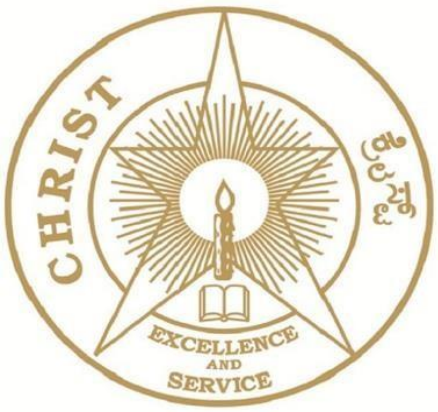


CHRIST

(DEEMED TO BE UNIVERSITY)

BANGALORE · INDIA

MAT 651E: Financial Mathematics Using Python & Excel-

E-Record

Department of Mathematics

-Arnav Sinha, 2040811

FINANCIAL MATHEMATICS- MAT641E

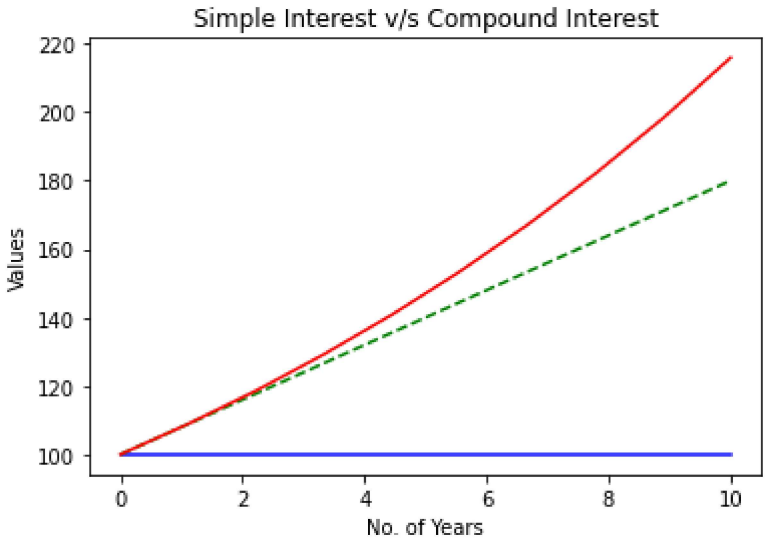
Arnav Sinha, 2040811

FORCE OF INTEREST

Simple interest v/s Compound Interest
SI: $a(t) = (1 + st)$
SI: $A_K(t) = K(1 + st)$
CI: $a(t) + (i + 1)^t$
CI: $A_K(t) = Ka(t)$

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
from pylab import *
K=100
s=0.08
i=s
n=10
t=np.linspace(0,n,n) #values from 0 to n with n divisions
y1=np.ones(len(t))*K #a horizontal line on the graph
y2=K*(1+s*t) #simple interest: a(t) = (1+st); A_K(t) = K(1+st)
y3=K*(1+i)**t #compound interest: a(t) + (i+1)^t; A_K(t) = Ka(t)
xlabel('No. of Years')
ylabel('Values')
plt.title("Simple Interest v/s Compound Interest") #As time proceeds,
plt.plot(t,y1,'b-')
plt.plot(t,y2,'g--')
plt.plot(t,y3,'r-')
plt.show()
```



Nominal rate, effective rate, discount rate

APR - Annual Percentage Rate $i_{nom} = i^{(m)}$

EAR - Effective Annual Rate i

$i^{(m)} = m * ((1+i)^{\frac{1}{m}} - 1)$

$i = (1 + \frac{i^{(m)}}{m})^m - 1$

$d^{(m)} = \frac{i^{(m)}}{1+\frac{i^{(m)}}{m}}$

How to arrive at the formula for $d^{(m)}$

$(1 + \frac{i^{(m)}}{m})(1 + \frac{d^{(m)}}{m}) = 1$

$1 + \frac{i^{(m)}}{m} - \frac{d^{(m)}}{m} - \frac{i^{(m)}d^{(m)}}{m^2} = 1$

$i^{(m)}$ $i^{(m)}$ $i^{(m)}$

In [2]:
```
def i_nom(i,m):
  i_nom = ((1+i)**(1/m) - 1)
  return i_nom
```

In [3]:
```
def d_nom(d,m):
  d_nom = (1-(1-d)**(1/m))
  return d_nom
```

In [4]:
```
i_nom(0.12,12)
```

Out[4]: 0.009488792934583046

In [5]:
```
d = 0.12/(1+0.12)
print(d)
```

```
0.10714285714285712
```

In [6]:
```
import numpy as np
import matplotlib.pyplot as plt
import math
def i_nom(i,m):
    i_nom = (((1+i)**(1/m))-1)*m
    return i_nom
i_nom(0.12,2) #i^(2)
```

Out[6]: 0.11660104885167266

In [7]:
```
X=[1,2,3,4,6,12,24,365,365*24,365*24*60,365*24*60*60]
for x in X:
    y11 = i_nom(0.12,x)
    print(y11)
```

```
0.1200000000000001
0.11660104885167266
0.11549646111066236
0.11494937888832091
0.11440573836312806
0.11386551521499655
0.11359667760742376
0.1133462808142105
0.1133294183803013
0.11332869750475538
0.1133286825663049
```

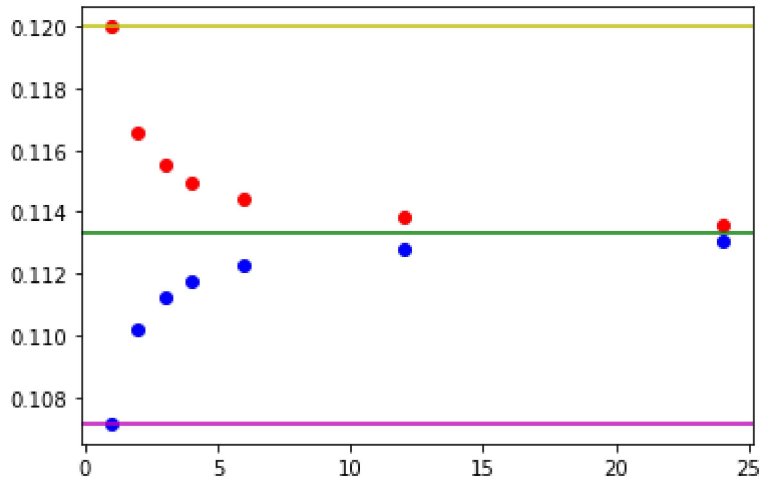
In [8]:

```
from array import *
import numpy as np
X=[1,2,3,4,6,12,24]
y11=[]
y22=[]
y111=[]
y33=[]
for x in X:
    y11.append(i_nom(0.12,x))
print(y11)
y111 = np.divide(y11,X)
y33 = [x + 1 for x in y111]
y22 = np.divide(y11,y33)
print(y22)
```

