Paying Off a Loan

In this lesson, you will create a program to calculate the time it takes to pay off a loan. Your program will use lists to track the months, amount paid and balance. You will import theses lists to the graphs page. You will fit linear, piecewise, step and recursive function to model the data.

Objectives:

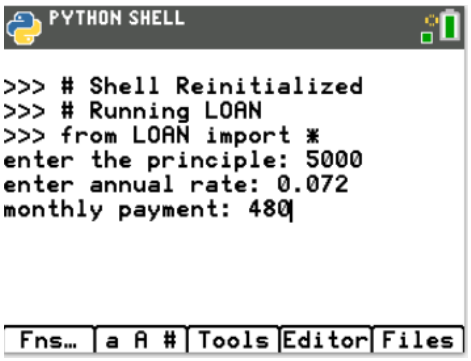
Programming Objectives:

- Use the input function and a variable to collect and store data from a user
- Use lists to store data
- Use a while loop to repeat interest and loan payments until the loan is paid off
- Export lists to the native TI system

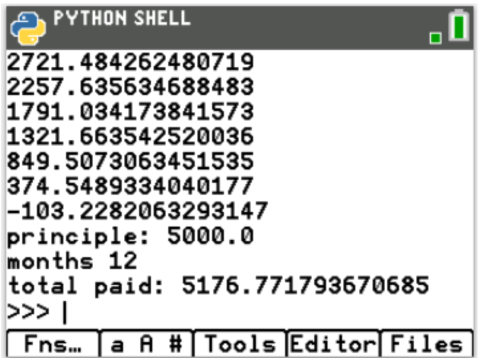
Math Objectives:

- Use linear, piecewise, step and recursive functions to model data

For this project, you will write a program that asks for the principle, annual interest rate and monthly payment. The output will display the approximate number of months it will take to pay off as well as the total amount paid.

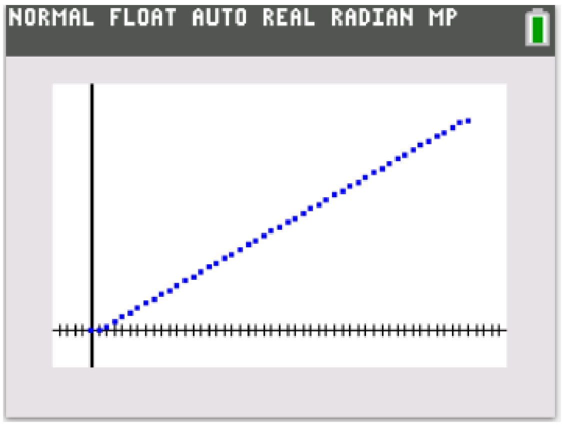


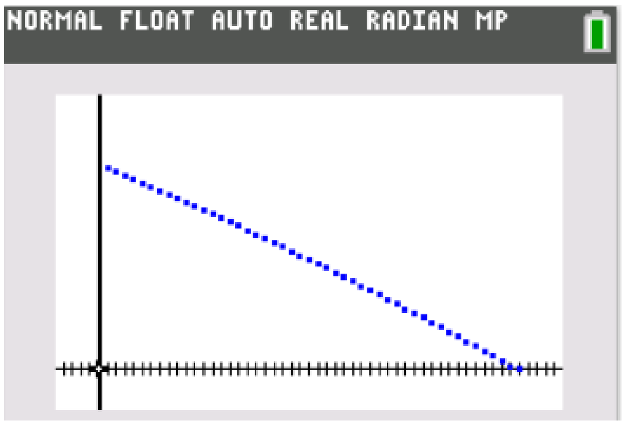
```
>>> # Shell Reinitialized
>>> # Running LOAN
>>> from LOAN import *
enter the principle: 5000
enter annual rate: 0.072
monthly payment: 480
```



```
2721.484262480719
2257.635634688483
1791.034173841573
1321.663542520036
849.5073063451535
374.5489334040177
-103.2282063293147
principle:  5000.0
months 12
total paid:  5176.771793670685
>>> |
```

The program will export the data for use in other native TI-84 CE Python documents.
You will add the data to a scatterplot on a graphs page.



