In [ ]:
```
import json
python_data <-->json_data

import sqlite3
conn = sqlite3.connect('dbFile')
sth = conn.cursor()
|
sth.execute('QUERY')
|
QUERY -> 'select statement' --> fetchone() fetchall() fetchmany(n=3) (or) genera
conn.close()

Flask
htmlForm (or) web UI - User Inputs - login.html --Submit ---- {'Key':Value}
============================>=============================    _____(p1.py)_____  -

======================<===================================   _______(p1.py)______
                    display.html
```

In [ ]:
```
record1 = (101,'pA',1000)
record1[0] # 101
record1[1] # pA
record1[2] # 1000
----------------------------//index based access - Access by index - recordbased

record2 = {'pid':101,'pname':'pA','pcost':1000} <== dict  // Access by Column
record2['pid']  Vs record1[0]
sqlite3.Row  ----> Sqlite3 to return rows - access values by column name instead
| -> {{variable["Key"]}} <== template
```

In [ ]:
```
import requests
requests.get('http://127.0.0.1:5000/dataview').headers
```

In [ ]:
```
print(requests.get('http://127.0.0.1:5000/dataview').text)
```

In [ ]:
```
requests.get('http://127.0.0.1:5000/pageview').headers
```

In [ ]:
```
print(requests.get('http://127.0.0.1:5000/pageview').text)
```

In [ ]:
```
REST API
REST API - Representational State Transfer API
[node1] <------------------> [node2]
          HTTP methods
           - GET POST PUT DELETE
Key Points about REST APIs
----------------------------
1. Client-Server
   |
   Client - ask for resource
   Server - provide the resource

2. Stateless architecture
    |-> Each request in independent - server doesnot remember previous request (
    |-> Every HTTP request from client -->server doest not keep track of previou
          (there is no session state)
```

```
3. Uniforminterface
         |-> /example/101 <== ID 101
         |-> /example/102 <== ID 102
4. Resource based - every entity is a resource - identified by URL
|
5. HTTP Methods
   GET - Read/Get the data   --> GET/products - Get list of products
   POST - create data        --> POST/products/105 - Add new product
   PUT - Update data         --> PUT/product/105 -- modify product 105 ID detail
   DELETE - remove data      --->DELETE/prodcut/105 -- delete this ID 105

JSON
```

In [ ]:
```python
products = [{'pid':101,'pname':'pA','pcost':1000},
            {'pid':102,'pname':'pB','pcost':2000},
            {'pid':103,'pname':'pC','pcost':3000}
           ] # dataset (or) database

app = Flask(__name__)
@app.route("/products",methods=['GET'])
def get_products():
    return jsonify(products)

@app.route("/products",methods=["POST"])
def add_new_product():
```

In [ ]:
```python
import requests
requests.get('http://localhost:5000/dataview')
```

In [ ]:
```python
r = requests.get('http://localhost:5000/dataview')
print(type(r.json())) # Convert to python using requests module Vs json.loads()
```

In [ ]:
```python
products = [{'pid':101,'pname':'pA','pcost':1000},
            {'pid':102,'pname':'pB','pcost':2000},
            {'pid':103,'pname':'pC','pcost':3000}
           ] # dataset (or) database

app = Flask(__name__)
@app.route("/products",methods=['GET'])
def get_products():
    return jsonify(products)

@app.route("/products",methods=["POST"])
def add_new_product():
    new_product={'pid':104,'pname':'pD','pcost':4000}
    products.append(new_product)
    return jsonify(new_product),201
```

In [ ]:
```python
url = 'http://localhost:5000/products'
new_product={'pid':104,'pname':'pD','pcost':4000}

response = requests.post(url,json = new_product)
print(response.json())
```

url = 'http://localhost:5000/products' updated_data = {'pcost':1459.42} response = requests.put(f"{url}/101",json = updated_data) print(response.json())

