In [ ]:
```
import re
re.search("Pattern","inputstring") -><Ack>/None
re.findall("Pattern","inputstring") ->[Result]/[]

re.split("Pattern","inputstring")->[ ]

re.compile("Pattern") --><PObject>
<PObject>.search("inputstring")
<PObject>.findall("inputstring")
<Pobject>split("inputstring")
```

In [ ]:
```
lambda - unnamed function - function call with arguments return a value

i.e., def f1(args):
           ....
           return value
lambda - expression and nested call
Syntax:-
--------
lambda args:expression
```

In [2]:
```
def f1(a1,a2):
    return a1+a2

f1(10,20)
f1("A","B")
```

Out[2]: 'AB'

In [5]:
```
f2=lambda a1,a2:a1+a2
f2(10,20)
f2("A","B")
```

Out[5]: 'AB'

In [13]:
```
f3=lambda a:a+100
f3(1000)

f4=lambda a,b:a>b
f4(100,500)

f5=lambda a:a.upper()
f5("abcd")
'''
def f6(a):
    return a.upper()
f6("abcd")
'''
```

Out[13]: 'ABCD'

In [14]:
```
def fx(a):
    return a+100

f6=lambda a1:fx(a1)
f6(10)
```

Out[14]:
```
110
```

In [15]:
```
L=list()
for var in range(5):
    r=var+100
    L.append(r)
print(L)
```

```
[100, 101, 102, 103, 104]
```

In [16]:
```
# [final_value for iterable]
#               -----(1)->--
# ---<-(2)-----

[var+100 for var in range(5)]
```

Out[16]:
```
[100, 101, 102, 103, 104]
```

In [18]:
```
L=list()
for v in range(10):
    if(v>5):
        L.append(v+100)
    else:
        L.append(v+500)
L
```

Out[18]:
```
[500, 501, 502, 503, 504, 505, 106, 107, 108, 109]
```

In [20]:
```
[v+100 if v>5 else v+500 for v in range(10)]
```

Out[20]:
```
[500, 501, 502, 503, 504, 505, 106, 107, 108, 109]
```

In [21]:
```
[v.upper() for v in open("D:\\emp.csv")]
```

Out[21]:
```
['RAM,SALES,PUNE,1000\n',
 'ASHI,PROD,BGLORE,2345\n',
 'XEROX,SALES,CHENNAI,45900\n',
 'YAHOO,PROD,PUNE,32450\n',
 'ANU,HR,HYD,4560\n',
 'BIJU,PROD,BGLORE,4567\n',
 'VIJAY,HR,CHENNAI,3453\n',
 'THEEB,SALES,HYD,5678\n',
 'NITHIN,PROD,PUNE,1236']
```

In [22]:
```
import pprint
d={"CSV":[v.upper() for v in open("D:\\emp.csv")]}
pprint.pprint(d)
```

```
{'CSV': ['RAM,SALES,PUNE,1000\n',
         'ASHI,PROD,BGLORE,2345\n',
         'XEROX,SALES,CHENNAI,45900\n',
         'YAHOO,PROD,PUNE,32450\n',
         'ANU,HR,HYD,4560\n',
         'BIJU,PROD,BGLORE,4567\n',
         'VIJAY,HR,CHENNAI,3453\n',
         'THEEB,SALES,HYD,5678\n',
         'NITHIN,PROD,PUNE,1236']}
```

In [23]:
```
FILE=list()
FILE.append([v.upper() for v in open("D:\\emp.csv")])
FILE
```

Out[23]:
```
[['RAM,SALES,PUNE,1000\n',
  'ASHI,PROD,BGLORE,2345\n',
  'XEROX,SALES,CHENNAI,45900\n',
  'YAHOO,PROD,PUNE,32450\n',
  'ANU,HR,HYD,4560\n',
  'BIJU,PROD,BGLORE,4567\n',
  'VIJAY,HR,CHENNAI,3453\n',
  'THEEB,SALES,HYD,5678\n',
  'NITHIN,PROD,PUNE,1236']]
```

In [ ]:
```
map
filter
reduce ->python 3.x ->functools -> import functools; functools.reduce
# python 3.x                                  vs     python 2.x
map(function,collection) -><map> ->list(map) vs  map(function,collection)->[]
filter(function,collection) -><filter> ->list(filter) vs filter(function,collecti
functools.reduce(function,collection)->Single <-- reduce(function,collection)
```

In [28]:
```
def f1(a1):
    return a1+100

L=list()
for var in range(5):
    rv=f1(var)
    L.append(rv)
L
```