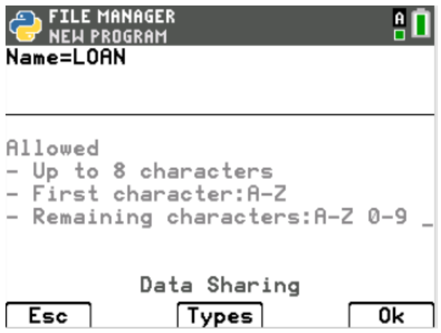


You will find different types of functions to model the data.

1. Start a new Python program. Name the project **LOAN**.

   This project will export lists of data.
   To export lists, the program will need the library ti_system.

   Select **Data Sharing** as the type.
   This will add the ti_system library.

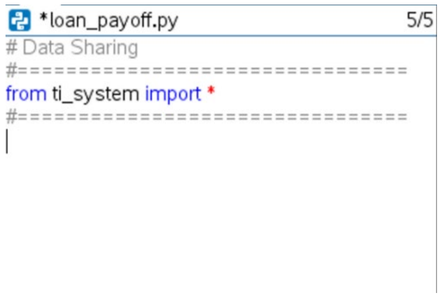


```
FILE MANAGER
NEW PROGRAM
Name=LOAN

Allowed
- Up to 8 characters
- First character:A-Z
- Remaining characters:A-Z 0-9 _

Data Sharing
```

2. Make sure the ti_system library was imported.
   If you forgot to select **Data Sharing**, you can go to
   Fns> Modul> TI System > from ti_system import*

   The comments, denoted by the # appear in light gray. These lines of code are optional. They do not affect how the program executes. Instead, they are notes to you, the programmer.

   Comments are used to describe sections of code. This can be useful when debugging or modifying a program later. They help a programmer remember the purpose for different sections of code. These comments tell you the coder; this library is used for data sharing.



```
*loan_payoff.py                         5/5
# Data Sharing
#================================
from ti_system import *
#================================
```

3. For this project, assume you want to purchase a car. You have found the perfect car, but need to take out a loan.

   What are some of the factors you need to consider when taking out a loan?

   What are some questions you might ask the bank?

4. Look at your list:
   Does it include the ***principle,*** *the initial amount borrowed?*
   How about an ***interest rate*** *or a* ***monthly payment****?*

   A bank loan charges interest on the amount borrowed. For example, if you borrow $10,000 with an annual interest rate of 8.9%. Each day, the loan balance increases using the following process.

Day	Initial Balance	Interest	New Balance
1	10000	1000*.089/365 = 0.23	10000 + 0.23 = 10000.23
2	10000.23	1000.23*.089/365 = 0.23	10000.23 + 0.23 = 10000.46
3	10000.46	1000.46*.089/365 = 0.23	10000.46 + 0.23 = 10000.69
..	…	..	..

   Each <u>day</u>, interest is added to the balance. This **increases** the balance.

   Each <u>month</u>, one payment is made. This will **decrease** the balance.

   This process of adding interest and making a payment are repeated until the balance is zero.

5. The first step to create your loan calculation program is to ask the user for the **principle**, the **annual interest rate** and the **monthly payment**.

   To request information, use **input("Enter question here").** You will store this information in a variable with a descriptive name.

   The statement:
   principle = input("Enter the principle amount: ")
   will ask the user the question then store the answer as characters.

   Use the command float()to store the principle as numbers instead of characters.

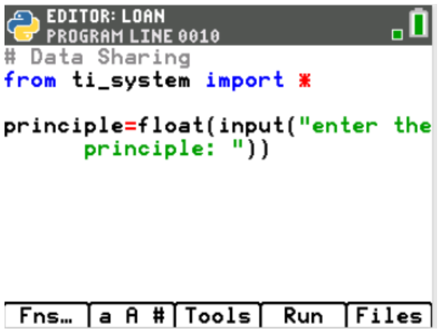
   principle = float(input("Enter the principle amount: "))
   The equation above asks the question and stores the result as a number.

