

**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 these 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

**Math Course Connections:**

This project could be used either in Algebra 1 or Algebra II to review linear, piecewise, recursive and stepwise functions.

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.

```

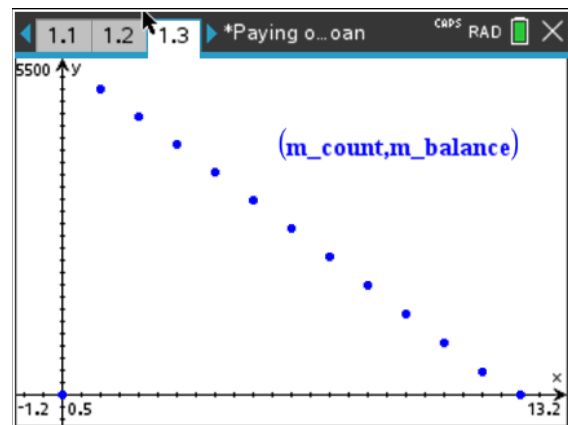
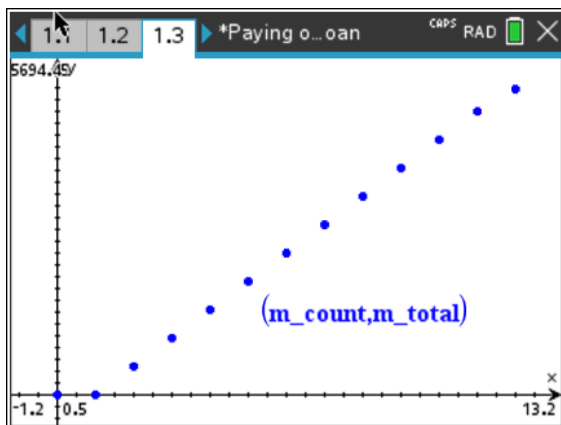
Python Shell 5/5
>>>#Running loan_payoff.py
>>>from loan_payoff import *
Enter the principle: 5000
Enter the annual interest rate: .072
Enter the monthly payment: 480
  
```

```

Python Shell 20/20
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-Nspire 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\_payoff**.

You are not allowed to use a space in the name of a project. Often, programmers will use an underscore to separate words variables and project names. They also use something called camel case, which does not include an underscore, instead it requires capitalizing the first letter of each word after the first word. In this case, loanPayoff could be used to name the file.

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.

2. Make sure the ti\_system library was imported.

If you forgot to select **Data Sharing**, you can go to

Menu> More Models> 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.*

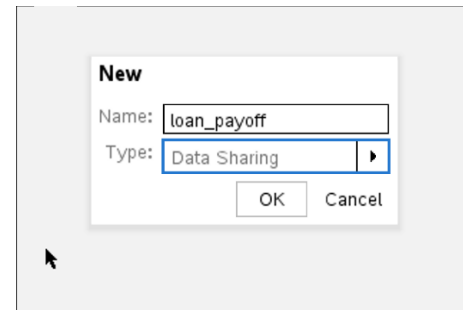
*Comments are used to describe sections of code. This can be useful when debugging or modifying a program later. They help a programmer remember the purpose for different sections of code. These comments tell you the coder; this library is used for data sharing.*

**Teacher Tip:** If students failed to initially include the ti\_system library, step 2 will not add the three lines of comments. Students may add these lines or choose to leave them out. They do not affect the code's execution.

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?*



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

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 day, interest is added to the balance. This **increases** the balance.

Each month, 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.

```
Menu> Built-ins> Type> float()
Menu> Built-ins> I/O > input()
```



# Math Explorations with Python

TI-NSPIRE™ CX II TECHNOLOGY

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.

```

1.1 *loan_payoff.py 6/6
# Data Sharing
#-----
from ti_system import *
#-----
principle=float(input("Enter the principle: "))
|

```

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

```

*loan_payoff.py 8/8
# Data Sharing
#-----
from ti_system import *
#-----
principle=float(input("Enter the principle: "))
arate = float(input("Enter the annual interest rate:"))
payment=float(input("Enter the monthly payment:"))
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:

Menu> Edit> Comment

or

ctrl → t

Create lists **m\_balance**, **m\_total**, and **m\_count**.