Out[28]:
```
[100, 101, 102, 103, 104]
```

In [29]:
```
list(map(f1,range(5)))
```

Out[29]:
```
[100, 101, 102, 103, 104]
```

In [30]:
```
list(map(lambda a:a+100,range(5)))
```

Out[30]:
```
[100, 101, 102, 103, 104]
```

In [33]:
```
L=list(map(lambda a:a.upper(),open("D:\\emp.csv")))
L
```

Out[33]:
```
['RAM,SALES,PUNE,1000\n',
 'ASHI,PROD,BGLORE,2345\n',
 'XEROX,SALES,CHENNAI,45900\n',
 'YAHOO,PROD,PUNE,32450\n',
 'ANU,HR,HYD,4560\n',
 'BIJU,PROD,BGLORE,4567\n',
 'VIJAY,HR,CHENNAI,3453\n',
 'THEEB,SALES,HYD,5678\n',
 'NITHIN,PROD,PUNE,1236']
```

In [37]:
```
def f1(a):
    if(a>10):
        return True
    else:
        return False

L=list()
for var in range(15):
    rv=f1(var)
    L.append(rv)
print(L)

print(list(map(lambda a:a>10,range(15))))
```

```
[False, False, False, False, False, False, False, False, False, False, False, T
rue, True, True, True]
[False, False, False, False, False, False, False, False, False, False, False, T
rue, True, True, True]
```

In [38]:
```
print(list(filter(lambda a:a>10,range(15))))
```

```
[11, 12, 13, 14]
```

In [39]:
```
list(filter(lambda a: a in "python",["java","python","perl","sql"]))
```

Out[39]:
```
['python']
```

In [45]:
```
F=open("D:\\emp.csv")
L=F.readlines()
for var in L:
    cost=var.split(",")[-1]
    if(int(cost)>30000):
        print(cost)
```

```
45900

32450
```

In [53]:
```
list(filter(lambda a:a>30000,[int(var.split(",")[-1]) for var in L]))
```

Out[53]:
```
[45900, 32450]
```

In [56]:
```
L=list(filter(lambda a:a>30000,[int(var.split(",")[-1]) for var in open("D:\\emp.
L
```

Out[56]:
```
[45900, 32450]
```

In [62]:
```
import functools
functools.reduce
from functools import reduce
reduce(lambda a,b:a+b,['A','B','C','D','E'])
reduce(lambda a,b:a+b,[10,20,30,40,50,60])
```

Out[62]:
```
210
```

In [ ]:
```
function call
call with args
scope of the function members -> global vs return
decorator @function
generator ->yield  -> return vs return
lambda - unnamed
|
functional programming - in-linebased code - not block code
(map,filter,reduce)
```

In [65]:
```
reduce(lambda a,b:int(a)+int(b),[var.split(",")[-1] for var in open("D:\\emp.csv'
```

Out[65]:
```
101189
```

In [69]:
```
s="345:"
int(s.strip(":"))
```

Out[69]:
```
345
```

In [70]:
```
d={"cost":reduce(lambda a,b:int(a)+int(b),[var.split(",")[-1] for var in open("D:
L=[reduce(lambda a,b:int(a)+int(b),[var.split(",")[-1] for var in open("D:\\emp.c
T=(reduce(lambda a,b:int(a)+int(b),[var.split(",")[-1] for var in open("D:\\emp.c
if reduce(lambda a,b:int(a)+int(b),[var.split(",")[-1] for var in open("D:\\emp.c
    print("Action-1")
else:
    print("Action-2")
```

```
Action-1
```

In [ ]:
```
# Global Interpreter Lock (or) GIL - like mutex(lock)
#
# in python everything is an object
# each object having own address
# a=10    10 |0x1234
# b=11    11 |0x3466
# c=4+6  ------------------reference to 0x1234
#
# by default CPython (original) - lock - GIL
#
# GIL - allows only one thread to hold the control of the python interpreter
#

# a=[]
# b=a
# ---
#
```

In [ ]:
```
# Process - Data + Address
# Thread - Data

# Process
# --------
#  ->execution of DATA(File)
#  ->Address
#  ->State
# Process Ctrl Block (PCB)

# P1   Wait  - PID:101  -> Address:0x1234
# |
# P2 - Child R+(Running) - PID:102 ->Address->0x3456
#

# Thread - DATA
# -------------------
# Thread Ctrl Block (TCB)
#  -> TID
#  -> StackPointer
#  -> PPP(Parent PRocess Pointer) ->PCB (Process ctrl Block)
#
# file:p1.py <=== process
# ----------      |  |  |
#                 t1 t2 t3 - common address
#
# t1 (Parent)- R+ = There is no waiting state
# |
# t2 (Child) - R+

-----                    -------
PCB                        TCB
   <---------------------- PPP
-----                    -------
  |                      |
 -----------             |
  stack -----------------|
 -----------
   |
------------
------------
--------------

|[code][data][file] |
|[reg]       [stack]|
|-------------------|
|     Thread1       |


|[code][data][file] |
|-------------------|
|[reg] |[reg]|[reg] |
|------|-----|------|
|[st]  |[st] |[st]  |
|-------------------|
| T1   | T2  | T3   |
```

```
--------------------
  T - Thread
  st- stack

Thread - flow of execution (or) execution unit(data/instruction)
```