   Fns> Type> float()
   Fns> I/O > input()

Write 3 lines of code.

1. Ask for the principle, store the value in **principle.**
2. Ask for the annual interest rate, store it as **arate**.
3. Ask for the monthly payment and store it as **payment.**

The picture to the right shows the first line of code.



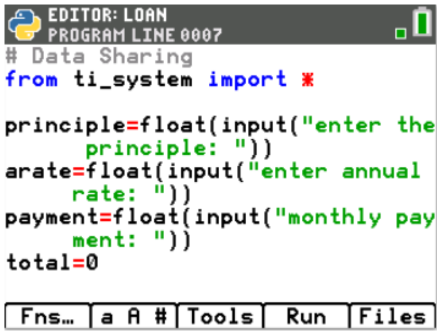
```
# Data Sharing
from ti_system import *

principle=float(input("enter the
      principle: "))
```

Tech Tip Using the [a A #] menu by pressing the [window] button might help when typing text.

6. The program will calculate the total amount paid at the end of the loan.

   Create and set a variable **total** to zero. This variable will store the total amount paid.

   total = 0

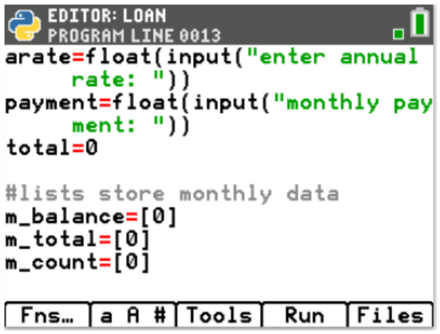


```
# Data Sharing
from ti_system import *

principle=float(input("enter the
      principle: "))
arate=float(input("enter annual
     rate: "))
payment=float(input("monthly pay
     ment: "))
total=0
```

7. Use lists to store each month's balance, the total paid as of the end of each month and the number of months.

   Add a comment to remind you the purpose of these lists:
   [a A #] menu, press the [window] button

   Create lists **m_balance**, **m_total**, and **m_count**.
   Make the initial value in each list 0.



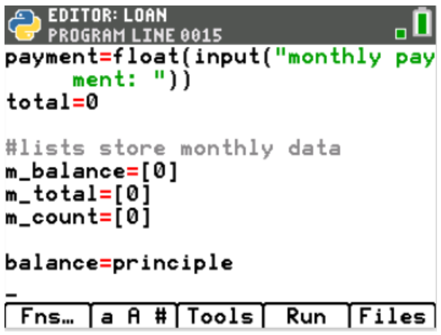
```
arate=float(input("enter annual
     rate: "))
payment=float(input("monthly pay
     ment: "))
total=0

#lists store monthly data
m_balance=[0]
m_total=[0]
m_count=[0]
```

8. Store the principle in a variable named balance.

   balance = principle

   Principle will stay a constant value, the initial amount borrowed.

   Balance will increase each day with interest. It will decrease each month with a payment.



```
payment=float(input("monthly pay
    ment: "))
total=0

#lists store monthly data
m_balance=[0]
m_total=[0]
m_count=[0]

balance=principle
_
```

9. Each month, approximately 30 days of interest are added to the balance. One payment is made.

   a. Use the variables **balance** and **arate**, write an expression that represents the new balance after one day of interest.

   b. Modify your expression to represent the balance after 30 days of interest.

   c. Modify your expression to remove one payment for the balance after 30 days of interest.

