24-5-23.

Parsing

Parsing: Parsing is a phenomenon of converting any data format to another data format.

-> Parsing is the phenomenon of converting any data to the string format/ any other format.

parser:

-> It is a translator which will parse our data into specified format.

-> there are many types of parsers available in the market.

1. Lexical parser
2. yacc parser
3. xml parser
4. json parser
5. pickle parser

-> lexical and yacc parser are mainly used in system software.

-> xml parser, json parser and pickle parser are used in application Software.

-> xml parser used in application/networking Software, it is difficult for developers and they will not prefer it.

Json parsing technique:

-> Json stands for javascript object notation. it's annotation where the data will be in the form of a javascript object will be completely similar to the string type.

-> By using json parser we convert the normal data into the json data or json data into the normal data format without data loss.

-> In the application programming interface the data will be in the form similar to a dictionary.

-> The process of converting the data from python to json is called serialization or another name called dumping and it will be done by using functions called dumps or dump function. It is converting from python to json format. Finally it will give the result like a string data type.

-> The process of converting the data from json format to python format is called deserialization and another name is called loading. This can be done by the load and loads function. Finally, it will give the result like original python data types.

-> If we can perform json techniques first import the json module in our python file. import json.

Dumping function:

json.dumps():

-> It is a function which is used to convert the data into json format and return the converted data.

Syntax:

```
json.dumps(data)
```

Note: only it takes data/values.

Example:

```
data = {'name': 'sam', 'phone': 12548 9,
        'email': 'sam@gmail.com',
        'list': [1,2,3,5,6,'hello'], 'str': 'helloworld'}
fo = open('parsing.txt','w')
res = json.dumps(data) # Converting the python code to json code.
fo.write(res)
fo.close().
```

json.dump():

-> It is a function which is used to convert the data into json format as well as write it into a specific file (.json, .txt, .doc)

Syntax:

json.dump(data, file pointer)

Note: it takes data/values, with file-pointer.

Example:

```
fo = open("par.json", "w")
data = {"name": "sam", "phone": 125489,
        "email": "sam@gmail.com",
        "list": [1,2,3,5,6,"hello"], "str":
        "helloworld"}
json.dump(data, fo)
fo.close().
```

Loading function:

json.loads:

-> It is a function which is used to convert json format into the data and return the converted data.

Syntax :-

```
json.loads(data)
```

Note: only it takes data/values.

Example :

```
fo = open('par.json', 'r')
res = fo.read()
r = json.loads(res)        # converting the json code to python code.
print(r)
print(type(r))
```

json.load :

-> It is a function which is used to convert json format into the python data as well as read it into a specific file (json, txt, doc) and return the converted data.

Syntax :

```
json.load(filepointer)
```

Note : it takes filepointer.

Example :

```
fo = open('par.json', 'r')
res = json.load(fo)
print(res)
fo.close().
```

25-5-23

```
import json
# serilization
d = {'name': 'sai', 'age':25, 'phone': 9988774455,
     'email': 'sai@gmail.com'}
print(d, type(d))
res = json.dumps(d)
print(res, type(res))

print('-*-' * 30)
# deserilization

print(res, type(res))
temp = json.loads(res)
print(temp, type(temp))
```

o/p :-

{'name': 'sai', 'age':25.... } <class 'dict'>

{"name":"sai", "age":25.... } <class 'str'>

-*--*-*--

{"name":"sai", "age":25....} <class 'str'>

{'name': 'sai', 'age':25...} <class 'dict'>

```
import pickle
# encryption.

d = {'name': 'sai', 'age': 25, 'phone': ....}
print(d, type(d))
res = pickle.dumps(d)
print(res, type(res))

print('-*-' * 30)
# decryption.

print(res, type(res))
temp = pickle.loads(res)
print(temp, type(temp))
```

Op:-

```
{'name': 'sai', 'age': 25, ....} <class 'dict'>
b'\x80\x04\. . . . . ' <class 'bytes'>

-*--*-

b'\x80\x04\. . . . . ' <class 'bytes'>
{'name': 'sai', 'age': 25, ....} <class 'dict'>
```

Dumps Concept in file handling (write operation)

```
import json
d= {'sai':{'name':'sai', 'age':25,
    'phone':.....},'ramya':
    {'name'.....}}
fp=open('employee.json','w')
res=json.dumps(d)
fp.write(res)
fp.close()
```