```
[0.1200000000000001, 0.11660104885167266, 0.11549646111066236, 0.11494937
888832091, 0.11440573836312806, 0.11386551521499655, 0.11359667760742376]
[0.10714286 0.11017763 0.11121482 0.11173832 0.11226511 0.11279522
 0.11306154]
```

$$y_{11} = i^m$$
$$y_{111} = \frac{i^{(m)}}{m}$$
$$y_{33} = 1 + \frac{i^{(m)}}{m}$$
$$y_{22} = \frac{i^{(m)}}{1+\frac{i^{(m)}}{m}}$$

In [9]:
```
import matplotlib.pyplot as plt
import math
delta = math.log(1+0.12)
# delta is defined as lim m->inf i^(m) - limiting value of nominal rat
# lim m->inf i^(m) is the same lim m->inf d^(m)
plt.scatter(X,y11,color='red')
plt.scatter(X,y22,color='blue')
plt.axhline(y=delta, color='g',linestyle='-')
plt.axhline(y=0.12, color='y',linestyle='-')
plt.axhline(y=d, color='m',linestyle='-')
plt.figure(figsize=(50,10))
plt.show()
```




```
<Figure size 3600x720 with 0 Axes>
```

In [10]:
```
import io
import pandas as pd
```

In [ ]:
```
from openpyxl import Workbook
wb = Workbook()
test_filename = 'FOI_comparison.xlsx'
test_filename2='FOI_comparison2.xlsx'
```

In [ ]:
```
wb.save('/content/gdrive/My Drive/Colab Notebooks/FINANCIAL MATH/'+tes
```

In [ ]:
```
from openpyxl import load_workbook
wb1 = load_workbook('/content/gdrive/My Drive/Colab Notebooks/FINANCIA
```

In [ ]:
```
calc_data = ([0.12, 1],[0.12, 2],[0.12, 3],[0.12, 4],[0.12, 6],[0.12,
```

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import math
```

In [ ]:

```
def i_nom(i,m):
    i_nom = (((1+i)**(1/m))-1)*m
    d_nom = i_nom / (1 + (i_nom/m))
    i_eff = i
    d_eff = i/(1+i)
    delta = math.log(1+i)
    return i_nom, d_nom, i_eff, d_eff, delta
```

In [ ]:

```
from array import *
import numpy as np
X = (np.transpose(calc_data))[1]
y11=[]
y22=[]
d=[]
delta=[]
i=[]

for x in X:
  d.append(i_nom(0.12,x)[3])
  y22.append(i_nom(0.12,x)[1])
  delta.append(i_nom(0.12,x)[4])
  y11.append(i_nom(0.12,x)[0])
  i.append(i_nom(0.12,x)[2])

print(d)
print(y22)
print(delta)
print(y11)
print(i)

```

In [ ]:

```
data={'d':d,'d(m)':y22,'delta':delta,'i(m)':y11,'i':i}
df=pd.DataFrame(data)
writer=pd.ExcelWriter('/content/gdrive/My Drive/Colab Notebooks/FINANC
df.to_excel(writer,sheet_name='Sheet1')
wb=writer.book
ws=writer.sheets['Sheet1']

from openpyxl.chart import (LineChart,Reference)
chart=LineChart()
values=Reference(ws,min_col=2,min_row=1,max_col=6,max_row=8)
chart.add_data(values,titles_from_data=True)

chart.y_axis.scaling.min=0.1
chart.y_axis.scaling.max=0.125

s1=chart.series[0]
s1.marker.symbol="diamond"
s1.marker.graphicalProperties.solidFill="di21b9"

s2=chart.series[1]
s2.marker.symbol="star"
s1.marker.graphicalProperties.solidFill="249e19"

s3=chart.series[2]
s3.marker.symbol="circle"
s1.marker.graphicalProperties.solidFill="4846db"

s4=chart.series[3]
s4.marker.symbol="triangle"
s1.marker.graphicalProperties.solidFill="bf3348"

s5=chart.series[4]
s5.marker.symbol="square"
s1.marker.graphicalProperties.solidFill="f4c92e"

ws.add_chart(chart,"A11")
writer.save()
```

SIMPLE & COMPOUND INTEREST

In [13]:
```
def MA(K,r,t,type):
  if type=='simple':
    s=K*(1+r*t)
    return s
  elif type=='compound':
    c=K*(1+r)**t
    return c
  else:
    return 'enter a valid interest type'

MA(1000,0.08,3,'compound')
```

Out[13]: 1259.7120000000002

LEVEL ANNUITIES

In [14]:
```
def a(i):
  a_i=(1+i)
  return a_i
```

In [15]:
```
def v(i):
  v_i=1/(1+i)
  return v_i
```

In [16]:
```
a(0.12)
```

Out[16]: 1.12

In [17]:
```
v(0.12)
```

Out[17]: 0.8928571428571428

In [18]:
```
## Annuity immediate present worth without multipledispatch
def ani_ai(i,n,Q):
  ani=Q*((1-(1+i)**(-n))/i)
  return ani
```

In [19]:
```
ani_ai(0.0025,24,189.12)
```

Out[19]: 4400.062057113476

In [20]:
```
## Annuity immediate present worth without multipledispatch
def sni_ai(i,n,Q):
  sni=Q*(((a(i))**n - 1)/i)
  return sni
```

In [21]:
```
sni_ai(4/996,12,100)
```

Out[21]: 1226.8640841048723

In [ ]:
```
!pip install multipledispatch
```

In [22]:
```
from multipledispatch import dispatch
```

ANNUITY IMMEDIATE

PRESENT WORTH

In [23]:
```
@dispatch(float,int,float)
def AIPW(i,n,Q):
  ani=ani_ai(i,n,Q)
  return ani
```

In [24]:
```
AIPW(0.0025,24,189.12)
```

Out[24]: 4400.062057113476

In [25]:
```
@dispatch(float,int,float,str)
def AIPW(d,n,Q,dtype):
  i=d/(1-d)
  ani=ani_ai(i,n,Q)
  return ani
```

In [26]:
```
AIPW(0.00250625,24,189.12,'discount')
```

Out[26]: 4399.380341332246

FUTURE WORTH

In [27]:
```
@dispatch(float,int,float)
def AIFW(i,n,Q):
  sni=sni_ai(i,n,Q)
  return sni
```

In [28]:
```
AIFW(4/996,12,100.00)
```

Out[28]: 1226.8640841048723

In [29]:
```
@dispatch(float,int,float,str)
def AIFW(d,n,Q,dtype):
  if dtype == 'discount':
    i=d/(1-d)
    sni=sni_ai(i,n,Q)
    return sni
  else:
    print("Error")

```

In [30]:
```
AIFW(4/996,12,100.00,'discount')
```

Out[30]: 1226.9738698389751

ANNUITY DUE

PRESENT WORTH

In [31]:
```
@dispatch(float,int,float)
def ADPW(i,n,Q):
  Ani=ani_ai(i,n,Q)*(1+i)
  return Ani
```