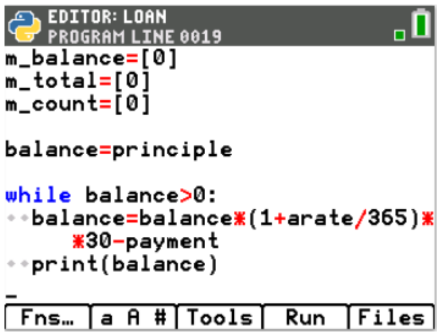
10. Do your expressions look similar to the ones below?

    a. balance + balance*arate/365
    or
    balance*(1 + arate/365)

    b. balance*(1 + arate/365)^{30}

    c. balance*(1 + arate/365)^{30} – payment

11. While the balance is greater than zero, the loan will increase by the interest and decrease by the payment.



```
EDITOR: LOAN
PROGRAM LINE 0019
m_balance=[0]
m_total=[0]
m_count=[0]

balance=principle

while balance>0:
◆◆balance=balance*(1+arate/365)*
    *30-payment
◆◆print(balance)
_
```

Fns… | a A # | Tools | Run | Files

Add a while statement to calculate and print the monthly balance.

```
while balance > 0:
    balance = balance*(1 + arate/365) **30 – payment
    print(balance)
```

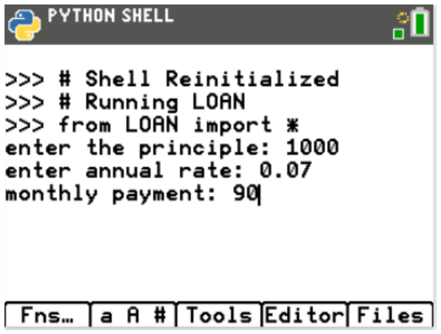
Fns> Ctl > while

Fns> I/O> print

*Note: In python, use ** to represent the ^ symbol for exponents.

The code inside the while loop is indented two spaces. In Python, the indent is used to denote the lines that should be included in the loop. Make sure both lines of code are indented two spaces. The TI-84 CE Python prints two diamonds, ◆◆, to visually show the indentation.

12. Let's check your code so far. In large projects such as this one, programmers execute the code often to check for errors.



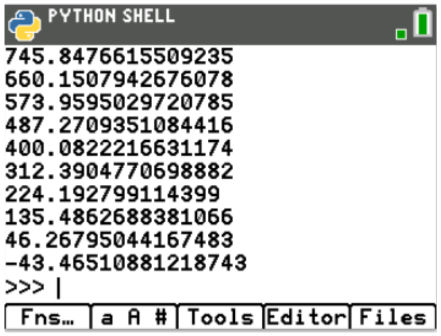
```
PYTHON SHELL

>>> # Shell Reinitialized
>>> # Running LOAN
>>> from LOAN import *
enter the principle: 1000
enter annual rate: 0.07
monthly payment: 90
```

Fns… | a A # | Tools | Editor | Files

Execute the program. Run

Set:
Principle = 1000
Annual Rate = 0.07 (Make sure you enter 0.07 for 7%)
Monthly Payment = 90

Make sure the output matches the output to the right.



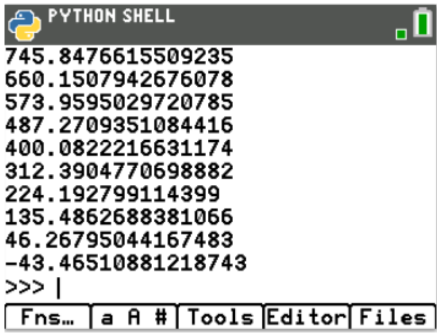
```
PYTHON SHELL
745.8476615509235
660.1507942676078
573.9595029720785
487.2709351084416
400.0822216631174
312.3904770698882
224.19279911 4399
135.4862688381066
46.26795044167483
-43.46510881218743
>>> |
```

Fns… | a A # | Tools | Editor | Files

***It's ok if you get stuck in an infinite loop** You just need to know how to get out:*

On the handheld calculator, hold down the On button for a few seconds.