Dump concept in file handling (write operation)

```
import json
d = {.....}
fp=open('student.json','w')
json.dump(d,fp)
fp.close()
```

loads concept in file handling (read operation)

```
import json
fp = open('employee.json','r')
res = fp.read()
```

```
print (res, type(res))

temp = json.loads (res)
print (temp, type (temp))
fp.close( )
```

load concept in file handling (read operation)

```
import json
fp = open ('employee.json', 'r')
res = json.load (fp)
print (res, type(res))
```

pickle dumps concept in file handling (write operation)

```
import pickle
d = {                    }
fp = open ('employee.pickle', 'wb')
res = pickle.dumps (d)
fp.write (res)
fp.close ( )
```

pickle dump concept in file handling

```
import pickle
d = {  .  .  .  .  }
fp = open ('student.pickle', 'wb')
res = pickle.dump (d, fp)
fp.close( ).
```

pickle loads concept in file handling (read operation)

```
import pickle
fp = open ('employee.pickle', 'rb')
temp = fp.read()
res = pickle.loads (temp)
print(res)
fp.close()
```

pickle load concept in file handling (read operation)

```
import pickle
fp = open ('employee.pickle', 'rb')
res = pickle.load(fp)
print(res)
fp.close()
```

register.

```
from requests import request
from json import loads

url = "https://reqres.in/api/register"
payload = {'email': "sai@reqres.in",
           "password": 'sai'}
response = request("Post", url, json=Payload)
print(response.status_code)
```

```
from requests import request
from json import loads

url = "https://reqres.in/api/login"
payload = {'email': "sai@reqres.in",
           "password": 'sai'}
response = request("post", url, json=payload)
print(response.status_code)
```

O/P :-

400

```
print(response.text)
res = json.loads(response.text)
print(res)
if res['id'] == '630':
    print("pass")
else:
    print("fail")
```

27-5-23.

Pickle Parsing technique:

-> It is a phenomenon of converting the data into the encrypted format (unreadable format like binary format).

Pickle Parser:

-> It is a parser which is responsible for converting the data into the unreadable format which will be like a binary.

-> To store pickle data we can use a special file called '.pkl'

-> The process of converting the data into pickled format is called pickling.

-> the process of converting the pickle format to normal data is called as unpickling.

Pickle function:

pickle.dumps(): It is a function which is used to convert the data into encrypted format and return the converted data.

Syntax:

pickle.dumps(data)

Note: only it takes data/values.

Example:

```
fo = open('parsing.txt', 'a')

data = {'name': 'sam', 'Phone': 125489,
        'email': 'sam@gmail.com',
        'list': [1, 2, 3, 5-6, 'hello'],
        'str': 'helloworld'}
res = pickle.dumps(data)

fo.write(res)
fo.close()
```

pickle.dump():

-> It is a function which is used to convert the data into encrypted format as well as write it into a specific file (json, txt, doc) and return the converted data.

Syntax:

pickle.dump(data, filepointer)

Note: it takes /values, filepath.

Example:

```
fo = open('par.pkl', 'w')
data = {'name': 'sam', 'Phone': 1254899,
        'email': 'sam@gmail.com',
        'list': [1,2,3,5,6,'hello'],
        'str': "helloworld"}
pickle.dump(data, fo)
fo.close()
```

unpickle function:

pickle.loads():

-> It is a function which is used to convert decrypt format into the data and return the converted data.

Syntax:

```
pickle.loads(data)
```

Note: only it takes data/values.

Example:

```
fo = open('par.json', 'r')
res = fo.read()
r = pickle.loads(res) # converting the decrypt code to python code.
```

```
print(r)
print(type(r))
```

pickle.load :

-> It is a function which is used to convert decrypt format into the data as well as read it into a specific file (json, txt, doc) and return the converted data.

Syntax :

```
pickle.load(filePointer)
```

Note : it takes a filePath.

example :

```
fo = open('par.json', 'r')
res = pickle.load(fo)
print(res)
fo.close().
```

Package

Package :-

-> It is a set of programs designed to do a specific task.

-> It is a folder containing a set of programs / files designed for specific tasks.

-> If we want to call a folder as a Package, then that folder must have a `__init__.py` file in it.

-> A Package is a way to organize related modules and sub-packages together. It provides a hierarchical structure for organizing and managing code in larger projects. Packages help avoid naming conflicts and provide a structured approach to organizing Python code.

Here are the key concepts related to Packages in Python:

Package structure: A package is a directory that contains one or more python module files (.py files) and an optional `__init__.py` file.

