

**def** generator\_Magic(n1):

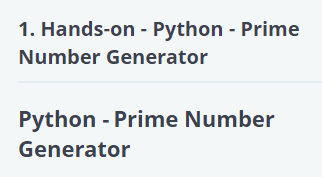
*# Write your code here*

**for** i **in** **range**(3,n1+1):

        m=0

        m=(i\*(i\*\*2+1))/2

**yield** m



**def** primegenerator(num, val):

    flag=0

    a=[]

    b=[]

    a.append(2)

**for** i **in** **range**(3,num):

**for** j **in** **range**(2,i-1):

**if**(i%j==0):

                flag=1

**break**

**else**:

           a.append(i)

**if** val==1:

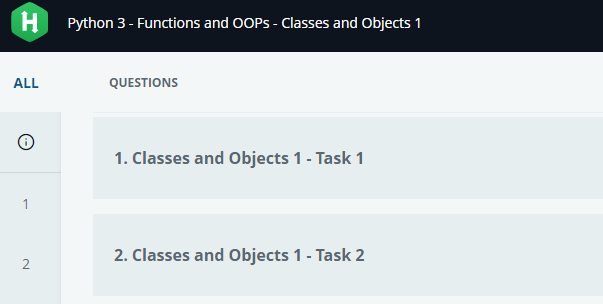
**for** j **in** **range**(0,**len**(a),2):

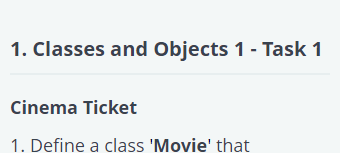
**yield** a[j]

**else**:

**for** j **in** **range**(1,**len**(a),2):

**yield** a[j]





**class** Movie:

**def** \_\_init\_\_(**self**,name,n,cost):

**self**.name=name

**self**.n=n

**self**.cost=cost

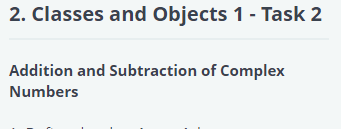
**def** \_\_str\_\_(**self**):

        b="Number of Tickets : "+**str**(**self**.n)

        c="Total Cost : "+**str**(**self**.cost)

        s="Movie : "+**str**(**self**.name)+"\n"+b+"\n"+c

**return**(s)



*#*

*#Write your code here*

**class** comp:

**def** \_\_init\_\_(**self**,real,imaginary):

**self**.real=real

**self**.imaginary=imaginary

**def** add(**self**,p1):

**print**("Sum of the two Complex numbers :"+**str**(p1.real+**self**.real)+"+"+**str**(**self**.imaginary+p1.imaginary)+"i")

**def** sub(**self**,p1):

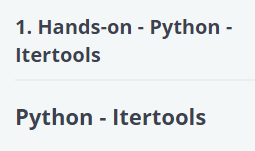
        a=**self**.imaginary-p1.imaginary

**if**(a>=0):

**print**("Subtraction of the two Complex numbers :"+**str**(**self**.real-p1.real)+"+"+**str**(**self**.imaginary-p1.imaginary)+"i")

**else**:

**print**("Subtraction of the two Complex numbers :"+**str**(**self**.real-p1.real)+**str**(**self**.imaginary-p1.imaginary)+"i")



**def** performIterator(tuplevalues):

**import** itertools **as** **iter**

**import** operator

    a=[]

    l=[]

    s=**iter**.cycle(tuplevalues[0])

    j=0

**for** i **in** s:

     j=j+1

**if**(j>4):

**break**

     l.append(i)

    a.append(**tuple**(l))

    l1=**len**(tuplevalues[1])

    a.append(**tuple**(**iter**.repeat(tuplevalues[1][0],l1)))

    a.append(**tuple**(**iter**.accumulate(tuplevalues[2])))

    b=**tuple**(**iter**.chain(tuplevalues[0],tuplevalues[1],tuplevalues[2],tuplevalues[3]))

    a.append(**tuple**(**iter**.chain(tuplevalues[0],tuplevalues[1],tuplevalues[2],tuplevalues[3])))

    a.append(**tuple**(**iter**.filterfalse(**lambda** x:x%2==0,b)))

**return**(**tuple**(a))



**from** cryptography.fernet **import** Fernet

**def** encrdecr(keyval, textencr, textdecr):

*# Write your code here*

    a=[]

    encryptype=Fernet(keyval)

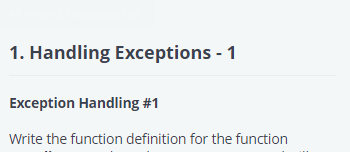
    textencr=encryptype.encrypt(textencr)

    a.append(textencr)

    textdecr=encryptype.decrypt(textdecr)

    a.append(textdecr.decode())