In [ ]:
```
root@host~]# <== working shell (Active process/Running process)
root@host~]# python {Enter} <== newprocess(Child)
>>>
>>> Active process is python ; parent is working shell
>>>                             ------------------------
>>>                                |__Waiting
>>> exit() - Child exit
root@host~]# parent is Active process

file:p1.py
----------------------
import time
def f1():
    time.sleep(5)
def f2():
    time.sleep(15)
print("main-1")
f1()
f2()
print("main-2")
----------------------
```

In [72]:
```
import time
def f1():
    print("\tF1 block")
    time.sleep(5)
    print("\tExit from F1 block")
def f2():
    print("\t--\t F2 block")
    time.sleep(10)
    print("\t--\t Exit from f2 block")
print("main-1")
f1()
f2()
print("main-2")
```

```
main-1
        F1 block
        Exit from F1 block
        --      F2 block
        --      Exit from f2 block
main-2
```

In [ ]:
```
# Thread
# -----------
#   -> createa a thread
#                1 to 1   /  1 to many
#           T1          T1
#           |           |
#          T2     ---------------
#                 |     |       |
#                T2     T3      T4
#

#  -> synchronization/lock
#
threading <== thread module (import threading)
run() - entry point of thread
start() - starts a thread
join() - parent thread section - wait for child thread to terminate
```

In [73]:
```
import threading
import time
def f1():
    print("\t F1 block")
    time.sleep(5)
    print("\t Exit from f1 block")

print("main-1")
f1()
print("main-2")
print("main-3")
print("Exit from main process")
```

```
main-1
         F1 block
         Exit from f1 block
main-2
main-3
Exit from main process
```

In [74]:
```
import threading
import time
def f1():
    print("\t F1 block")
    time.sleep(5)
    print("\t Exit from f1 block")

print("main-1")
tobj=threading.Thread(target=f1)
tobj.start()
print("main-2")
print("main-3")
print("Exit from main process")
```

```
main-1
         F1 blockmain-2

main-3
Exit from main process
         Exit from f1 block
```

In [75]:
```
import time,threading
def f1():
    print("\tF1 block")
    time.sleep(5)
    print("\tExit from F1 block")
def f2():
    print("\t--\t F2 block")
    time.sleep(10)
    print("\t--\t Exit from f2 block")
print("main-1")
t1=threading.Thread(target=f1)
t1.start()
t2=threading.Thread(target=f2)
t2.start()
print("main-2")
print("Exit from main process.")
```

```
main-1
        F1 block
        --       F2 blockmain-2
Exit from main process.

        Exit from F1 block
        --       Exit from f2 block
```

In [98]:
```
import threading

def f1(*a1):
    print(a1)

t1=threading.Thread(target=f1,args=([10,20]))
t1.start()
```

```
(10, 20)
```

In [91]:
```
#help(threading.Thread)
```

In [82]:
```
'''
type('a') -><class 'str'>
type(()) --><class 'tuple'>
type(('a')) -><class 'str'>
type(('a',)) -><class 'tuple'>

v1=10,20,30
type(v1)
'''
```

Out[82]:
```
tuple
```

In [105]:
```
import os
def f1():
    print("This is {} thread.".format(threading.current_thread().name))
    print(threading.get_ident())
    time.sleep(2)
    print("PID:{}".format(os.getpid()))
def f2():
    print("This is {} thread.".format(threading.current_thread().name))
    print(threading.get_ident())
    print("PID:{}".format(os.getpid()))
def f3():
    print("This is {} thread.".format(threading.current_thread().name))
    print(threading.get_ident())
    print("PID:{}".format(os.getpid()))

print("MAIN-THREAD:{}".format(os.getpid()))
t1=threading.Thread(target=f1,name='THREAD-1')
t2=threading.Thread(target=f2,name='THREAD-2')
t3=threading.Thread(target=f3,name='THREAD-3')

t1.start()
t2.start()
t3.start()
print("MAIN-THREAD:{}".format(os.getpid()))
print(t1.is_alive())
```

```
MAIN-THREAD:4020
This is THREAD-1 thread.This is THREAD-2 thread.
5428
PID:4020
This is THREAD-3 thread.MAIN-THREAD:4020

5516

True
7852
PID:4020
PID:4020
```