-> The `__init__.py` file is executed when the package is imported and can contain

initialization code or define variables, functions, or classes that are accessible within the package.

Package structure: A package is a directory that contains one or more Python modules files (.py files) and an optional __init__.py file.
-> The __init__.py file is executed when the package is imported and can contain initialization code or define variable, functions, or classes that are accessible within the package.

Sub-packages: packages can have sub-packages which are simply nested packages within a package. Sub-packages have their own __init__.py file and can contain modules and sub-packages, forming a hierarchical structure.

Importing packages: To use code from a package, you need to import ~~int~~ it into your python script or module. You can use the import statement to import a package, a module within the package, or specific objects from a module.

Packages create their own namespace, meaning that the objects (variables, functions, classes) defined within a package are accessed using the package name as a prefix.

-> for example, if there is a function. myfunction. inside mymodule in mypackage you would access it as mypackage. mymodule. myfunction ( ).

--init---.py :- It is an initialization file which will have the code related to the platform setup for environment setup.

-> when you execute a project then every --init---.py file will get executed.

-> pro environment setup will be done by --init---.py file.

Accessing the data from 1 file to other :- It can be achieved by using 2 keywords.

A. from B. import

① from :- It is a keyword which is used to specify the location.

② import :- It is a keyword which is used to specify the content that needs to be imported.

-> we should always consider the path after the project name because the application knows the address into the project name.

31-5-23

Program :-

```
st = "pyspiders  <span>9988774455</span>
      qspiders  456  jsp <p>5566443322<p>
      test  <h1> 7788996655 </h1>".
```

o/p : ['9988774455', '7788996655']

```
l = [ ]
s = ''
for i in st:
    if '0' <= i <= '9':
        st = i
    else:
        if len(s) == 10 and s[0] in '6789':
            l += [s]
        s = ''
print(l)

import re
res = re.findall('[6-9][0-9]{9}', st)
print(res)
```

o/p :-

['9988774455', '7788996655']

1-6-23. Regular expression

-> Regular expression is the collection of special symbol which has a predefined meaning which is associated with each and every special symbol it performs the action.

-> It defines the structure that is formed using the set of special symbols and values is called as pattern.

-> Regular expression helps us in extracting the required data from the collection of data.

-> If we want to represent a group of strings according to a particular format/pattern then we should go for regular expressions.

-> regular expressions is a declarative mechanism to represent a group of strings according to a particular format/pattern.

-> Re is used in unstructured collection data (file system).

Eg1: we can write a regular expression to represent all mobile numbers.

Eg 2: we can write a regular expression to represent all mail ids.

The main important application areas of regular expressions are.

1. To develop validation frameworks / validation logic.
2. To develop Pattern matching applications (ctrl-f in windows, grep in UNIX etc)
3. To develop translators like compilers, interpreters etc
4. To develop digital circuits.
5. To develop communication Protocols like TCP/IP, UDP etc.

-> We can develop regular expression based applications by using python module : re

-> This module contains several inbuilt functions / special classes to use regular expressions very easily in our applications.

-> re is name of the module that is used in the regular expressions to utilize the re module we need to import it first using import statement

import re.

Character Classes :-

a - it matches only one character called a

aa - it matches only 2 characters called aa

[a-z] - this matches lower case alphabets

[^a-z] - this matches except lower case alphabet

[A-Z] - this matches upper case alphabets

[^A-Z] - matches except upper case alphabet.

[a-zA-Z] - matches both the upper and lower case alphabet

[^a-zA-Z] - matches except the upper and lower case alphabet.

[a-zA-Z]+ -> matches both U.C and l.C minimum should be one char and maximum should be ∞ char.

[a-zA-Z]* -> matches both U.C and l.C minimum zero chars and maximum ∞ no. of chars.

[a-zA-Z]{m,n} -> m and n are numeric which tells us to match the alphabets from m to n.

[a-zA-Z]{n} – it matches exactly n alphabets.

[0-9] – matches the numeric character which is equivalent to ⟶ \d

[^0-9] – matches the characters except numeric character ⟶ \D

[a-zA-Z0-9] – matches one alphanumeric character which is equivalent to ⟶ \W.

[^a-zA-Z0-9] – matches one character except alphabet or number ⟶ \W.

\ – is SP1 symbol used to remove the special meaning of the special symbol.

Predefined Classes :-

The special sequences consist of "\" and a character from the list below. If the ordinary character is not on the list, then the resulting RE will match the second character.

\number matches the contents of the group of the same number.