In [32]:
```
ADPW(0.04,30,100.0)
```

Out[32]: 1798.371463269107

In [33]:
```
@dispatch(float,int,float,str) #if discount is given instead of intere
def ADPW(d,n,Q,dtype):
  if dtype == 'discount':
    i=d/(1-d)
    Ani=ani_ai(i,n,Q)*(1+i)
    return Ani
  else:
    print("Error")
```

In [34]:
```
ADPW(0.03846153846,30,100.0,'discount')
```

Out[34]: 1798.3714633025636

FUTURE WORTH

In [35]:
```
@dispatch(float,int,float)
def ADFW(i,n,Q):
  Sni=sni_ai(i,n,Q)*(1+i)
  return Sni
```

In [36]:
```
ADFW(0.03,24,80.0)
```

Out[36]: 2836.7411457445764

In [37]:
```
@dispatch(float,int,float,str)
def ADFW(d,n,Q,dtype):
  if dtype == 'discount':
    i=d/(1-d)
    Sni=sni_ai(i,n,Q)*(1+i)
    return Sni
  else:
    print("Error")
```

In [38]:
```
ADFW(0.02912621359,24,80.0,'discount')
```

Out[38]: 2836.741145653856

DEFERRED ANNUITIES

In [39]:
```
@dispatch(float,int,int,float,str)
def DeferredAnnuityPresentWorth(i,n,k,Q,dtype):
  if dtype == 'deferred':
    DAPW = Q*(1+i)**k*(1-(1+i)**(-(n-k)))/i
    return DAPW
  else:
    print("Error")
```

In [40]:
```
DeferredAnnuityPresentWorth(0.03,30,12,100.0,'deferred')
```

Out[40]: 1960.9221005417508

In [41]:
```
@dispatch(str, float, int, float, float, type)
def AnnuityImmediatePresentWorth(d,n,k,Q,dtype):
    if dtype == 'deferred':
      i=1/(1-d)
      P = Q * (1+i)**k * ((1 - (1+i)**(-(n-k))) / i)
      return P
    else:
        raise ValueError('Invalid input parameters')
```

In [42]:
```
DeferredAnnuityPresentWorth(0.02912621359,30,12,100.0,'deferred')
```

Out[42]: 1955.4699970141646

PERPETUITIES

In [43]:
```
@dispatch(str, float, float)
def AnnuityImmediatePresentWorth(d_type, i, Q):
    if d_type == 'infinite payment':
        return Q / i
    else:
        raise ValueError('Invalid input parameters')
```

DEFERRED INFINITE PAYMENTS

In [44]:
```
@dispatch(str, str, float, int, float)
def AnnuityImmediatePresentWorth(d_type, inf_type, i, Q, k):
    if d_type == 'deferred' and inf_type == 'infinite':
        return Q * ((1 - (1+i)**(-k)) / i)
    else:
        raise ValueError('Invalid input parameters')
```

OUTSTANDING LOAN BALANCES

Retrospective Method

1. Let L be the loan amount borrowed at time zero.
2. Let n be number of periods for which loan is borrowed.
3. Let k be the number of periods such that $k < n$.
4. Maturity of the loan amount L at the end of k periods is $L(1+i)^k$.
5. Let Q be the periodic equal payments done to repay the loan such that it is paid at the end of each period.
6. Accumulation of payments of Q's after k periods is $Qs_{\overline{k}|_i}$.
7. So, OLB_k (outstanding loan balance after k periods) is $L(1+i)^k$ - $Qs_{\overline{k}|_i}$.

Prospective Method

1. Let L be the loan amount borrowed at time zero.
2. Let n be number of periods for which loan is borrowed.
3. Let k be the number of periods such that $k < n$.
4. Maturity of the loan amount L at the end of k periods is $L(1+i)^k$.
5. Let Q be the periodic equal payments done to repay the loan such that it is paid at the end of each period.
6. Immediately after completing k periods, Worth of the payments Q's, paid after the k^{th} period is $Qa_{\overline{n-k}|_i}$.

7. So, OLB_k (outstanding loan balance after k periods) is $Qa_{\overline{n-k}|_i}$.
8. For if the last payment is not leveled with Q and is different say R, then OLB_k (outstanding loan balance after k periods) is $Qa_{\overline{n-k-1}|_i} + R(1+i)^{-(n-k)}$.

In [ ]:
```
!pip install multipledispatch
```

In [45]:
```
from multipledispatch import dispatch
```

In [46]:
```
@dispatch(float, int)
def a_i(i_eff, k_periods):
  a = (1 + i_eff)**k_periods
  return a
```

In [47]:
```
@dispatch(float, int)
def v_i(i_eff, k_periods):
  v = (1 + i_eff)**(-k_periods)
  return v

```

In [48]:
```
def a_d(d_eff, k_periods):
  i_eff = d_eff / (1- d_eff)
  a = (1 + i_eff)**k_periods
  return a, i_eff

```

In [49]:
```
def v_d(d_eff, k_periods):
  i_eff = d_eff / (1- d_eff)
  v = (1 + i_eff)**{-k_periods}
  return v, i_eff

```

In [50]:
```
@dispatch(float, int)
def sni_ai(i_eff, k_periods):
  sni = (a_i(i_eff, k_periods)-1)/i_eff
  return sni

```

In [51]:
```
@dispatch(float, int, str)
def sni_ai(d_eff, k_periods, discount):
  if discount == "d":
    ad = a_d(d_eff, k_periods)
  else:
    print("Error")
  sni = (ad[0]-1)/ad[1]
  return sni
```

In [52]:
```
@dispatch(float, int)
def ani_ai(i_eff, n_periods):
  ani = (1 - v_i(i_eff, n_periods))/i_eff
  return ani

```

In [53]:
```
@dispatch(float, int, str)
def ani_ai(d_eff, n_periods, discount):
  if discount == "d":
    vd = v_d(d_eff, n_periods)
  else:
    print("Error")
  ani = (1 - vd[0] ) / vd[1]
  return ani
```

Sasha is obligated to repay a loan made on the first of the month by paying Rs 80 at the end of each of the next thirty months, including this month. The effective monthly rate on the loan is 0.4%. Using a user-defined Python function, find Sasha's loan balance immediately after her twelfth payment.

In [54]:
```
@dispatch(float,int, float,int) #prospective
def OLBk1(i_eff,n_periods,Q,k_periods):
  OLBk1 = Q*ani_ai(i_eff,n_periods-k_periods)
  return OLBk1
```

In [55]:
```
OLBk1(0.004,30,80.0,12)
```

Out[55]: 1386.709087962752

A loan of Rs 20000 is being repaid by payments of Rs 2500 at the end of each year and a final smaller payment one year after the last Rs 2500 payment. The annual effective interest rate on the loan is 8%. Using a user defined python function, find the outstanding loan balance just after the borrower has made payments totalling Rs 15000.

