

→ can be empty

→ Also add list '+' (concatenate)

→ Methods also apply

→ Mutable (changeable) | str ≠ mutable

→ can convert strings into list by split() method.

→ By default split also divide str to lists by white spaces

→ a = [] → constant to create

⇒ Dictionaries:

→ By using "key" we insert and retrieve the value
→ Mutable like lists but use key to change value or replace instead of index [] in strings
→ a = {} → constant to create
→ dict.get(key, value) has default value 0
(name, 0)
→ Only values are mutabe not keys
→ for i in dict:
print(i) → it prints only key of dict not values like {a:1} → print only a
→ In order to print both key and value → print(i, dict[i])
(i, dict[i])
→ Can get a list of keys, values or items (both) from dict → print(list(dict.items())) and it return in tuples inside list

→ In a loop we can also use two iteration variable → for i, j in dict.items():
i for key, j for value.

⇒ Tuples:

→ Immutable, not sort, append and reverse
→ only ['count', 'index'] possible as method
→ more efficient and used if need fixed values
→ (x, y) = (4, 'haris') also possible}

print(x) = 4

→ items() method in dictionaries returns the list of (key, value) tuples
→ Tuple are comparable (1,2) > (2,1) False and only check first value and skips others. if first value not differ then go on second and so on
→ Cannot directly sort tuples but d.items() with sorted (d.items()) it sort dictionary items by key that are return in list form of tuples.
→ If we want to sort in by values intead of keys we have to create empty list and change values to key position of dictionary with append

```
a = {'haris' : 1 , 'fairy' : 3 }
b = [ ]
for i, j in a. items ():
        b . append (j ,i)
b = sorted (b, reverse = True) → if want high to low
```

also for value sorting → key = lambda x: x[1]

Regular Expressions :

→ import re → import this when use
→ Multiple method like re.search() for return a matches we like to find
→ re.findall() if we want to extract the matching strings. → returns lists of all matching string

```
x = 'fairy Meadows'
y = re.findall('[a-k]+', x)
```

(+ → one or more)

print(y) → ['fai', 'end'] → from each string in x only matching alphabets from a-k it extracts and print and make a list of that matching strings. and if no matching strings found it return empty list []

→ `^f.+:`

- `+` → one or more characters
- `^f` → first character in matching is f
- `:` → last character

`+` → one or more
`*` → zero or more.

This regex is known as "Greedy Matching" and it always shows the largest possible string and if shortest avaible too it doesnot care for it.

→ .+? → Non Greedy Matching

→ .* is like if x = 'fgl' then first and last only avaible and we can't use .+ because it need one or more inside first and last → 'fxy' then .+ → (f,y) fy. and we use group() in print for re search method for exact output

→ re.findall() like for extracting email only \S+@\S+ → atleast one non whitespace character, so this method is for that extracting

→ further if we want to extrat the portion after @ then y = re.findall('@([^]*)', x)

↓ this parenthesis preclude @ in the output.

All but Not space ← [^] → matches non blank character

* → matches more ←

→

Networked Programs

→ All ports have speitic unique numbers like for Http the post number is 80

Steps:

→ import socket → (1)

(2) mysock = socket.socket(socket.AF_INET, socket.SOCK.STREAM)

(3) mysock.connect(('data.pr4e.org', 80))

'data.pr4e.org' ← Host

80 ← post

→ Making an HTTP request (4)

cmd="GET http://www.dr-chuck.com/page1.htm HTTP/1.0"

(5) mysock.send(cmd.encode())

While True:

data = mysock.recv(512)

if (len(data) < 1):

break

print(data.decode())

\r\n a newline

\r\n\r\n End of line HTTP request headers.