\A matches only at the start of the string.

\z matches only at the end of the string.

\b matches the empty string, but only at the start or end of a word.

\B matches the empty string, but not at the start or end of a word.

\d matches any decimal digit; equivalent to the set [0-9] in bytes patterns or string.

-> patterns with the ASCII flag.

-> In string patterns without the ASCII flag, it will match the whole range of unicode digits.

\D matches any non-digit character (except numeric); equivalent to [^\d]

\s matches any whitespace character; equivalent to [\t\n\r\f\v] in bytes patterns or string.

-> Patterns with the ASCII value.

-> In string patterns without the ASCII flag, it will match the whole range of

Unicode whitespace characters.

\S matches any non-whitespace character; equivalent to [^\s] in byte patterns or string patterns with the ASCII flag.

-> In string patterns without the ASCII flag, it will match the range of Unicode alphanumeric characters (letters plus digits plus underscore).

-> with Locale, it will match the set [0-9] plus characters defined as letters for the current locale

\W matches the complement of \w, equivalent to [^\w].

Quantifiers classes :

a ——> exactly one characters

Aa ——> continually / exactly that two characters

a+ ——> at least one 'a' will be consider (1 to n no of characters)

a* ——> any number of 'a', including 0 (0 to n no of characters).

a? ——> at most one 'a' will Consider or zero 'a' will consider (either one time or zero time)

a{m} ——> exactly m numbers of a's.

a{x,y} ——> minimum x no of a's and maximum y no of a's in (range)

^ ——> matches the except of that string or matches the start of the string.

\A ——> matches the start of the string.

$ ——> matches the end of the string.

\z ——> matches the end of the string.

The special characters are :

". " Matches any character except a newline.

"^" matches the start of the string.

"$" matches the end of the string or just before the newline at the end of the string.

"*" matches 0 or more (greedy) repetitions of the preceding RE.

"+" matches 1 or more (greedy) repetitions of the preceding RE.

-> Greedy means that it will match as many repetitions as possible.

"?" matches 0 or 1 (greedy) of the preceding RE.

{m,n} matches from m to n repetitions of the preceding RE

{m,n}? Non-greedy version of the above.

"\" either escapes special characters or signals a special sequence.

[] : Indicates a set of characters.

-> A "^" as the first character indicates a complementing set.

"|" A|B, creates an RE that will match either A or B.

(...) Matches the RE inside the parentheses. The contents can be retrieved or matched later in the string.

2-6-23.

1. import re

st = "PYTHON IS A PROGRAMMING LANG PYTHON IS A EASY LANG"

res = re.findall('PYTHON', st)

print(res)

o/p :-

['PYTHON', 'PYTHON']

2. import re

st =

res = re.findall('p', st)

print(res)

o/p :-

['p', 'p', 'p']

3. import re

```
st =
res=re.findall('[A-Z]{2}', st)
print(res)
```

O/P :-

['PY', 'TH', 'ON', 'IS', 'PR', 'OG', 'RA', 'MM', ...]

4. import re

```
St =
res = re.findall('[0-9]', st)
print(res)
```

O/P :-

['1', '2', '3', '9']

5. import re

```
St =
res=re.findall('[^0-9]', St)
print(res)
```

O/P :-

['P', 'Y', 'T', 'H', 'O', 'N', 'I', ...]

6. import re

```
res = re.findall('[^a-z, A-Z 0-9]', st)
print(res)
```

O/P :-

[' ', ' ', ' ', ' ', ' ', ' ', ' ', '&', ...
...]

7. import re

```
res = re.findall('[A-Z]+', st)
print(res)
```

['PYTHON', 'IS', 'A', 'PROGRAMMING', 'LANG', 'PYTHON', 'I', 'A', 'EASY', 'LANG'].