In [124]:
```
def f1():
    for var in range(1000):
        var=var+1
    print("**** Exit from f1 thread {} ****".format(threading.get_ident()))

print("This is MAIN Thread:{}".format(threading.get_ident()))
for var in range(3):
    th=threading.Thread(target=f1)
    th.start()
print("After creating Thread")
print("Exit from MAIN Thread:{}".format(threading.get_ident()))
```

```
This is MAIN Thread:7952
**** Exit from f1 thread 2452 ****
**** Exit from f1 thread 8652 ****
**** Exit from f1 thread 5260 ****After creating Thread
Exit from MAIN Thread:7952
```

In [127]:
```
def f1():
    print("F1 block")

for var in range(10):
    th=threading.Thread(target=f1)
    th.start()
print("Main section code-1")
print("Main section code-2")
```

```
F1 block
F1 block
F1 block
F1 block
F1 block
F1 block
F1 block
F1 block
F1 block
F1 blockMain section code-1
Main section code-2
```

In [128]:
```
def f1():
    print("F1 block")

th1=threading.Thread(target=f1)
th1.start()
th2=threading.Thread(target=f1)
th2.start()
print("Main section code-1")
print("Main section code-2")
```

```
F1 block
F1 blockMain section code-1
Main section code-2
```

In [129]:
```
def f1():
    print("F1 block")

th1=threading.Thread(target=f1)
th1.start()
th2=threading.Thread(target=f1)
th2.start()
th1.join()
th2.join()
print("Main section code-1")
print("Main section code-2")
```

```
F1 block
F1 block
Main section code-1
Main section code-2
```

In [ ]:
```
# oops - style - syntax
import threading
class classname(threading.Thread):
    def __init__(self):
        super(Box,self).__init__()
        # Threading.Thread.__init__(self) 2.x
    def run(self):
        # this is entry point
```

In [133]:
```
import threading

class Box(threading.Thread):
    def __init__(self):
        super(Box,self).__init__()
        self.name="THREAD1"
    def run(self):
        print("This is {} entry point".format(self.name))

obj=Box()
obj.start()
```

```
This is THREAD1 entry point
```

In [135]:
```
import threading

class Box(threading.Thread):
    def __init__(self,a1):
        super(Box,self).__init__()
        self.name=a1
    def run(self):
        print("This is {} entry point".format(self.name))

obj1=Box("Thread1")
obj1.start()
obj2=Box("Thread2")
obj2.start()
obj3=Box("Thread3")
obj3.start()

obj1.join()
obj2.join()
obj3.join()
print("Exit from main thread")
```

```
This is Thread1 entry point
This is Thread2 entry point
This is Thread3 entry point
Exit from main thread
```

In [136]:
```
import threading

class Box(threading.Thread):
    def __init__(self):
        super(Box,self).__init__()
        self.name="THREAD1"
    def run(self):
        print("This is {} entry point".format(self.name))
    def f1(self):
        print("This is f1 block")
        return ["D1","D2","D3"]

obj=Box()
obj.start() # calling run() method
obj.f1()
```

```
This is THREAD1 entry pointThis is f1 block
```

Out[136]:
```
['D1', 'D2', 'D3']
```

In [137]:
```
import threading

class Box(threading.Thread):
    def __init__(self):
        super(Box,self).__init__()
        self.name="THREAD1"
    def run(self):
        print("This is {} entry point".format(self.name))
        f1() # nested call

def f1():
        print("This is f1 block")
        return ["D1","D2","D3"]

obj=Box()
obj.start() # calling run() method
```

```
This is THREAD1 entry point
This is f1 block
**** Exit from f1 thread 6100 ****
**** Exit from f1 thread 9008 ****
**** Exit from f1 thread 6740 ****
**** Exit from f1 thread 7264 ****
**** Exit from f1 thread 2796 ****
**** Exit from f1 thread 3092 ****
**** Exit from f1 thread 7288 ****
**** Exit from f1 thread 6504 ****
**** Exit from f1 thread 5584 ****
**** Exit from f1 thread 4976 ****
**** Exit from f1 thread 7940 ****
**** Exit from f1 thread 8736 ******** Exit from f1 thread 8432 ******** Exit f
rom f1 thread 8740 ******** Exit from f1 thread 8248 ****



**** Exit from f1 thread 4816 ****
**** Exit from f1 thread 8664 ****
**** Exit from f1 thread 7396 ****
**** Exit from f1 thread 8460 ****
**** Exit from f1 thread 8660 ****
```