13. Look at the example on the right. Notice the final balance is negative. What does a negative balance indicate?



```
745.8476615509235
660.1507942676078
573.9595029720785
487.2709351084416
400.0822216631174
312.3904770698882
224.192799114399
135.4862688381066
46.26795044167483
-43.46510881218743
>>> |
```

14. The negative balance indicates an overpayment on the last payment. The last payment was more than the balance owed.

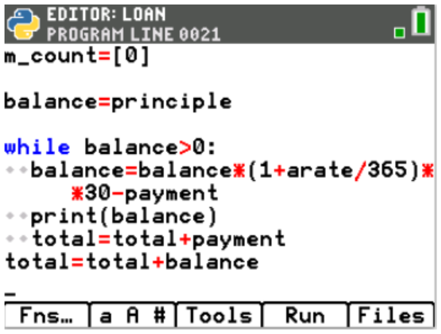
To find the total for the payments made, each time through the loop, the payment should be added to the total.

After the loop, the amount overpaid on the balance should be subtracted from the balance.

a. Add one line of code in the loop to add the payment to the total.

b. After the loop, add a line of code to remove the overpaid amount from the total.

15. Does your code look similar to the code on the right?



```
m_count=[0]

balance=principle

while balance>0:
  balance=balance*(1+arate/365)*
     *30-payment
  print(balance)
  total=total+payment
total=total+balance
```

In Python, the line

total = total + payment

can be written as

total += payment

The line

total = total + balance

can be written as

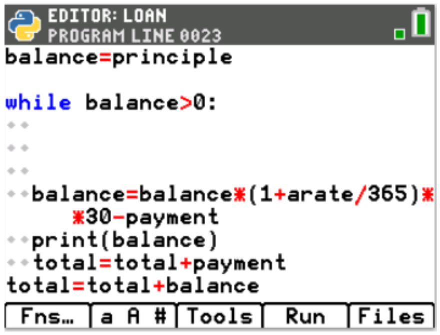
total += balance

The format used is up to the programmer's preference.

16. The program uses a loop to repeatedly add daily interest and make a monthly payment. The balance value is replaced each time through the loop. The total is also replaced each time through the loop.

    Use lists to keep track of each balance and each payment total.

    Add three blank lines at the beginning of the loop.

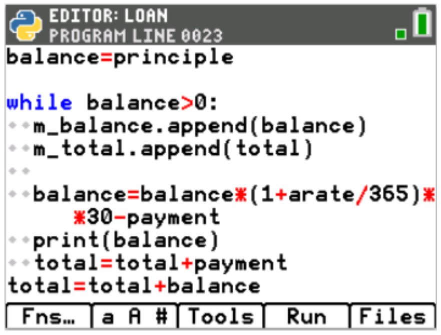


```
EDITOR: LOAN
PROGRAM LINE 0023
balance=principle

while balance>0:
  
  
  
  balance=balance*(1+arate/365)*
    *30-payment
  print(balance)
  total=total+payment
total=total+balance
Fns… | a A # | Tools | Run | Files
```

    Use the append() function to add each balance to a list of balances in the list m_balance.
    Add: m_balance.append(balance)

    Use the append() function to add each month's total .
    Add: m_balance.append(total)

    Fns> Lists> append

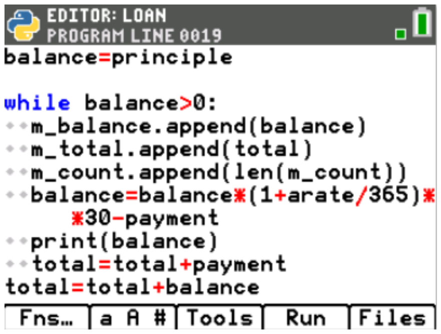


```
EDITOR: LOAN
PROGRAM LINE 0023
balance=principle

while balance>0:
  m_balance.append(balance)
  m_total.append(total)
  
  balance=balance*(1+arate/365)*
    *30-payment
  print(balance)
  total=total+payment
total=total+balance
Fns… | a A # | Tools | Run | Files
```

17. This list m_count will keep track of the number of months by keeping the values 0, 1, 2… Initially, the code stored 0 in the list m_count. That means the length of the list m_count is one.