15000 is paid by the end of 6 periods

In [56]:
```
@dispatch(float,float,int,int)
def OLBk2(i_eff,Q,k_periods,L):
  OLBk2= L*(1+i_eff)**k_periods - Q*sni_ai(i_eff,k_periods)
  return OLBk2
```

In [57]:
```
OLBk2(0.08,2500.0,6,20000)
```

Out[57]: 13397.663866879993

Mr and Mrs Harper purchased a home for Rs 256000. They make a down payment of Rs 40000 and finance the remainder of the purchase price with a thirty-year mortgage at an annual effective interest rate of 6.5%. The Harpers sell their home at the end of eight years, just after having made their ninety-sixth end-of-month mortgage payment. The sales price is Rs 282000,

and the Harpers closing costs are 3% of the selling price. The outstanding loan balance is deducted from the amount the Harpers receive and sent to the lender. How large a check do the Harpers receive, and how much interest did they pay over the eight years? (Use Python)

Loan taken = Home cost - downpayment = 256000 - 40000 = 216000

L = 216000

Monthly rate = i(12)/12 = (1+0.065)^(1/12) - 1 = 0.005261694

Q = 1338.96

k = 96

In [58]:
```
@dispatch(float,int, float,int) #prospective
def OLBk3(i_eff,n_periods,Q,k_periods):
  OLBk3 = Q*ani_ai(i_eff,n_periods-k_periods)
  return OLBk3
```

In [59]:
```
OLBk3(0.005261694,360,1338.96,96)
```

Out[59]: 190800.8589167266

Check received = 216000 - 190800.8589167266 = 25199.76

Interest paid = 1338.96*96 - 25199.76 = 103340.40

ANNUITIES IN GEOMETRIC PROGRESSION

In [ ]:
```
!pip install multipledispatch
```

In [60]:
```
from multipledispatch import dispatch
```

In [61]:
```
def a_i(i_eff,n_periods):
  a=(1+i_eff)**(-n_periods)
  return a
```

In [62]:
```
def v_i(i_eff,n_periods):
  v=(1+i_eff)**(-n_periods)
  return v
```

In [63]:
```
def a_d(d_eff,n_periods):
  i_eff=d_eff/(1-d_eff)
  a=(1+i_eff)**n_periods
  return a, i_eff
```

In [64]:
```
def v_d(d_eff,n_periods):
  i_eff=d_eff/(1-d_eff)
  v=(1+i_eff)**(-n_periods)
  return v, i_eff
```

In [65]:
```
def j_eff(i_eff,gp_percentage):
  j_eff=(i_eff - gp_percentage)/(1 + gp_percentage)
  return j_eff
```

In [66]:
```
j_eff(0.07,0.03)
```

Out[66]: 0.03883495145631068

In [67]:
```
def gpv_i(i_eff,gp_percentage):
  v_j = (1+ gp_percentage)/(1+i_eff)
  return v_j
```

In [68]:
```
@dispatch(float, int)
def ani_ai(i_eff, n_periods):
  ani = (1 - v_i(i_eff, n_periods))/i_eff
  return ani
```

In [69]:
```
@dispatch(float, int)
def ani_ad(i_eff, n_periods):
  an_due = ( 1 + i_eff ) * ani_ai(i_eff, n_periods)
  return an_due
```

In [70]:
```
@dispatch(float, int, str)
def ani_ai(d_eff, n_periods, discount):
  if discount == "d":
    vd = v_d(d_eff, n_periods)
  else:
    print("Error")
  ani = (1 - vd[0] ) / vd[1]
  return ani
```

In [71]:
```
@dispatch(float, int, str)
def ani_ad(d_eff, n_periods, discount):
  if discount == "d":
    vd=v_d(d_eff,n_periods)
  else:
    print("Error")
  an_due = (1+vd[1])*ani_ai(d_eff,n_periods,discount)
  return an_due
```

In [72]:
```
@dispatch(float, int, float, float)
def an_gp_pw(i_eff, n_periods, P, gp_percentage):
  jeff = j_eff(i_eff, gp_percentage)
  an_gp_pw = P*((1 + i_eff)**(-1))*ani_ad(jeff, n_periods)
  return an_gp_pw
```

In [73]:
```
@dispatch(float, int, float, float, str)
def an_gp_pw(d_eff, n_periods, P, gp_percentage, discount):
  if discount == "d":
    vd = v_d(d_eff, n_periods)
    i_eff = vd[1]
    j_efff = j_eff(i_eff, gp_percentage)
    an_gp_pw = P*((1 + i_eff)**(-1))*ani_ad(j_efff, n_periods)
  else:
    print("Error")
  return an_gp_pw
```

In [74]:
```
an_gp_pw(0.07, 25, 800.0, 0.03)
```

Out[74]: 12284.462770939861

In [75]:
```
0.07/1.07
```

Out[75]: 0.06542056074766356

In [76]:
```
an_gp_pw(0.06542056074766356,25,800.00,0.03,"d")
```

Out[76]: 12284.462770939861

ANNUITIES IN AP

In [ ]:
```
!pip install multipledispatch
```

In [77]:
```
from multipledispatch import dispatch

```

In [78]:
```


def a_i(i_eff, n_periods):
  a = (1 + i_eff)**n_periods
  return a

```

In [79]:
```
def v_i(i_eff, n_periods):
  v = (1 + i_eff)**(-n_periods)
  return v

```

In [80]:
```
def a_d(d_eff, n_periods):
  i_eff = d_eff / (1- d_eff)
  a = (1 + i_eff)**n_periods
  return a, i_eff

```

In [81]:
```
def v_d(d_eff, n_periods):
  i_eff = d_eff / (1- d_eff)
  v = (1 + i_eff)**(-n_periods)
  return v, i_eff

```

In [82]:
```
@dispatch(float, int)
def ani_ai(i_eff, n_periods):
  ani = (1 - v_i(i_eff, n_periods))/i_eff
  return ani

```

In [83]:
```
@dispatch(float, int, str)
def ani_ai(d_eff, n_periods, discount):
  if discount == "d":
    vd = v_d(d_eff, n_periods)
  else:
    print("Error")
  ani = (1 - vd[0] ) / vd[1]
  return ani

```

In [84]:
```
@dispatch(float, int)
def sni_ai(i_eff, n_periods):
  sni = ani_ai(i_eff, n_periods) * a_i(i_eff, n_periods)
  return sni

```

In [85]:
```
@dispatch(float, int, str)
def sni_ai(d_eff, n_periods, discount):
  sni = ani_ai(d_eff, n_periods, discount) * a_i(i_eff, n_periods)
  return sni


```

In [86]:
```
@dispatch(float, int, float, float)
def ia_PQ_sni(i_eff, n_periods, P, Q):
  ia_sni = P * sni_ai(i_eff, n_periods) + (Q/i_eff)*(sni_ai(i_eff, n_p
  return ia_sni
```

In [87]:
```
@dispatch(float, int, float, float)
def ia_PQ_ani(i_eff, n_periods, P, Q):
  ia_ani = P * ani_ai(i_eff, n_periods) + (Q/i_eff)*(ani_ai(i_eff, n_p
  return ia_ani
```