In [1]:
```
import requests
url = 'http://localhost:5000/products'
response = requests.get(url)
print(response.json())
```

```
[{'pcost': 1000, 'pid': 101, 'pname': 'pA'}, {'pcost': 2000, 'pid': 102, 'pname':
'pB'}, {'pcost': 3000, 'pid': 103, 'pname': 'pC'}]
```

In [2]:
```
response = requests.get(f"{url}/101")
print(response.json())
```

```
{'pcost': 1000, 'pid': 101, 'pname': 'pA'}
```

In [3]:
```
response = requests.get(f"{url}/103")
print(response.json())
```

```
{'pcost': 3000, 'pid': 103, 'pname': 'pC'}
```

In [5]:
```
response = requests.get(f"{url}/105")
print(response.json())
```

```
{'error': 'Product ID not found'}
```

In [ ]:
```
response = requests.get(url)
print(response.json())
```

In [ ]:
```
# Get - all the products
response = requests.get(URL)
print(response.status_code)
print(response.json())
```

In [8]:
```
URL = 'http://localhost:5000/products'
new_product = {'pid':104,'pname':'pD','pcost':4000}
response = requests.post(URL,json = new_product)
print(response.status_code)
print(response.json())
```

```
201
{'pcost': 4000, 'pid': 104, 'pname': 'pD'}
```

In [9]:
```
# Get a specific product
response = requests.get(f"{URL}/104")
print(response.json())
```

```
{'pcost': 4000, 'pid': 104, 'pname': 'pD'}
```

In [10]:
```
response = requests.get(URL)
print(response.status_code)
print(response.json())
```

```
200
[{'pcost': 1000, 'pid': 101, 'pname': 'pA'}, {'pcost': 2000, 'pid': 102, 'pname':
'pB'}, {'pcost': 3000, 'pid': 103, 'pname': 'pC'}, {'pcost': 4000, 'pid': 104, 'p
name': 'pD'}]
```

update_data = {'pcost':1234} response = requests.put(f"{URL}/102",json = update_data) print(response.json())

In [ ]:
```
+-----------+                                   +---------------+
| node1     |                                   |               |
|           | <---------SSH---------------->    |   node2       |
+-----------+                                   +---------------+
  IP:10.20.30.40                                IP:10.44.23.31
  hostname: node1                               hostname: node2
```

```
paramiko - python module
 |->supports SSH,SFTP for file upload/download
 |->commands ; device managements ..

pip install paramiko

1. import paramiko module
2. create ssh client
                |->SSHClient - class ->object
                |->object.methods() - host keyparameters

3. connect to remote server
              |->object.connect() - keywords arguments
                         hostname,username,password,port

4. run a command
              |->object.exec_command('os command') ->tuple (stdin,stdout,stderr)

5. close connection
            |->object.close()

=========================================================================
Upload and Download Files - SFTP
On commandline -> ftp <remoteNode>   sftp
                  ftp> put file1.log # upload to remotenode
                  ftp> get emp.csv # download file from remote node to local
                  ftp> mget *.csv ...
                  ftp> close()
|
1. import paramiko module
2. Create transport channel => paramiko.Transport(('remoteServer',<portNumber>))
3. Start SFTP session => paramiko.SFTPClient.from_transport(transport_object) ->s
|
4. upload a file => sftp.put('filename','/remotenode/path/remoteFile')
   download a file => stfp.get('/remotenode/path/remoteFile','download_file.txt'
|
5. sftp.close()
6. transport_object.close()
```