    Each time through the loop, the line
    m_count.append(len(m_count))
    will add the length of the list to the list each time through the loop.

    Fns> > Lists> len

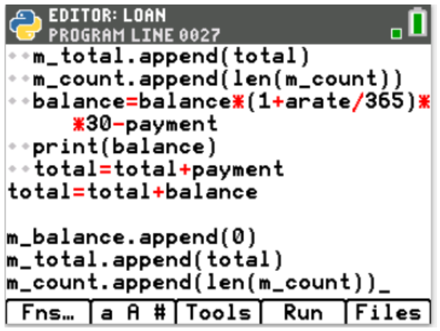


```
EDITOR: LOAN
PROGRAM LINE 0019
balance=principle

while balance>0:
  m_balance.append(balance)
  m_total.append(total)
  m_count.append(len(m_count))
  balance=balance*(1+arate/365)*
    *30-payment
  print(balance)
  total=total+payment
total=total+balance
Fns… | a A # | Tools | Run | Files
```

18. After the loop, the balance is zero.

    Add the 0 balance to the m_balance list.
    Add the final total amount paid to the m_total list.
    Add one more month to the m_count list.

    m_balance.append(0)
    m_total.append(total)
    m_count.append(len(m_count))



```
EDITOR: LOAN
PROGRAM LINE 0027
  m_total.append(total)
  m_count.append(len(m_count))
  balance=balance*(1+arate/365)*
    *30-payment
  print(balance)
  total=total+payment
total=total+balance

m_balance.append(0)
m_total.append(total)
m_count.append(len(m_count))_
Fns… | a A # | Tools | Run | Files
```

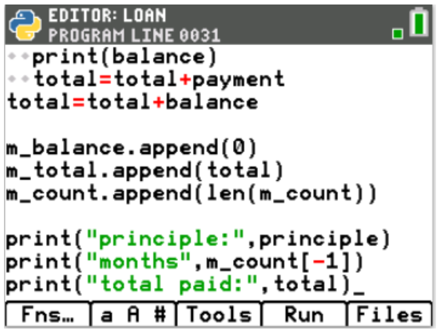
19. Add three print statements.

    One statement should print the initial amount borrowed.

    The second statement should print the number of months it took to pay back the loan. The last item in the list m_count contains the last month. To access the last month in the list, use m_count[-1].

    The last statement should print the total amount spent to pay back the loan.

    Fns> I/O> print()



```
··print(balance)
··total=total+payment
total=total+balance

m_balance.append(0)
m_total.append(total)
m_count.append(len(m_count))

print("principle:",principle)
print("months",m_count[-1])
print("total paid:",total)_
```

20. Use the program to answer the following questions.

    If you borrow $15,000 at an annual interest rate of 6% and make monthly payments of $300, how many months does it take to pay off the loan? What is the total amount paid back?

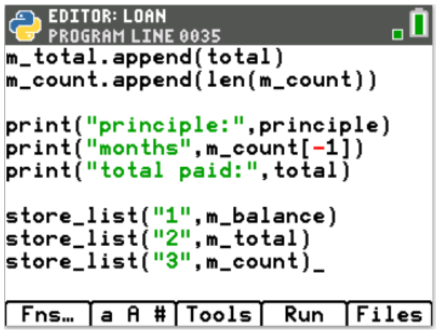
    If you borrow $15,000 at an annual interest rate of 6% and make monthly payments of $400, how many months does it take to pay off the loan? What is the total amount paid back?

21. Export the lists using the store_list() command from the ti_system library. This will allow you to use the lists on non-Python pages.

    Fns> Modul> TI System> store_list

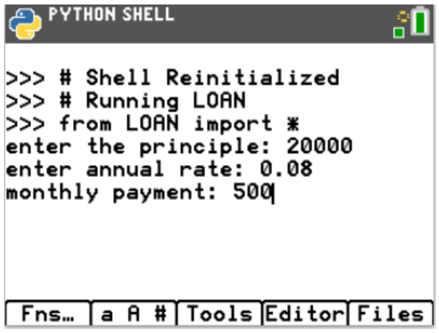
    Export all three lists.