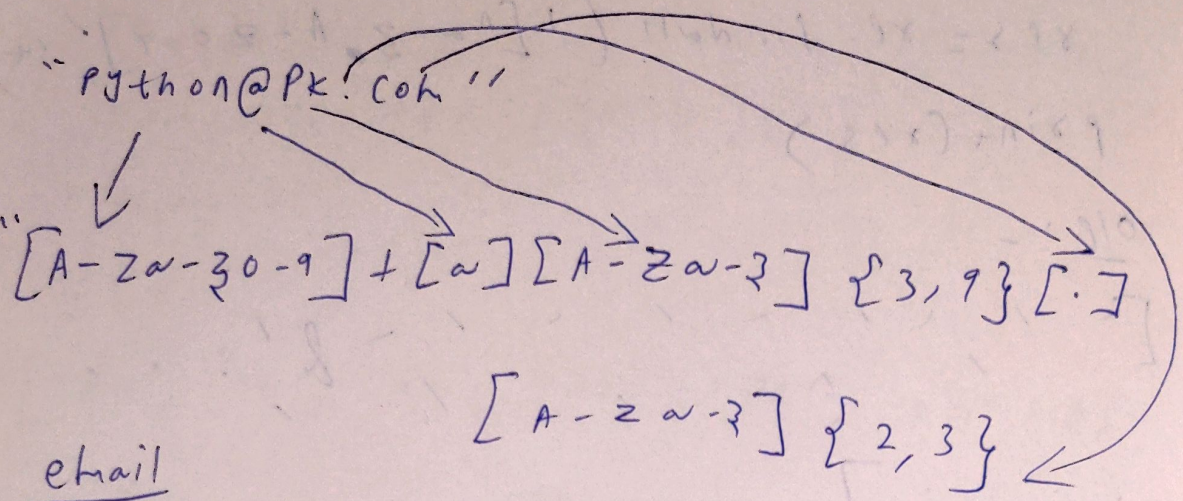
8. import re

```
res = re.findall('[A-Z]*', st)

print(res)
```

['PYTHON', '', 'IS', '', 'A', '', 'PROGRAMMING', '', '', '', '', '', '', '', ...].

email id Pattern



email

ex: (1)

```
import re
st = "python@pyc.compython is a scripting
    lang pythondev@pyc.com pavan@gmail.
    comrvvnvgvg.",
res=re.findall("[a-zA-Z0-9]+[@]
    [a-zA-Z]{3,9}[.][A-Za-z]
    {2,3}", st)
print(res)
```

o/p :-

['python@pyc.com', 'pythondev@pyc.com',
'pavan@gmail.comrvvnvgvg']

② import re

```
st = "python@pyc.com python is a scripting lang python.@pyc.com pavan@gmail.co.in hrvvnvgvg";
res = re.findall("[a-zA-Z0-9]+[.]?[A-Za-z0-9]*[@][a-zA-Z]{3,9}[.][A-Za-z]{2,3}[.]?[a-zA-Z]*", st)
print(res)
```

['python@pyc.com', 'python.@pyc.com', 'pavan@gmail.co.in']

3-6-23 . Numbers

① import re

```
st = "9988774455 bbcbs +918956234578 18004558999 dsf×jhh ughdas 1800 456 1889 5700009565"
res = re.findall("[+][0-9]{2}[0-9]{10}", st)
print(res)
```

o/p :-

['+918956234578']

② import re

```
res = re.findall("[0-9]{10}", st)
print(res)
```

['9988774455', '9189562345', '1800455899',
'5700009565']

③ import re

```
st = "9988774455 bbcbs +91 8956234578
1800455899 dsfx3jhh ughdas 1800 456
1884 hgh 5700009565"
res = re.findall("\d{4}[ -]?\d{3}
[ -]?\d{3,4}", st)
print(res)
```

o/p :-

['9988774455', '8956234578', '1800455899',
'1800 456 1884', '5700009565']

④ import re

```
st = "9988774455 bbcbs +91 8956234578
jgjh +911800455899 dsf2 3jhh
ughdas hgh 5700009565"
res = re.findall("\+\d{2}\s?\d{10}", st)
print(res)
```

o/p :-

['+91 8956234578', '+911800455899']

(5) import re

st = "9988774455 bbcbs +91 8956234578
+92 1800455899 dsfx3jhh ughdas
1800 456 1899 hgh 57000095 65"

res = re.findall("\+\d{2}\s?\d{10}|
\d{4}[,]?\d{3}[,]?\d
+", st)

O/P :-

['9988774455', '+91 8956234578',
'+92 1800455899', '1800 456 1899',
'57000095 65']

Websites

(1) import re

st = """ sdhasgdsas http://www.FB.co.
in/signup/ hdkjsfgdf hfk
https://www.youtube.com/login/
sahgdhashdsga http://www.google.com/
ghg www.insta.com"""

res = re.findall("http[s]?://[wW]{3}.
\w+.[a-zA-Z]{2,3}[.]?[a-zA-Z]{2,3}
/|W*/?", st)

print(res) O/P :-

['http://www.FP.co.in/singup/', 'https://www.youtube
.com/login/', ...]