In [88]:
```
@dispatch(float, int, float, float, str)
def ia_PQ_sni(d_eff, n_periods, P, Q, discount):
  if discount == "d":
    vd = v_d(d_eff, n_periods)
    i_eff = vd[1]
    ia_snd = P * sni_ai(i_eff, n_periods) + (Q/i_eff)*(sni_ai(i_eff, r
  else:
    print("Error")
  return ia_snd
```

In [89]:
```
@dispatch(float, int, float, float, str)
def ia_PQ_ani(d_eff, n_periods, P, Q, discount):
  if discount == "d":
    vd = v_d(d_eff, n_periods)
    i_eff = vd[1]
    ia_and = P * ani_ai(i_eff, n_periods) + (Q/i_eff)*(ani_ai(i_eff, r
  else:
    print("Error")
  return ia_and
```

In [90]:
```
@dispatch(float, int)
def iaf_ani(i_eff, n_periods):
  iaf = ia_PQ_ani(i_eff, n_periods, 1.0, 1.0)
  return iaf
```

In [91]:
```
ia_PQ_ani(0.042,17,500.00,500.00)
```

Out[91]: 48039.74546871386

In [93]:
```
iaf_ani(0.042,17)
```

Out[93]: 96.0794909374277

In [94]:
```
500*iaf_ani(0.042,17)
```

Out[94]: 48039.74546871385

In [95]:
```
@dispatch(float, int, str)
def iaf_ani(d_eff, n_periods, discount):
  iaf = ia_PQ_ani(d_eff, n_periods, 1.0, 1.0, discount)
  return iaf

```

In [96]:
```
@dispatch(float, int)
def iaf_sni(i_eff, n_periods):
  iaf_fv = ia_PQ_sni(i_eff, n_periods, 1.0, 1.0)
  return iaf_fv

```

In [97]:
```
@dispatch(float, int, str)
def iaf_sni(d_eff, n_periods, discount):
  iaf_fv = ia_PQ_sni(d_eff, n_periods, 1.0, 1.0, discount)
  return iaf_fv
```

In [98]:
```
@dispatch(float, int)
def sni_ad(i_eff, n_periods):
  sniad = (ani_ai(i_eff, n_periods) * a_i(i_eff, n_periods)) * (1+i_ef
  return sniad
```

In [99]:
```
sni_ad(0.042,17)
```

Out[99]: 25.121405703368044

In [100]:
```
@dispatch(float, int)
def ani_ad(i_eff,n_periods):
  aniad = (ani_ai(i_eff, n_periods)) * (1+i_eff)
  return aniad
```

In [101]:
```
@dispatch(float,int,str)
def sni_ad(d_eff, n_periods, discount):
  i_eff = d_eff / (1- d_eff)
  sniad = (sni_ai(i_eff, n_periods)) * ((1+i_eff))
  return sniad

```

In [102]:
```
sni_ad(0.04030710172744722,17,"d")

```

Out[102]: 25.121405703368044

In [103]:
```
@dispatch(float, int, str)
def ani_ad(d_eff, n_periods, discount):
  i_eff = d_eff / (1- d_eff)
  aniad = (ani_ai(i_eff, n_periods)) * ((1+i_eff))
  return aniad

```

In [104]:
```
ani_ad(0.04030710172744722,17,"d")
```

Out[104]: 12.48224535386423

In [105]:
```
from multipledispatch import dispatch
```

In [106]:
```
import math
```

In [107]:
```
def a_d(d_eff, n_periods):
  i_eff = d_eff / (1- d_eff)
  a = (1 + i_eff)**n_periods
  return a, i_eff


```

In [108]:
```
def v_d(d_eff, n_periods):
  i_eff = d_eff / (1- d_eff)
  v = (1 + i_eff)**(-n_periods)
  return v, i_eff
```

In [109]:
```
def foi(i_eff):
  foi_0 = math.log(1+i_eff)
  return foi_0

```

In [110]:
```
@dispatch(float, int)
def ani_bar(i_eff, n_periods):
  foi_1 = foi(i_eff)
  ani_bar = (1 - math.exp(-foi_1*n_periods))/foi_1
  return ani_bar


```

In [111]:
```
@dispatch(float, int)
def sni_bar(i_eff, n_periods):
  foi_2 = foi(i_eff)
  sni_bar = (math.exp(foi_2*n_periods)-1)/foi_2
  return sni_bar

```

In [112]:
```
@dispatch(float, int, str)
def ani_bar(d_eff, n_periods, discount):
  if discount == "d":
    vd = v_d(d_eff, n_periods)
  else:
    print("Error")
  i_eff = vd[1]
  anibar = ani_bar(i_eff, n_periods)
  return anibar
```

In [113]:
```
@dispatch(float, int, str)
def sni_bar(d_eff, n_periods, discount):
  if discount == "d":
    vd = v_d(d_eff, n_periods)
  else:
    print("Error")
  i_eff = vd[1]
  snibar = sni_bar(i_eff, n_periods)
  return snibar

```

In [114]:
```
@dispatch(float, int)
def Iani_bar(i_eff, n_periods):
  foi_1 = foi(i_eff)
  Iani_bar = (ani_bar(i_eff, n_periods) - (n_periods * (1+ i_eff)**(-r
  return ani_bar

```

In [115]:
```
@dispatch(float, int)
def Isni_bar(i_eff, n_periods):
  foi_1 = foi(i_eff)
  Isni_bar = (sni_bar(i_eff, n_periods) - n_periods)/foi_1
  return Isni_bar

```

In [116]:
```
@dispatch(float, int, str)
def ani_bar(d_eff, n_periods, discount):
  if discount == "d":
    vd = v_d(d_eff, n_periods)
  else:
    print("Error")
  i_eff = vd[1]
  Ianibar = Iani_bar(i_eff, n_periods)
  return Ianibar

```

In [117]:
```
@dispatch(float, int)
def Isni_bar(i_eff, n_periods):
  foi_1 = foi(i_eff)
  Isni_bar = (sni_bar(i_eff, n_periods) - n_periods)/foi_1
  return Isni_bar

```

In [118]:
```
@dispatch(float, int, str)
def Iani_bar(d_eff, n_periods, discount):
  if discount == "d":
    vd = v_d(d_eff, n_periods)
  else:
    print("Error")
  i_eff = vd[1]
  Ianibar = Iani_bar(i_eff, n_periods)
  return Ianibar

```

In [119]:
```
@dispatch(float, int, str)
def Isni_bar(d_eff, n_periods, discount):
  if discount == "d":
    vd = v_d(d_eff, n_periods)
  else:
    print("Error")
  i_eff = vd[1]
  Isnibar = Isni_bar(i_eff, n_periods)
  return Isnibar

```

AMMORTIZATION- LEVEL PAYMENTS

In [122]:
```
!pip install amortization
```

```
Collecting amortization
  Downloading amortization-2.2.1-py3-none-any.whl (6.7 kB)
Collecting tabulate<0.10.0,>=0.9.0
  Downloading tabulate-0.9.0-py3-none-any.whl (35 kB)
Installing collected packages: tabulate, amortization
Successfully installed amortization-2.2.1 tabulate-0.9.0
```