Make the initial value in each list 0.

```

1.1 1.2 *Doc RAD 14/14
*loan_payoff.py
#-----
principle=float(input("Enter the principle: "))
arate = float(input("Enter the annual interest rate:"))
payment=float(input("Enter the monthly payment:"))
total = 0

#lists to store data for each month
m_balance = [0]
m_total = [0]
m_count = [0]
|

```



# Math Explorations with Python

TI-NSPIRE™ CX II TECHNOLOGY

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.

9. Each month, approximately 30 days of interest are added to the balance. One payment is made.
- Use the variables **balance** and **arate**, write an expression that represents the new balance after one day of interest.
  - Modify your expression to represent the balance after 30 days of interest.
  - 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?

- $balance + balance * arate / 365$   
or  
 $balance * (1 + arate / 365)$
- $balance * (1 + arate / 365)^{30}$
- $balance * (1 + arate / 365)^{30} - payment$

## PAYING OFF A LOAN

### TEACHER NOTES

```
*loan_payoff.py 16/16
arate = float(input("Enter the annual interest rate:
payment=float(input("Enter the monthly payment
total = 0

#lists to store data for each month
m_balance = [0]
m_total = [0]
m_count = [0]

balance = principle
```



# Math Explorations with Python

TI-NSPIRE™ CX II TECHNOLOGY

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

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

while balance > 0:

```
    balance = balance*(1 + arate/365) **30 – payment
    print(balance)
```

Menu> Built-ins> Control > while

Menu> Built-ins> 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-Nspire 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.

Execute the program. **Ctrl> r**

**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.*

**\*\*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/Home button for a few seconds.
- On the TI-Nspire™ software on Windows®, press F12 then enter. On the TI-Nspire™ software on a Mac®, press F5 then enter

## PAYING OFF A LOAN

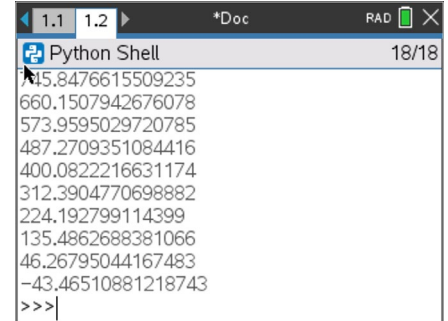
### TEACHER NOTES

```
*loan_payoff.py 19/19
#lists to store data for each month
m_balance = [0]
m_total = [0]
m_count = [0]
}
balance = principle
while balance > 0:
♦♦balance=balance*(1+arate/365)**30-payment
♦♦print(balance)
```

```
Python Shell 5/5
>>>#Running loan_payoff.py
>>>from loan_payoff import *
Enter the principle: 1000
Enter the annual interest rate: .07
Enter the monthly payment: 90
```

```
Python Shell 18/18
15.8476615509235
660.1507942676078
573.9595029720785
487.2709351084416
400.0822216631174
312.3904770698882
224.192799114399
135.4862688381066
46.26795044167483
-43.46510881218743
>>>
```

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



```
Python Shell 18/18
15.8476615509235
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.

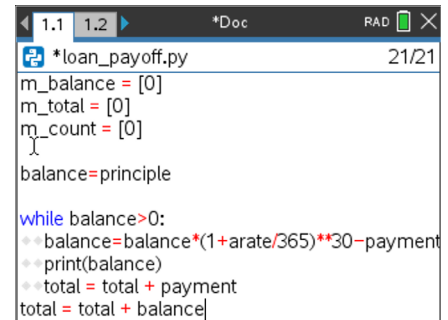
- Add one line of code in the loop to add the payment to the total.
- After the loop, add a line of code to remove the overpaid amount from the total.

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

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.

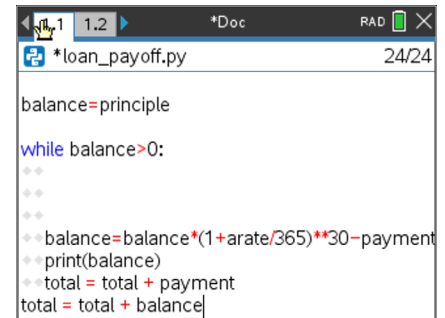


```
*loan_payoff.py 21/21
m_balance = [0]
m_total = [0]
m_count = [0]
}
balance=principle
while balance>0:
    * balance=balance*(1+arate/365)**30-payment
    * print(balance)
    * total = total + payment
total = total + balance|
```

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.