In [ ]:

```
[root@node2 ~]# cat ssh1.py
import paramiko

# Remote machine credentials
hostname = 'remoteNode-hostname'
port = 22
username = 'remoteNode-login'
password = 'remoteNode-password'

# Initialize SSH client
ssh = paramiko.SSHClient()
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())  # Auto-accept unknown

try:
    # Connect to the remote machine
    ssh.connect(hostname, port=port, username=username, password=password)
    print("Connected successfully.")

    # Run a command
    stdin, stdout, stderr = ssh.exec_command('uptime')
    print(stdout.read().decode())

    # Optionally handle errors
    error = stderr.read().decode()
    if error:
        print("Error:", error)

finally:
    ssh.close()
    print("Connection closed.")

#######################################################
import paramiko

# Remote machine credentials
hostname = 'remoteNode-hostname'
port = 22
username = 'remoteNode-login'
password = 'remoteNode-password'

# Initialize SSH client
ssh = paramiko.SSHClient()
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())  # Auto-accept unknown

try:
    # Connect to the remote machine
        ssh.connect(hostname, port=port, username=username, password=password)
        print("Connected successfully.")

    # Run a command
        for var in ["uptime","hostname","ps","ls -l"]:
                print(f"Command {var} execution results")
                stdin, stdout, stderr = ssh.exec_command(var)
                print(stdout.read().decode())

    # Optionally handle errors
        error = stderr.read().decode()
        if error:
                print("Error:", error)

finally:
        ssh.close()
        print("Connection closed.")

================================================================================
import paramiko
# upload and download file with sftp

# create transport channel
transport = paramiko.Transport(('remoteNodeHostname',22))
transport.connect(username='remoteNodeLoginName',password='remoteNodepassword')

# start sftp server
sftp = paramiko.SFTPClient.from_transport(transport)

sftp.put("sysinfo.log","/root/ol7sysinfo.log") # upload a file

sftp.get("/tmp/r1.log","/root/myfiler1.log") # download a file

sftp.close()
```

```
transport.close()
================================================================================

import paramiko
import json
import pprint

# Remote machine credentials
hostname = 'remoteNode-hostname'
port = 22
username = 'remoteNode-login'
password = 'remoteNode-password'
# Initialize SSH client
ssh = paramiko.SSHClient()
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())  # Auto-accept unknown

try:
    # Connect to the remote machine
        ssh.connect(hostname, port=port, username=username, password=password)
        print("Connected successfully.")

        results=[] # empty list
        d={} # empty dict
    # Run a command
        for var in ["uptime","hostname","date","uname -r"]:
                print(f"Command {var} execution results")
                stdin, stdout, stderr = ssh.exec_command(var)
                #print(stdout.read().decode())
                results.append(stdout.read().decode()) # append command results
        d[hostname] = results # dict of list # add results to dictionary
        jd = json.dumps(d,indent=2)  # convert to json
        # Vs
        with open('results.json','w') as wobj:
                json.dump(d,wobj) # writing to json format

        pprint.pprint(jd) # display data - structured view (or) dumper view

finally:
        ssh.close()
        print("Connection closed.")
```

In [ ]:

```
Regular Expression (Regx)
========================
 |-> Search
 |-> Substitute
 +
 |-> Split

import re - module
Search
=======
re.search() -->--  re.search('pattern_string','input_string',re.I) --> <ack-obje
```

In [15]:

```
# file: ab.py
# -------------
def f1(a):
    class cname:
        def __init__(self,a=0):
            self.a=a
        def method1(self):
            return self.a+100
    obj = cname(a)
    return obj
```

In [13]:
```
# import ab
# ab.f1(15) -><cname_object>
f1(15)
```

Out[13]:
```
<__main__.f1.<locals>.cname at 0x1ef9ba417f0>
```

In [16]:
```
myobj = f1(15)
myobj.method1()
```

Out[16]:
```
115
```

In [ ]:
```
Search
=======
re.search() -->--  re.search('pattern_string','input_string',re.I) --> <ack-obje
```

In [17]:
```
import re
re.search('sales','101,raj,sales,pune,1000')
```

Out[17]:
```
<re.Match object; span=(8, 13), match='sales'>
```

In [18]:
```
bool(re.search('sales','101,raj,sales,pune,1000'))
```

Out[18]:
```
True
```

In [19]:
```
re.search('sales','101,raj,prod,pune,1000')
```

In [20]:
```
bool(re.search('sales','101,raj,prod,pune,1000'))
```

Out[20]:
```
False
```

In [22]:
```
s='101,raj,sales,pune,1000'

if(re.search('sales',s)):
    print('Yes - Given pattern sales is exists')
    print(s)
else:
    print('Sorry - Given pattern sales is Not exists')
```

```
Yes - Given pattern sales is exists
101,raj,sales,pune,1000
```