In [120]:
```
1 import pandas as pd
2 import numpy as np
```

In [123]:
```
1 from amortization.schedule import amortization_schedule
```

In [124]:
```
1 table = amortization_schedule(200000, 0.15, 20)
```

In [125]:
```
1 df1 = pd.DataFrame(table,columns=['TIME','AMOUNT','INTEREST','PRINCIPA
```

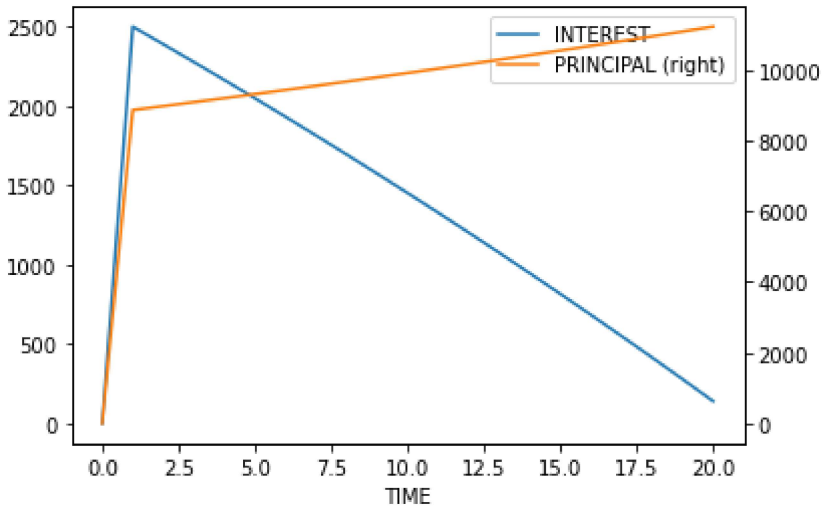
In [126]:
```
1 new_row = pd.DataFrame({'TIME':0,'AMOUNT':0,'INTEREST':0,'PRINCIPAL':0
2 df1=pd.concat([new_row,df1]).reset_index(drop=True)
```

In [127]:
```
1 display(df1.round(2))
```

	TIME	AMOUNT	INTEREST	PRINCIPAL	BALANCE
0	0	0.00	0.00	0.00	200000.00
1	1	11364.08	2500.00	8864.08	191135.92
2	2	11364.08	2389.20	8974.88	182161.04
3	3	11364.08	2277.01	9087.07	173073.97
4	4	11364.08	2163.42	9200.66	163873.31
5	5	11364.08	2048.42	9315.66	154557.65
6	6	11364.08	1931.97	9432.11	145125.54
7	7	11364.08	1814.07	9550.01	135575.53
8	8	11364.08	1694.69	9669.39	125906.14
9	9	11364.08	1573.83	9790.25	116115.89
10	10	11364.08	1451.45	9912.63	106203.26
11	11	11364.08	1327.54	10036.54	96166.72
12	12	11364.08	1202.08	10162.00	86004.72
13	13	11364.08	1075.06	10289.02	75715.70
14	14	11364.08	946.45	10417.63	65298.07
15	15	11364.08	816.23	10547.85	54750.22
16	16	11364.08	684.38	10679.70	44070.52
17	17	11364.08	550.88	10813.20	33257.32
18	18	11364.08	415.72	10948.36	22308.96
19	19	11364.08	278.86	11085.22	11223.74
20	20	11364.04	140.30	11223.74	0.00

In [128]:
```
import matplotlib.pyplot as plt #import relevant module
fig, ax=plt.subplots() #Create figure and axes objects
df1.plot(x="TIME",y='INTEREST',ax=ax,kind="line")
df1.plot(x="TIME",y='PRINCIPAL',ax=ax,kind="line",secondary_y=True)
```

Out[128]: <AxesSubplot:label='db68e2bf-6503-4efb-b221-42c952e93aef'>



BOND AMMORTIZATION

Basic Price Formula

P = $(Fr)a_{\overline{n}|_j} + Cv_j^n$

where

P = Price at issue
F = Face value of the bond
C = Redemption amount
j = effective yield rate for the coupon period
r = coupon rate per coupon period
Fr = amount of each coupon \

Premium-Discount formula \

P = $C(g-j)a_{\overline{n}|_j} + C$

where

P = Price at issue
C = Redemption amount
j = effective yield rate for the coupon period
g = modified coupon rate \

Base amount formula

P = $(C - G)v_j^n + G$

where

P = Price at issue
C = Redemption amount
G = Base amount
j = effective yield rate for the coupon period \

Makeham's formula \

P = $\frac{g}{j}(C - K) + K$

where

P = Price at issue
C = Redemption amount
G = Base amount j = effective yield rate for the coupon period
g = modified coupon rate
K = present value of the redemption amount C

In [129]:
```
import numpy as np
import pandas as pd
from multipledispatch import dispatch
```

In [130]:
```
def v_j(j_eff, n_periods):
  v = (1 + j_eff)**(-n_periods)
  return v
```

In [131]:
```
def ani_j(j_eff,n_periods):
  ani = ((1-v_j(j_eff,n_periods))/(j_eff))
  return ani
```

In [132]:
```
@dispatch(float, int, float, float)
def BP_issue(C, n, j, g):
  K = C * v_j(j, n)
  bp = (g/j) * (C - K) + K #Makeham's formula
  return bp
```

In [133]:
```
@dispatch(float, int, float, float, float)
def BP_issue(C, n, j, F, r):
  K = C * v_j(j, n)
  ani = ani_j(j,n)
  cv = F * r
  bp = cv * ani + K
  return bp
```

In [134]:
```
def B_t(C, n, j, g, t):
  Bt = BP_issue(C, n-t, j, g)
  return Bt
```

In [135]:
```
def I_t(C, n, j, g, t):
  It=(j*B_t(C, n, j, g, t-1))
  return It
```

In [136]:
```
def P_t(C, n, j, g, t):
  Pt=((C*(g-j))*v_j(j, n-t+1))
  return Pt
```