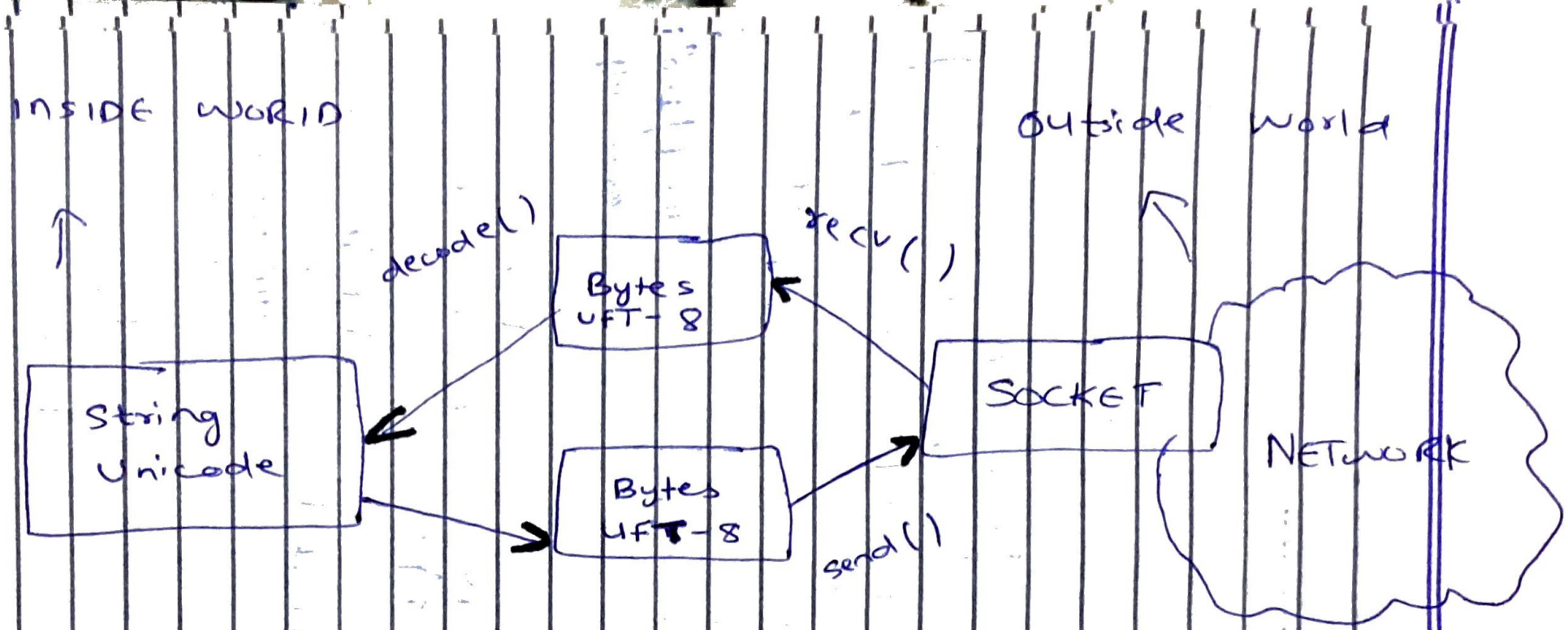
→ Unicode Characters and Strings:

→ Ascii, UFT-32, UFT-16, UFT-8

1 byte, 4 byte, 2 byte, 1-4 byte

↓

Best and ideal



→ Python don't understand Human readable that's why string convert to bytes.

→ urllib → is a library that does all socket work and make web pages looks like a file

```
import urllib.request, urllib.parse, urllib.error
```

works as open(file)

```
fhand = urllib.request.urlopen('http://data.pr4e.org/romeo.txt')
for line in fhand:
    print(line.decode().strip())
```

→ Beautiful Soup

```
import urllib.request, urllib.parse, urllib.error
from bs4 import BeautifulSoup

url = input('Enter - ')
html = urllib.request.urlopen(url).read()
soup = BeautifulSoup(html, 'html.parser')

# Retrieve all of the anchor tags
tags = soup('a')
for tag in tags:
    print(tag.get('href', None))
```

→ Syntax for ignoring SSL errors because some websites have SSL certificates (http://) that python refuses to trust

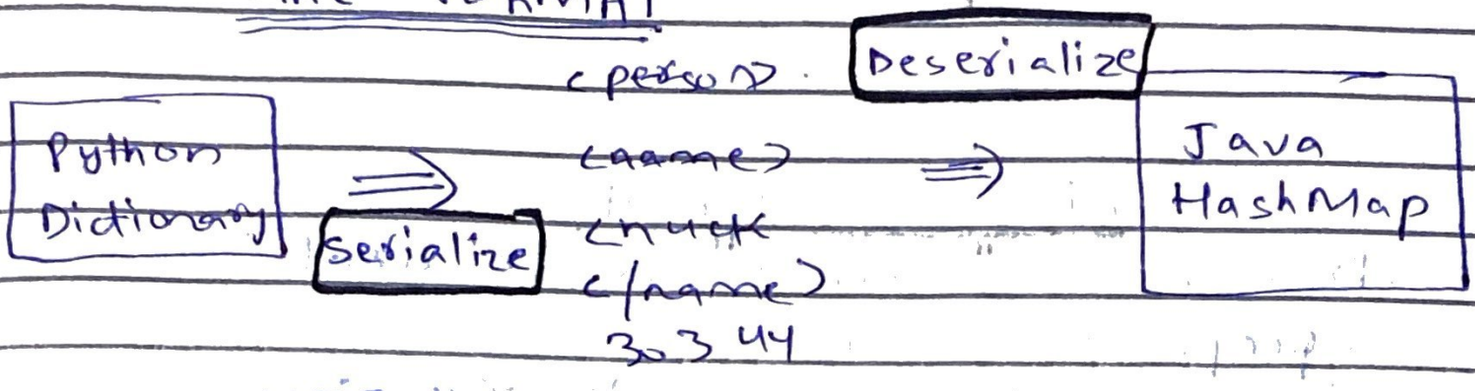
```
# ctx = ssl.create_default_context()
ctx.check_hostname = false
ctx.verify_mode = ssl.CERT_NONE
```

→ Different tags to filter result like

class → tag.get('class') → returns list
Any link → tag.get('href') → string
text → tag.get.find_all() → string
id → tag.get('id) → string.