    Store:
      m_balance in list1
      m_total in list2
      m_count in list3



```
m_total.append(total)
m_count.append(len(m_count))

print("principle:",principle)
print("months",m_count[-1])
print("total paid:",total)

store_list("1",m_balance)
store_list("2",m_total)
store_list("3",m_count)_
```

22. Execute the program for a $20,000 loan with 8% annual interest and a $500 monthly payment.



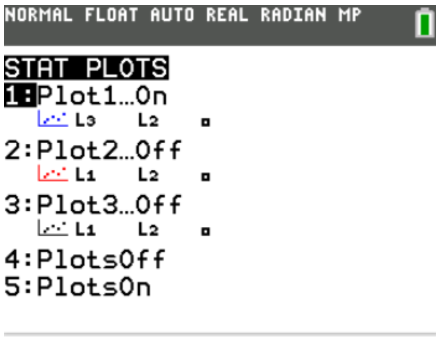
```
>>> # Shell Reinitialized
>>> # Running LOAN
>>> from LOAN import *
enter the principle: 20000
enter annual rate: 0.08
monthly payment: 500
```

Exit the Python Shell and Editor. [2nd] Quit [Ok]

23. Recall m_count is stored in L3 and m_total is stored in L2.

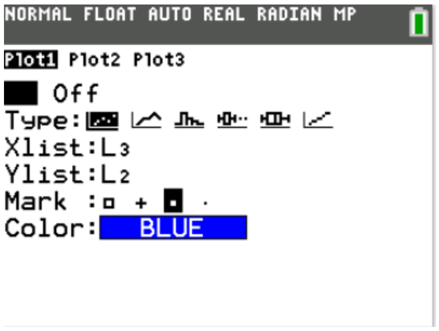
You will plot the month count as the independent variable on the x-axis and the month total as the dependent variable on the y-axis.

Check your Stats Plots setting. Turn only Plot 1 On.
[2nd] Statplot



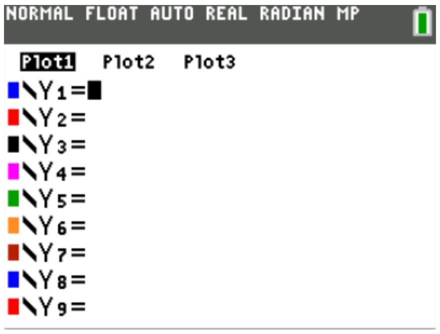
```
STAT PLOTS
1:Plot1…On
    L3   L2   ▫
2:Plot2…Off
    L1   L2   ▫
3:Plot3…Off
    L1   L2   ▫
4:PlotsOff
5:PlotsOn
```

For Plot 1:
Turn the Plot On
Choose scatter plot for the type.
Add L3 as the independent variable (xlist)
Add L2 as the dependent variable (yList)



```
Plot1 Plot2 Plot3
On Off
Type:
Xlist:L3
Ylist:L2
Mark :▫ + ▪ ·
Color: BLUE
```

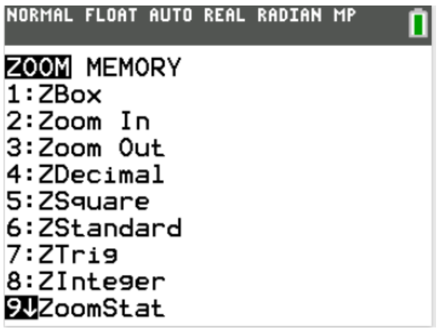
24. Go to y=.
Clear the functions.