In [23]:
```
'sales' in '101,raj,sales,pune'
```

Out[23]:
```
True
```

In [24]:
```
'SALES' in '101,raj,sales,bglore'
```

Out[24]:
```
False
```

In [25]:
```
bool(re.search('SALES','101,raj,sales,bglore'))
```

Out[25]:
```
False
```

In [26]:
```
bool(re.search('SALES','101,raj,sales,bglore',re.I))
```
Out[26]:
```
True
```

In [27]:
```
help(re.search)
```
```
Help on function search in module re:

search(pattern, string, flags=0)
    Scan through string looking for a match to the pattern, returning
    a Match object, or None if no match was found.
```

In [ ]:
```
grep/findstr - command operation logic
1. open an existing file - read the content line by line ==> FileHandling
2. search a pattern from inputLine                       ==> re.search()
3. print/display - matched pattern lines only            ==> if only
------------------------------------------------------------
```

In [28]:
```
fobj = open('emp.csv','r')
for var in fobj:
    if(re.search('sales',var)):
        print(var.strip())
```
```
101,raj,sales,pune,1000
450,shan,sales,bglore,3401
321,bibu,sales,hyd,5419
652,karthik,sales,bglore,3405
```

In [ ]:
```
# Substitute
#------------
re.sub() => re.sub('oldPattern_string','replaceString','inputString') --> result
                                                                        |->re
                                                                        |->ot

result_str
    |->replaced string if oldpattern is matched with inputString
    |->otherwise print original inputString
```

In [29]:
```
re.sub('sales','prod','101,raj,sales,pune')
```
Out[29]:
```
'101,raj,prod,pune'
```

In [30]:
```
re.sub('sales','prod','101,raj,QA,pune')
```
Out[30]:
```
'101,raj,QA,pune'
```

In [31]:
```
fobj = open('emp.csv','r')
for var in fobj:
    r = re.sub('sales','****',var)
    print(r.strip())
```

```
eid,ename,edept,ecity,ecost
101,raj,****,pune,1000
102,leo,prod,bglore,2301
230,raj,prod,pune,2300
450,shan,****,bglore,3401
542,anu,HR,mumbai,4590
321,bibu,****,hyd,5419
651,ram,hr,bglore,3130
541,leo,admin,chennai,4913
652,karthik,****,bglore,3405
```

In [32]:

```
fobj = open('emp.csv','r')
for var in fobj:
    if(re.search('sales',var)):
        r = re.sub('sales','****',var)
        print(r.strip())
```

```
101,raj,****,pune,1000
450,shan,****,bglore,3401
321,bibu,****,hyd,5419
652,karthik,****,bglore,3405
```

In [36]:

```
print(re.sub('sales','prod','101,raj,sales,pune'))

print(re.sub('sales','prod','sales,101,sales,raj,sales,pune,sales,sales'))
print(re.sub('sales','prod','Sales,101,sales,raj,SALES,pune,sales,sales'))
```

```
101,raj,prod,pune
prod,101,prod,raj,prod,pune,prod,prod
Sales,101,prod,raj,SALES,pune,prod,prod
```

In [37]:

```
help(re.sub)
```

```
Help on function sub in module re:

sub(pattern, repl, string, count=0, flags=0)
    Return the string obtained by replacing the leftmost
    non-overlapping occurrences of the pattern in string by the
    replacement repl.  repl can be either a string or a callable;
    if a string, backslash escapes in it are processed.  If it is
    a callable, it's passed the Match object and must return
    a replacement string to be used.
```