USING WEB SERVICES

WIRE FORMAT



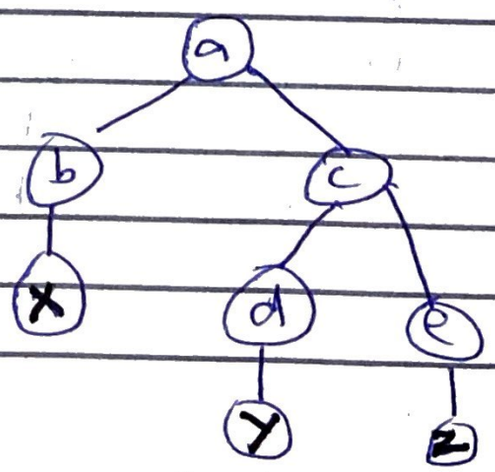
→ Two types of serialization format we'll learn are 'XML' and 'JSON'

XML → start tag, End tag, Text content, Attribute and x) Self closing tag.

'tags' → indicates beginning and Ending of element
'Attributes' → key/value pairs on opening tag of XML
'Serialize/De-serialize' → Convert data in common format that can stored or transmit between system in a programming language independant manner.

XML as a Tree

```
<a>                    Element  Text
  <b> X </b>
  <c>
    <d> Y </d>
    <e> Z </e>
  </c>
</a>
```



XML as Paths:

like in previous diagram XML path is /a/b X

/a/c/d Y

/a/c/e Z

XML SCHEMA

Describe a "contract" as to what is acceptable XML

→ for validation of specific XML piece

XML Schema Languages

- Document Type Defination (DTD) (ISO 8879:1986 SGML)
- XML Schema from W3C (XSD)

→ Defines structure

→ Explicit and strict by design

→ file extention is .XSD

→ By default minOccurs and maxOccurs is 1.

→ xs: sequence, xs: all, xs: choice

→ Complex elements contains other elements or has attributes, it is not plain text and has structure inside

Parsing XML

Library → import xml.etree.ElementTree as ET

''' Triple-quoted string is a potentially multi-line string

```
data = ''' <person>
  <name> Chuck </name>
  <phone type = "intl">
     +1 734 303 4456
  </phone>
  <email hide = "yes" />
</person>
```

→ data is a multiline string.

```
tree = ET.fromstring(data)
print('Name:', tree.find('name').text)
print('Attr:', tree.find('email').get('hide'))
```

JSON

JavaScript Object Notation

→ Json represents data as nested 'lists' and 'dictionaries'

```
import json
data = '''{
    "name" : "Chuck",
    "phone" : {
        "type" : "intl",
        "number" : +1 734 303 4456
    },
    "email" : {
        "hide" : "yes"
    }
}'''

info = json.loads(data)
print('Name:', info["name"])
print('Hide:', info["email"]["hide"])
```

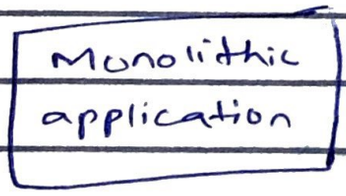
Service Oriented Architecture / Approach

→ Independent software Components (called services) communicates over a network to perform tasks

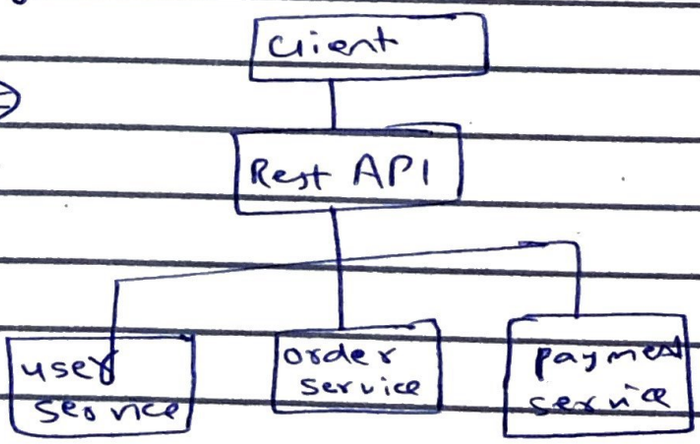
Each service can
- Perfoms specific business function (like "create user" or "process payment")
- Is loosly coupled
- can be reused, scaled or replaced independently

In practical u can say that for building app by team of independent experts in their respective service.

Monolithic (old way)



SOA



Example of pizza shop where independent services combine to run pizza shop

USING API (Application Program Interface)

→ SOA allows an application to be broken into parts and distributed across a network.

→ An API is a contract for interaction

→ Web Service provides infrastructure for application cooperating (an API) over a network. "SOAP" and "REST" are two style of web services.

→ XML and JSON are Serialization formats