**return** a



**def** Handle\_Exc1():

**try**:

             a=**int**(**input**())

             b=**int**(**input**())

**if**(a>150 **or** b<100):

**raise** ValueError('Input integers value out of range.')

**elif** a+b>400:

**raise** ValueError('Their sum is out of range')

**else**:

**print**("All in range")

**except** ValueError **as** e:

**print**(e)



**from** datetime **import** datetime

**from** datetime **import** date

**def** dateandtime(val,tup):

*# Write your code here*

    main\_list=[]

**if**(val==1):

        d=date(tup[0],tup[1],tup[2])

        main\_list.append(d)

        f\_d=d.strftime("%d/%m/%Y")

        main\_list.append(f\_d)

**if**(val==2):

        time\_stamp=tup[0]

        d=date.fromtimestamp(time\_stamp)

        main\_list.append(d)

**if**(val==3):

        d=datetime(1999,1,1,tup[0],tup[1],tup[2])

        t=datetime.time(d)

        main\_list.append(t)

        f\_t=t.strftime("%I")

        main\_list.append(f\_t)

**if**(val==4):

         d=date(tup[0],tup[1],tup[2])

         weekday=d.strftime("%A")

         main\_list.append(weekday)

         month=d.strftime("%B")

         main\_list.append(month)

         day=d.strftime("%j")

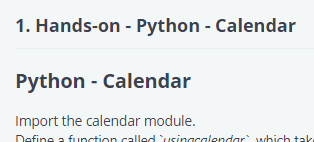
         main\_list.append(day)

**if**(val==5):

        d=datetime(tup[0],tup[1],tup[2],tup[3],tup[4],tup[5])

        main\_list.append(d)

**return** (main\_list)



**import** calendar

**from** collections **import** Counter

**def** usingcalendar(datetuple):

**if**(calendar.isleap(datetuple[0])):

       lst=**list**(datetuple)

       lst[1]=2

       datetuple=**tuple**(lst)

**print** (calendar.month(datetuple[0],datetuple[1]))

    obj = calendar.Calendar()

    l=[]

**for** day **in** obj.itermonthdates(datetuple[0],datetuple[1]):

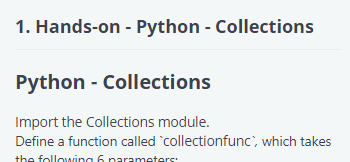
        l.append(day)

**print**(l[-7:])

    count = Counter(d.strftime('%A') **for** d **in** obj.itermonthdates(datetuple[0], datetuple[1]) **if** d.month==datetuple[1])

    common=count.most\_common(1)

**print**(common[0][0])



**import** collections

**from** collections **import** defaultdict

**from** collections **import** Counter

**from** collections **import** OrderedDict

**def** collectionfunc(text1, dictionary1, key1, val1, deduct, list1):

*# Write your code here*

    d = defaultdict(**int**)

**for** w **in** text1.split():

        d[w] += 1

    ks=**sorted**(d.keys())

    od=**dict**()

**for** val **in** ks:

        od[val]=d[val]

**print**(od)

    dc=Counter(dictionary1)

**for** i **in** deduct:

*#ls=list(deduct)*

*#ls[i]=deduct[i]-dc[i]*

        dc[i]=dc[i]-deduct[i]

    dc=**dict**(dc)

**print**(dc)

    od = OrderedDict()

**for** i **in** **range**(**len**(key1)):

        od[key1[i]]=val1[i]

    od.pop(key1[1])

    od[key1[1]] = val1[1]

    od=**dict**(od)

**print**(od)

    d = defaultdict()

    d["odd"] = []

    d["even"] = []

**for** i **in** list1:

**if**(i%2==0):

            d["even"].append(i)

**else**:

            d["odd"].append(i)

**if**(**len**(d["odd"])==0):

**del** d['odd']

**if**(**len**(d["even"])==0):

**del** d['even']

**print**(**dict**(d))



**def** stringmethod(para, special1, special2, list1, strfind):

*# Write your code here*

    l1=**list**(special1)

**for** i **in** l1:

        para=para.replace(i, '')

    word1=para

    l2=word1[0:70]

    word2=l2[::-1]

**print**(word2)

    l3=**list**(special2)

**for** i **in** word2:

        l4=word2.replace(' ','')

**print**(special2.join(l4[i] **for** i **in** **range**(0, **len**(l4), 1)))

    res = [ele **for** ele **in** list1 **if**(ele **in** para)]

**if**(**len**(res)==**len**(list1)):

**print**("Every string in ",list1,"were present")

**else**:

**print**("Every string in ",list1,"were not present")

    wordList=word1.split()

**print**(wordList[:20])

    word = word1.split()

    str2 = []

    str3 = []

**for** i **in** word:

**if** i **not** **in** str2:

            str2.append(i)