In [137]:
```
B_t(100000.00, 8, 0.03, 0.04, 6)
```

Out[137]: 101913.46969554153

In [138]:
```
I_t(100000.00, 8, 0.03, 0.04, 6)
```

Out[138]: 3084.8583406468406

In [139]:
```
P_t(100000.00, 8, 0.03, 0.04, 6)
```

Out[139]: 915.1416593531598

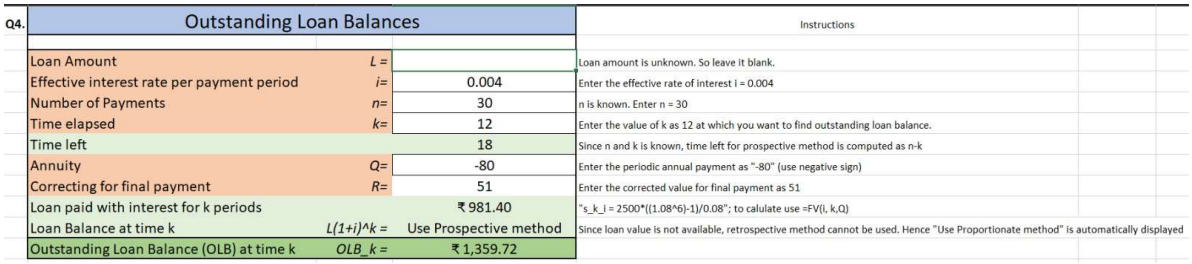
In [140]:
```
def bondamortization(C, n, j, g):
  btime = []
  binterest = []
  bpayment = []
  bbalance = []
  bcp = []
  for i in range(n+1):
    btime.append(i)
    binterest.append(I_t(C, n, j, g, i))
    bpayment.append(P_t(C, n, j, g, i))
    bbalance.append(B_t(C, n, j, g, i))
    bcp.append(C*g)
  binterest[0]=0
  bpayment[0]=0
  bcp[0] = 0
  data = {"Time": np.array(btime),  "Coupon Payment": np.array(bcp), '
  df = pd.DataFrame(data)
  dff = np.round(df, decimals = 2)
  return dff
```

In [141]:
```
1 bondamortization(100000.00, 8, 0.03, 0.04)
```

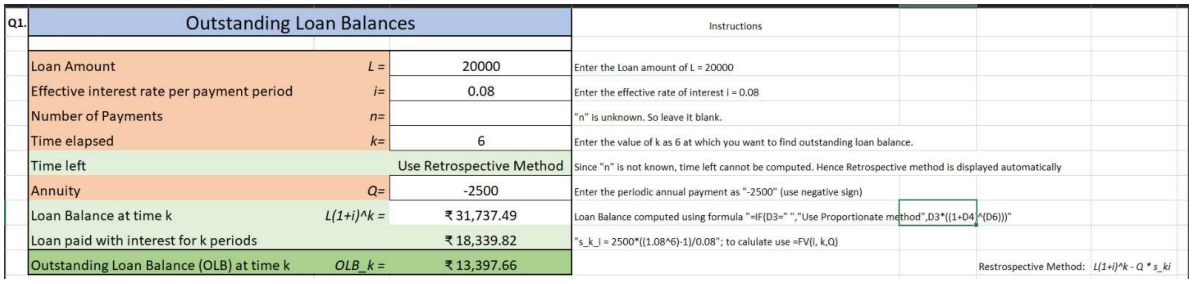
Out[141]:

	Time	Coupon Payment	It	Pt	Bt
0	0	0.0	0.00	0.00	107019.69
1	1	4000.0	3210.59	789.41	106230.28
2	2	4000.0	3186.91	813.09	105417.19
3	3	4000.0	3162.52	837.48	104579.71
4	4	4000.0	3137.39	862.61	103717.10
5	5	4000.0	3111.51	888.49	102828.61
6	6	4000.0	3084.86	915.14	101913.47
7	7	4000.0	3057.40	942.60	100970.87
8	8	4000.0	3029.13	970.87	100000.00

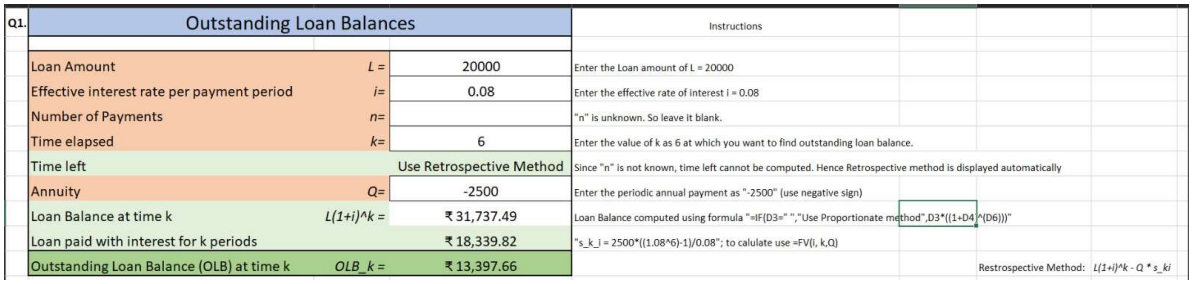
EXCEL



Q4.	Outstanding Loan Balances			Instructions
Loan Amount		*L* =		Loan amount is unknown. So leave it blank.
Effective interest rate per payment period		*i*=	0.004	Enter the effective rate of interest i = 0.004
Number of Payments		*n*=	30	n is known. Enter n = 30
Time elapsed		*k*=	12	Enter the value of k as 12 at which you want to find outstanding loan balance.
Time left			18	Since n and k is known, time left for prospective method is computed as n-k
Annuity		*Q*=	-80	Enter the periodic annual payment as "-80" (use negative sign)
Correcting for final payment		*R*=	51	Enter the corrected value for final payment as 51
Loan paid with interest for k periods			₹ 981.40	"s_k_i = 2500*((1.08^6)-1)/0.08"; to calulate use =FV(i, k,Q)
Loan Balance at time k		*L(1+i)^k* =	Use Prospective method	Since loan value is not available, retrospective method cannot be used. Hence "Use Proportionate method" is automatically displayed
Outstanding Loan Balance (OLB) at time k		*OLB_k* =	₹ 1,359.72	



Q1.	Outstanding Loan Balances			Instructions
Loan Amount		*L* =	20000	Enter the Loan amount of L = 20000
Effective interest rate per payment period		*i*=	0.08	Enter the effective rate of interest i = 0.08
Number of Payments		*n*=		"n" is unknown. So leave it blank.
Time elapsed		*k*=	6	Enter the value of k as 6 at which you want to find outstanding loan balance.
Time left			Use Retrospective Method	Since "n" is not known, time left cannot be computed. Hence Retrospective method is displayed automatically
Annuity		*Q*=	-2500	Enter the periodic annual payment as "-2500" (use negative sign)
Loan Balance at time k		*L(1+i)^k* =	₹ 31,737.49	Loan Balance computed using formula "=IF(D3=" ","Use Proportionate method",D3*((1+D4^(D6)))"
Loan paid with interest for k periods			₹ 18,339.82	"s_k_i = 2500*((1.08^6)-1)/0.08"; to calulate use =FV(i, k,Q)
Outstanding Loan Balance (OLB) at time k		*OLB_k* =	₹ 13,397.66	Restrospective Method: *L(1+i)^k - Q * s_ki*