② import re

```
res = re.findall(r'[\w]{3}.\w+.
                  [a-zA-Z]{2,3}[.],
print(res)        [a-zA-Z]{2,3}', st)
```

O/P :-

```
['www.FB.Co.in', 'www.youtube.com',
 'www.google.com', 'www.insta.com']
```

③ import re

```
res = re.findall(r'[\w]{3}.\w+.
    [a-zA-Z]{2,3}[.]?[a-zA-Z]
    {2,3}|http[s]?://[\w]{3}.
    \w+.[a-zA-Z]{2,3}[.],
    [a-zA-Z]{2,3}/\w*/?', st)
print(res)
```

O/P :-

```
['http://www.FB.Co.in/signup/',
 'https://www.youtube.com/login/',
 'http://www.google.com/',
 'www.insta.com']
```

6-6-23.

Multi-Threading

Multi-Tasking: Executing several tasks simultaneously is the concept of multitasking.

There are 2 types of multi tasking:

1. Process based multi-tasking 2. thread based multi-tasking.

1. Process based multi-Tasking: Executing several tasks simult. where each task is a seperate independent process is called multi-tasking.

Eg: while typing Python program in the editor we can listen mp3 songs from the same system. At the same time, we can download a file from the internet. All the these tasks are executing simultaneously and independent of each other. Hence it is process based multi-tasking.

Note: This type of multi-tasking is best suitable at OS level.

2. Thread based multitasking: Executing several tasks simult. where each task is a seperate independent part of the same program, is called thread based multi-tasking, and each independent part is called a thread.

Note: This type of multi-tasking is best suitable at program. level.

Note: Whether it is process based or thread based, the main advantage of multi-tasking to improve performance of the system by reducing response time.

The main important application areas of multi-threading are:

1. To implement multimedia graphics
2. To develop animations
3. To develop video games
4. To develop web and application servers etc....

Note: where ever a group of independent jobs are available, then it is highly recommended to execute simult. instead of executing one by one. For such type of cases we should go for multi-threading.

-> Python provides one inbuilt module "threading" to provide support for developing threads. Hence developing multi-threaded programs is very easy in Python.

-> Every Python program by default contains one thread which is nothing but main thread.

Q. Program to print name of current executing thread:

```
import threading
print("Current executing thread:", threading.current_
        thread().getName())
```

O/P :- Current executing thread : Main thread.

7-6-23.

```
from threading import *
def show():
    for i in range(1,11):
        print("main thread")
def display():
    for i in range(1,21):
        print("child thread")
t1 = thread(target = show)
t2 = thread(target = display)

t1.start()
t2.start()

t1.join()
t2.join()
```

by using Normal function :-

```
def show():
    for i in range(1,11):
        print("main thread")
def display():
    for i in range(1,11):
        print("child thread").
show()
display().
```

```
import time
from threading import *
start = time.time()
def show():
    for i in range(1, 11):
        print(f"show: {i**2}")
def display():
    for i in range(1, 15):
        print(f"display: {(i*3)/2}")
t1 = thread(target = show)
t2 = thread(target = display)
t1.start()
t2.start()
t1.join()
t2.join()
stop = time.time()
print(stop - start)
```

The ways of creating thread in Python we can create a thread in Python by using 3 ways.

1. creating a thread without using any class
2. creating a thread by extending thread class
3. creating a thread without extending thread class.

1. Creating a thread without using any class:

```
from threading import *
def display():
    for i in range(1,11):
        print("child thread")
t = thread(target = display) # creating thread object
t.start() # starting of thread.
for i in range(1,11):
    print("main thread")
```

-> if multiple threads are present in our program, then we cannot expect execution order and hence we cannot expect exact output for the multi-threaded programs. Because of this we cannot provide exact output for the above program. It varies from machine and run to run.

Note: Thread is a pre-defined class present in the threading module which can be used to create our own threads.

2. Creating a thread by extending thread class:

-> we have to create a child class for the thread class. In that child class we have to override the run() method with our required job. whenever we call start() method then automatically run() method will be executed and performs our job.

```
from threading import *
class mythread(thread):
    def run(self):
        for i in range(10):
            print("child thread-1")
t = mythread()
t.start()
for i in range(10):
    print("main thread-1")
```

3. creating a thread without extending thread class:

```
from threading import *
class test:
    def display(self):
        for i in range(10):
            print("child thread-2")
obj = test()
t = thread(target = obj.display)
t.start()
for i in range(10):
    print("main thread-2")
```