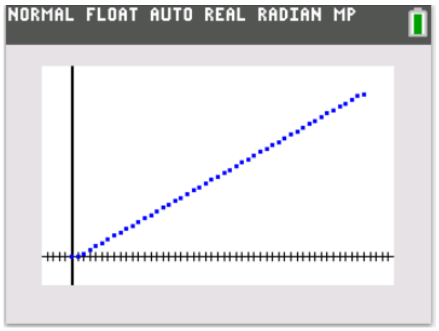


```
Plot1  Plot2  Plot3
\Y1=
\Y2=
\Y3=
\Y4=
\Y5=
\Y6=
\Y7=
\Y8=
\Y9=
```

25. Graph the data.

    The graph scale is set to the settings from your last graph. To fit the window to your data press [zoom]. Choose ZoomStat. This will automatically set the x-axis and y-axis based on your data.





26. Use the [trace] feature to complete the table below:

m_count	m_total
0	
1	
2	
3	
4	
5	

What is the total amount repaid?

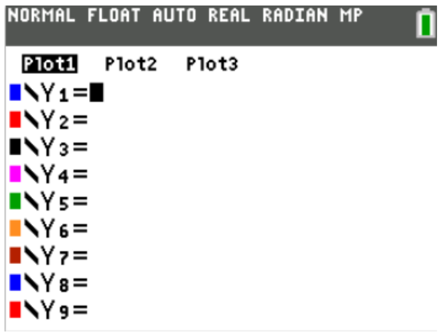
Identify the **domain** for this problem.

Identify the **range** for this problem.

27. Press [y=]

Menu> Graph Edit/Entry> Function.

a. Write a linear equation to model the data.
Enter your equation into the calculator.
Verify it matches the data in the graph



```
NORMAL FLOAT AUTO REAL RADIAN MP
 Plot1  Plot2  Plot3
■\Y1=■
■\Y2=
■\Y3=
■\Y4=
■\Y5=
■\Y6=
■\Y7=
■\Y8=
■\Y9=
```

b. Write the equation to a piecewise function that models the data.
Enter your equation into the calculator.
Verify it matches the data in the graph

c. Write a step function to model the data.
Enter your equation into the calculator.
Verify it matches the data in the graph

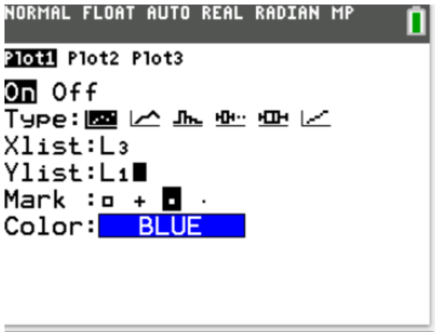
d. Write a recursive function to model the data.
(Change the graph from function to sequence.
[Mode] contains the Function/Seq option)
Enter your equation into the calculator.
Verify it matches the data in the graph

Look at the functions written above. Which function(s) would you pick to model the data? Explain why you picked this function(s).

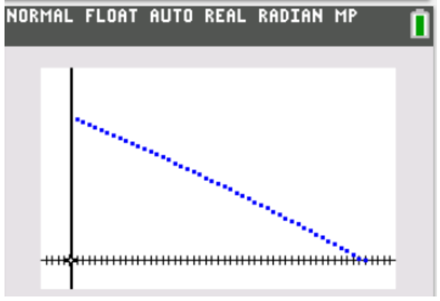
List one of the functions above you would not use to model the data. Explain why you picked this function.

28. Change the plotted ylist to m_balance.
Remember m_balance is stored in L1.

Adjust the window to fit the new data.
[zoom] ZoomStat



```
NORMAL FLOAT AUTO REAL RADIAN MP
Plot1 Plot2 Plot3
On Off
Type:
Xlist:L3
Ylist:L1
Mark :□ + ▪ ·
Color: BLUE
```



29. Use the [trace] feature to complete the table below:

m_count	m_balance
0	
1	
2	
3	
4	
5	

Identify the **domain** for this problem.

Identify the **range** for this problem.

Write a function to model the data.

Explain why you chose this function.