In [41]:

```
print(re.sub('sales','prod','101,raj,sales,pune'))

print(re.sub('sales','prod','sales,101,sales,raj,sales,pune,sales,sales',1))
print(re.sub('sales','prod','Sales,101,sales,raj,SALES,pune,sales,sales',1))
print(re.sub('sales','prod','Sales,101,sales,raj,SALES,pune,sales,sales',1,re.I)
```

```
101,raj,prod,pune
prod,101,sales,raj,sales,pune,sales,sales
Sales,101,prod,raj,SALES,pune,sales,sales
prod,101,sales,raj,SALES,pune,sales,sales
```

```
C:\Users\karth\AppData\Local\Temp\ipykernel_21424\1567956989.py:3: DeprecationWar
ning: 'count' is passed as positional argument
  print(re.sub('sales','prod','sales,101,sales,raj,sales,pune,sales,sales',1))
C:\Users\karth\AppData\Local\Temp\ipykernel_21424\1567956989.py:4: DeprecationWar
ning: 'count' is passed as positional argument
  print(re.sub('sales','prod','Sales,101,sales,raj,SALES,pune,sales,sales',1))
C:\Users\karth\AppData\Local\Temp\ipykernel_21424\1567956989.py:5: DeprecationWar
ning: 'count' is passed as positional argument
  print(re.sub('sales','prod','Sales,101,sales,raj,SALES,pune,sales,sales',1,re.
I))
```

In [ ]:

```
Regx chars
-----------
1. Basic Regular Expression    - BRE -> Single Pattern
                                 =====
                                  ^ $ ^pattern$  .   .*  [] ^[] []$ ^$

2. Extended Regular Expression - ERE -> Multiple pattern
                                 =====
                                | () + {}

1. Basic Regular Expression    - BRE -> Single Pattern
                                 =====
                                  ^ $ ^pattern$  .   .*  [] ^[] []$ ^$

\s <== matching space chars

^ ==> ^pattern ==> matches the pattern line begins with
```

In [43]:

```
re.search('sales','101,raj,sales,bglore') # General search
re.search('^sales','101,raj,sales,bglore') # Specific search
```

In [44]:

```
re.search('^sales','sales asfsadfsafsda')
```

Out[44]:

```
<re.Match object; span=(0, 5), match='sales'>
```

In [ ]:

```
re.search('^45' <==line starts with 45 ....)
re.search('^s' <== line starts with char 's')
re.search('^\s',inpuString)
           ----> line starts with space
```

In [ ]:

```
^ ==> ^pattern ==> matches the pattern line begins with

$ ==> pattern$ ==> matches the pattern line ends with
       re.search('sales$','safadss sales') ->OK

^pattern$ ==> like condition - pattern only style
---------
re.search('^sales','sales any text') # OK
re.search('sales$','anytext sales') # OK
|
re.search('^sales$','sales') # OK
re.search('^sales$','sales,') # Not-OK

. (dot) - match any single char except \n
re.search('^...','abcdefg') -> 'abc'
            ===
             |->line starts with any 3 chars
```

```
re.search('..$','abcdefg') -->'fg'

re.search('^sales..data$'
            =============
               |->pattern only style
        --> line starts with sales followed by any two chars endswith data

.* ==>list of all

Regx supports - character based search  - []
------------------------------------------
[]  - 1char
[][] - 2chars

[Aak]run
----------->Arun arun krun
[Aa][Rr]un
----------------->ARun Arun aRun arun

re.search('network[sx5t]$'
           networks
                  =
           networkx
                  =
           network5
                  =
           networkt
                  =

[a-z] - lowercase
[A-Z] - uppercase
[a-zA-Z] - alpha
[0-9] - digits    (or) \d
[a-zA-Z0-9] - alpha number  (or) \w

^[a-zA-Z].*[a-z]$
 ================
   |->line starts with any alpha followed by any text ends with any singlelowerc

^[a-z] <== line starts with any lowercase
    Vs
 [^a-z] <== NOT matching any lowercase chars

[^a-zA-Z0-9\s] - NOT Matching alpha number -- Match specialchars
   (or)
   [^\w\s]

^$ - empty line
```

