#Anonymous function

l=[1,2,3]

r=map(lambda x:x+x ,l)

print(list(r))

#map =helps to create the iterstion ,it returns map object

res=map(lambda n:pow(n,2),l)

print(list(res))

name="sam"

(lambda name:print(name)) (name)

$$$OOPS

1.ABSTRACTION:::

class abstractdemo(ABC):

@abstractmethod #called decorator to make the method(function) abstract one

def display(self):

None

@abtractmethod

def show(self):

None

#changing abstract to concrete

class demo(abstractdemo):

def display(self):

print("abstraction method")

def show(self):

print("2nd function")

obj=demo()

obj.display()

obj.display()

##.INHERITANCE

#one clasxs inheits the properties of the other class

class parent:

def display(self):

print("parent class")

#derived class

class child(parent):

def show(self):

print("child class")

c=child()

c.display() #c is object of the child

c.show()

#3\*\*multiple inheritance:having 2 classes

class A:

n=30

class b(A):

def calc(self):

c=self.n+70

print(c)7

obj=b()

obj.calc()

##Multiple inheritance

##two classes one child -like mom and dad

#inherits propertiues of mom and dad

class dad:

def ddisplay(self):

print("mom class")

class mom:

def mdisplay(self):

print("dad class")

class child(dad,mom):

def cdisplay(self):

print("child class")

obj=child()

obj.ddisplay()

obj.mdisplay()

obj.cdisplay()

##multilevel inheritance

#one class parent and one child class

class grandparent:

def display(self):

print("grandparent class")

class parent(grandparent):

def show(self):

print("parent class")

class child(parent):

def printing(self):

print("child class")

obj=child()

obj.display()

obj.show()

obj.printing()

##HIERARICAL INHERITANCE

##multiple dreived classes for one base class

class parent:

def display(self):

print("parent class")

class child1(parent):

def c1display(self):

print("first child class")

class child2(parent):

def c2display(self):

print("second class")

obj=child1()

obj.display()

obj.c1display()

obj=child2()

obj.display()

obj.c2display()

####HYBRID INHERITANCE

##IT IS JUST LIKE A TREE IN WHICH INHERITING CLASS AND CHILD CHILD CLASS JUSTV LIKE A TREE

class parent:

def display(self):

print("parent class")

class child1(parent):

def p1display(self):

print("child1 class")

class child2(parent):

def p2display(self):

print("child2 class")

class kid1(child1):

def k1display(self):

print("kid1 class")

class kid2(child1):

def k2display(self):

print("kid2 class")

class kidd1(child2):

def k1show(self):

print("kidd1 class of child2")

class kidd2(child2):

def k2show(self):

print("kidd2 class of child2")

obj=child1()

obj.display()

obj.p1display()

obj=child2

obj.display()

obj.p2display()

obj=kid1()

obj.k1display()

obj.p1display()

obj=kid2()

obj.p1display()

obj.k2display()

obj=kidd1()

obj.p2display()

obj.k1show()

obj=kidd2()

obj.p2display()

obj.k2show()

##neon number

num = int(input("Enter a number \n"))

sqr = num\*num #square of num

sumOfDigit = 0

#calculating sum of digits of sqr

while sqr>0:

sumOfDigit =sumOfDigit + sqr%10

sqr = sqr//10

if (num == sumOfDigit):

print("Neon Number \n")

else:

print("Not a Neon Number \n")

$$HAPPY NUMBER OR NOT means sum of digits should be 0

n=int(input("enter the given number"))

while n>=10:

s=0

for i in range(0,len(str(n))+1):

rem=n%10

s=s+rem\*\*2

n=n//10

n=s

if n==1:

print("happy number\n")

else:

print("Not a happy Number \n")