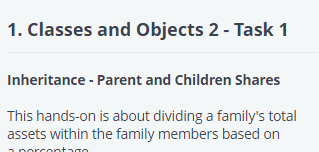
**for** i **in** **range**(0, **len**(str2)):

**if** word.count(str2[i])<3:

            str3.append(str2[i])

**print**(str3[-20 : ])

**print**(word1.rindex(strfind))



*# It is expected to create two child classes 'son' & 'daughter' for the above class 'parent'*

*#*

*#Write your code here*

**class** son(parent):

**def** \_\_init\_\_(**self**,Asset,Percentage\_for\_son):

        parent.\_\_init\_\_(**self**,Asset)

**self**.asset=Asset

**self**.Percentage\_for\_son=Percentage\_for\_son

**def** son\_display(**self**):

**print**("Share of Son is {} Million.".**format**(**round**((**self**.Percentage\_for\_son\***self**.asset))/100))

**class** daughter(parent):

**def** \_\_init\_\_(**self**,Asset,Percentage\_for\_daughter):

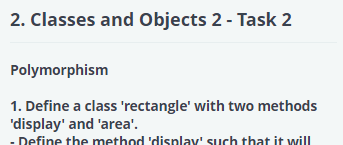
        parent.\_\_init\_\_(**self**,Asset)

**self**.Percentage\_for\_daughter=Percentage\_for\_daughter

**self**.asset=Asset

**def** daughter\_display(**self**):

**print**("Share of Daughter is {} Million.".**format**(**round**((**self**.Percentage\_for\_daughter\***self**.asset))/100))



**class** rectangle:

**def** display(**self**):

**print**("This is a Rectangle")

**def** area(**self**,length,breadth):

        ar=length\*breadth

**print**("Area of Rectangle is  {}".**format**(ar))

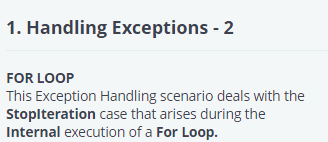
**class** square:

**def** display(**self**):

**print**("This is a Square")

**def** area(**self**,side):

**print**("Area of square is  {}".**format**(side\*side))



**def** FORLoop():

    n=**int**(**input**())

    l1=[]

**for** i **in** **range**(n):

        l1.append(**int**(**input**()))

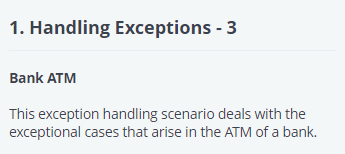
**print**(l1)

    iter1=**iter**(l1)

**for** i **in** **range**(**len**(l1)):

**print**(**next**(iter1))

**return** iter1



**class**   MinimumDepositError(Exception):

**def** \_init\_(**self**,value):

**self**.value=value

**def** \_str\_(**self**):

**return** **str**(**self**.value)

**class**   MinimumBalanceError(Exception):

**def** \_init\_(**self**,value):

**self**.value=value

**def** \_str\_(**self**):

**return** **str**(**self**.value)

**def** Bank\_ATM(balance,choice,amount):

**try**:

**if**(balance<500):

**raise** ValueError('As per the Minimum Balance Policy, Balance must be at least 500')

**if**(choice==1):

**if**(amount<2000):

**raise**  MinimumDepositError('The Minimum amount of Deposit should be 2000.')

**else**:

            balance=balance+amount

**elif**(choice==2):

**if**(balance-amount<500):

**raise** MinimumBalanceError('You cannot withdraw this amount due to Minimum Balance Policy')

**else**:

                balance=balance-amount

**except** ValueError **as** e:

**print**(e)

**except**  MinimumDepositError **as** e:

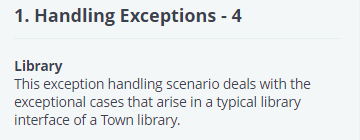
**print**(e)

**except** MinimumBalanceError **as** e:

**print**(e)

**else**:

**print**("Updated Balance Amount:  "+**str**(balance))



**def** Library(memberfee,installment,book):

    amount=0

    l=['philosophers stone','chamber of secrets','prisoner of azkaban','goblet of fire','order of phoenix','half blood prince','deathly hallows 1','deathly hallows 2']

**try**:

**if** installment>3:

**raise** ValueError('Maximum Permitted Number of Installments is 3')

**if** installment==0:

**raise** ZeroDivisionError('Number of Installments cannot be Zero.')

**else**:

**print**("Amount per Installment is ",(memberfee/installment))

**if** book.lower() **not** **in** l:

**raise** NameError('No such book exists in this section')

**else**:

**print**("It is available in this section")

**except** ValueError **as** e:

**print**(e)

**except** ZeroDivisionError **as** e:

**print**(e)

**except** NameError **as** e:

**print**(e)