In [ ]:

```
2. Extended Regular Expression - ERE -> Multiple pattern
                                 =====
                                 | () + {}

| => pattern1|pattern2  - any one pattern /any where is matched - OK
() => (pattern1)(pattern2) - both pattern should match same order

+ ==>  <pattern>+  -- 1time (or) more times
ab+c ---> abc abbbbbbbbbbbbbbbbbbbbbbbbbbbbbc //OK
              abbbbb,bbbbbbbbbbc //Not-Matched
```

```
{}
<pattern>{n} - n times
ab{3}c ==> abbbc //OK  abc abbc abbbbc //Not-OK

<pattern>{n,} - minimum times maximum nolimit
ab{3,}c ==> abbbc abbbbbbbbbbbbbbbbbbbbbbbbc //OK
           abc abbc //Not-Matched

ab+c ---same as-- ab{1,}c

<pattern>{n,m} - minimum 'n' times maximum 'm' times
ab{3,5}c ==> abbbc abbbbc abbbbbc //OK
           abc abbc abbbbbbc abbbbbbbbbbbbbc //Not-Matched
```

In [45]:
```
re.search('sales|prod|devops|de','list of sales emp records')
```

Out[45]:
```
<re.Match object; span=(8, 13), match='sales'>
```

In [47]:
```
print(re.search('(sales)(pune)','101,raj,sales,pune,1000'))
               # salespune
```

```
None
```

In [48]:
```
print(re.search('(sales).(pune)','101,raj,sales,pune,1000'))
               # sales<Char>pune
```

```
<re.Match object; span=(8, 18), match='sales,pune'>
```

In [49]:
```
print(re.search('(pune).(sales)','101,raj,sales,pune,1000'))
               # pune<Char>sales
```

```
None
```

In [ ]:
```
re.search('^\s+\d.*(dbus|net|api).*[a-e]$',inputString)
           ====
line starts with 1 or more space followed by any digits followed by list of anyt
anywhere substring called dubs (or) net (or) api any one pattern is matched foll
list of all ends with 'a'(or)'b'(or)'c'(or)'d'(or)'e'  //pattern only
```

In [ ]:
```
^[a-zA-Z][a-zA-Z][a-zA-Z][0-9][0-9][0-9][0-9][0-9][a-z][a-z]$  <== BRE
        Vs
^[a-zA-Z]{3}[0-9]{5}[a-z]{2}$ <== ERE
```

In [50]:
```
s='root:x:bin:bash:linux:usr:python'
s.split(':')
```

Out[50]:
```
['root', 'x', 'bin', 'bash', 'linux', 'usr', 'python']
```

In [51]:
```
s='root:x,bin-bash(linux)usr%python'
s.split(':')
```

Out[51]:
```
['root', 'x,bin-bash(linux)usr%python']
```

In [ ]:
```
# re.search(pattern,inputString,re.I) --> <ack>/None
# re.sub(oldpattern,replacestr,inputString,count,re.I) -->str

# re.split(pattern,inputString) --> list_output
# -------------------------------------------------
```

In [52]:
```
re.split('[^a-zA-Z0-9\s]',s)
```

Out[52]:
```
['root', 'x', 'bin', 'bash', 'linux', 'usr', 'python']
```

In [53]:
```
re.split('[^\w\s]',s)
```

Out[53]:
```
['root', 'x', 'bin', 'bash', 'linux', 'usr', 'python']
```

In [ ]:
```
>>>
>>> len(os.listdir('.'))
189
>>> import re
>>>
>>> for var in os.listdir('.'):
...     if(re.search('pdf$',var)):
...             print(var)
...
attention.pdf
>>>
>>> for var in os.listdir('.'):
...     if(re.search('pdf$|csv$',var)):
...             print(var)
...
attention.pdf
emp.csv
QA.csv
r1.csv
used_cars_data.csv
>>>
```