Q1.	Outstanding Loan Balances			Instructions
Loan Amount		*L* =	20000	Enter the Loan amount of L = 20000
Effective interest rate per payment period		*i*=	0.08	Enter the effective rate of interest i = 0.08
Number of Payments		*n*=		"n" is unknown. So leave it blank.
Time elapsed		*k*=	6	Enter the value of k as 6 at which you want to find outstanding loan balance.
Time left			Use Retrospective Method	Since "n" is not known, time left cannot be computed. Hence Retrospective method is displayed automatically
Annuity		*Q*=	-2500	Enter the periodic annual payment as "-2500" (use negative sign)
Loan Balance at time k		*L(1+i)^k* =	₹ 31,737.49	Loan Balance computed using formula "=IF(D3=" ","Use Proportionate method",D3*((1+D4^(D6)))"
Loan paid with interest for k periods			₹ 18,339.82	"s_k_i = 2500*((1.08^6)-1)/0.08"; to calulate use =FV(i, k,Q)
Outstanding Loan Balance (OLB) at time k		*OLB_k* =	₹ 13,397.66	Restrospective Method: *L(1+i)^k - Q * s_ki*

Q1.	Outstanding Loan Balances			Instructions
	Loan Amount	L =	20000	Enter the Loan amount of L = 20000
	Effective interest rate per payment period	i=	0.08	Enter the effective rate of interest i = 0.08
	Number of Payments	n=		"n" is unknown. So leave it blank.
	Time elapsed	k=	6	Enter the value of k as 6 at which you want to find outstanding loan balance.
	Time left		Use Retrospective Method	Since "n" is not known, time left cannot be computed. Hence Retrospective method is displayed automatically
	Annuity	Q=	-2500	Enter the periodic annual payment as "-2500" (use negative sign)
	Loan Balance at time k	$L(1+i)^k$ =	₹ 31,737.49	Loan Balance computed using formula "=IF(D3=" ","Use Proportionate method",D3*((1+D4)^(D6)))"
	Loan paid with interest for k periods		₹ 18,339.82	"s_k_i = 2500*((1.08^6)-1)/0.08"; to calulate use =FV(i, k,Q)
	Outstanding Loan Balance (OLB) at time k	OLB_k =	₹ 13,397.66	Restrospective Method: $L(1+i)^k - Q * s\_ki$

Type of Annuity	Find the following:	effective interest rate	Periodic equal payment	Number of Payments	effective discount rate	deferred period	Perpetuity (yes / no) (Use 1/i or Q/i)	**Whereever possible use inbuilt function and find the answer in the cell given below:**
Annuity Immediate:	Present Value	0.12	1	10	0.1071428571	0	no	(₹5.65)
		0.12	500	10	0.1071428571	0	no	(₹2,825.11)
		0.12	1	10	0.1071428571	0	no	(₹5.65)
		0.12	500	10	0.1071428571	0	no	(₹2,825.11)
		0.12	1	10	0.1071428571	2	no	(₹7.09)
		0.12	500	10	0.1071428571	2	no	(₹3,543.82)
		0.12	1	10	0.1071428571	2	no	(₹7.09)
		0.12	500	10	0.1071428571	2	no	(₹3,543.82)
		0.12	1	10	0.1071428571	0	yes	8.333333333
		0.12	500	10	0.1071428571	0	yes	4166.666667
		0.12	1	10	0.1071428571	0	yes	8.333333333
		0.12	500	10	0.1071428571	0	yes	4166.666667
		0.12	1	10	0.1071428571	2	yes	10.45333333
		0.12	500	10	0.1071428571	2	yes	5226.666667
		0.12	1	10	0.1071428571	2	yes	10.45333333
		0.12	500	10	0.1071428571	2	yes	5226.666667
Annuity Due:	Present Value	0.12	1	10	0.1071428571	0	no	(₹6.33)
		0.12	500	10	0.1071428571	0	no	(₹3,164.12)
		0.12	1	10	0.1071428571	0	no	(₹6.33)
		0.12	500	10	0.1071428571	0	no	(₹3,164.12)
		0.12	1	10	0.1071428571	2	no	(₹7.94)
		0.12	500	10	0.1071428571	2	no	(₹3,969.08)
		0.12	1	10	0.1071428571	2	no	(₹7.94)
		0.12	500	10	0.1071428571	2	no	(₹3,969.08)
		0.12	1	10	0.1071428571	0	yes	9.333333333
		0.12	500	10	0.1071428571	0	yes	4666.666667
		0.12	1	10	0.1071428571	0	yes	9.333333333
		0.12	500	10	0.1071428571	0	yes	4666.666667
		0.12	1	10	0.1071428571	2	yes	11.70773333
		0.12	500	10	0.1071428571	2	yes	5853.866667
		0.12	1	10	0.1071428571	2	yes	11.70773333
		0.12	500	10	0.1071428571	2	yes	5853.866667

Type of Annuity	Find the following:	effective interest rate	Periodic equal payment	Number of Payments	effective discount rate	deferred period	Perpetuity (yes / no) (Use 1/i or O/i)	**Whereever possible use inbuilt function and find the answer in the cell given below:**
Annuity Immediate:	Present Value	0.12	1	10	0.1071428571	0	no	(₹5.65)
		0.12	500	10	0.1071428571	0	no	(₹2,825.11)
		0.12	1	10	0.1071428571	0	no	(₹5.65)
		0.12	500	10	0.1071428571	0	no	(₹2,825.11)
		0.12	1	10	0.1071428571	2	no	(₹7.09)
		0.12	500	10	0.1071428571	2	no	(₹3,543.82)
		0.12	1	10	0.1071428571	2	no	(₹7.09)
		0.12	500	10	0.1071428571	2	no	(₹3,543.82)
		0.12	1	10	0.1071428571	0	yes	8.333333333
		0.12	500	10	0.1071428571	0	yes	4166.666667
		0.12	1	10	0.1071428571	0	yes	8.333333333
		0.12	500	10	0.1071428571	0	yes	4166.666667
		0.12	1	10	0.1071428571	2	yes	10.45333333
		0.12	500	10	0.1071428571	2	yes	5226.666667
		0.12	1	10	0.1071428571	2	yes	10.45333333
		0.12	500	10	0.1071428571	2	yes	5226.666667
Annuity Due:	Present Value	0.12	1	10	0.1071428571	0	no	(₹6.33)
		0.12	500	10	0.1071428571	0	no	(₹3,164.12)
		0.12	1	10	0.1071428571	0	no	(₹6.33)
		0.12	500	10	0.1071428571	0	no	(₹3,164.12)
		0.12	1	10	0.1071428571	2	no	(₹7.94)
		0.12	500	10	0.1071428571	2	no	(₹3,969.08)
		0.12	1	10	0.1071428571	2	no	(₹7.94)
		0.12	500	10	0.1071428571	2	no	(₹3,969.08)
		0.12	1	10	0.1071428571	0	yes	9.333333333
		0.12	500	10	0.1071428571	0	yes	4666.666667
		0.12	1	10	0.1071428571	0	yes	9.333333333
		0.12	500	10	0.1071428571	0	yes	4666.666667
		0.12	1	10	0.1071428571	2	yes	11.70773333
		0.12	500	10	0.1071428571	2	yes	5853.866667
		0.12	1	10	0.1071428571	2	yes	11.70773333
		0.12	500	10	0.1071428571	2	yes	5853.866667