```

1.1 1.2 *Doc RAD 24/24
*loan_payoff.py
balance=principle
while balance>0:
    balance=balance*(1+arate/365)**30-payment
    print(balance)
    total = total + payment
    total = total + balance
    
```

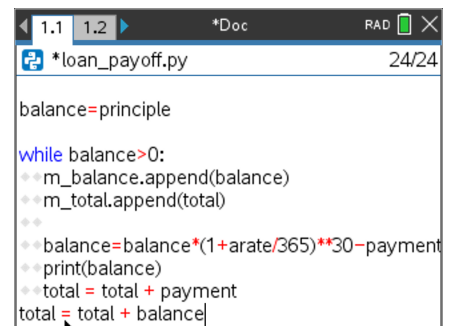
Use the **append()** function to add each balance to a list of balances in the list `m_balance`.

Add: `m_balance.append(balance)`

Use the `append()` function to add each month's total .

Add: `m_balance.append(total)`

**Menu> Built-ins> Lists> append**



```

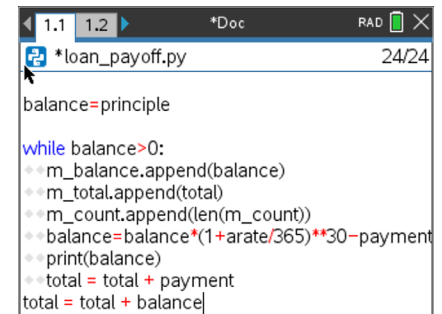
1.1 1.2 *Doc RAD 24/24
*loan_payoff.py
balance=principle
while balance>0:
    m_balance.append(balance)
    m_total.append(total)
    balance=balance*(1+arate/365)**30-payment
    print(balance)
    total = total + payment
    total = total + balance
    
```

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.



```

1.1 1.2 *Doc RAD 24/24
*loan_payoff.py
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
    
```

**Menu> Built-ins> Lists> len**



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))
```

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.

**Menu> Built-ins> I/O> print()**

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?*

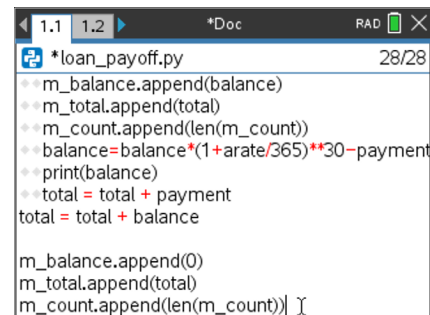
21. Export the lists using the **store\_list()** command from the **ti\_system** library. This will allow you to use the lists on non-Python pages.

**Menu> More Modules> TI System> store\_list**

Export all three lists.

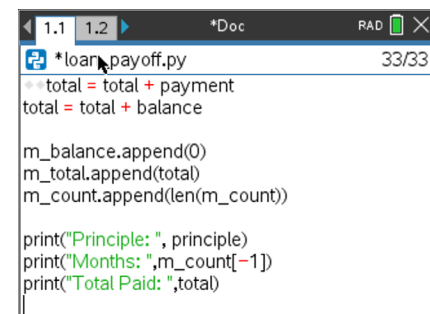
## PAYING OFF A LOAN

### TEACHER NOTES



```
*loan_payoff.py 28/28
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

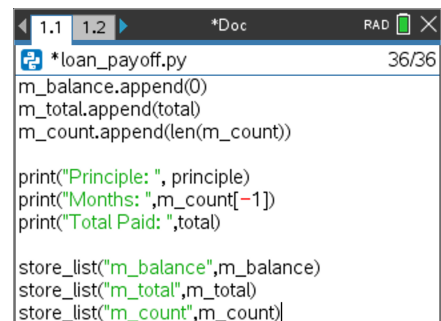
m_balance.append(0)
m_total.append(total)
m_count.append(len(m_count))
```



```
*loan_payoff.py 33/33
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)
```



```
*loan_payoff.py 36/36
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)

store_list("m_balance",m_balance)
store_list("m_total",m_total)
store_list("m_count",m_count)
```



# Math Explorations with Python

TI-NSPIRE™ CX II TECHNOLOGY

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

Insert a Graph page. (ctrl i)

Add the monthly count,  $m\_count$ , as the dependent variable.

Add the total amount paid,  $m\_total$ , as the independent variable.

Menu> Graph Entry/Edit> scatterplot

Menu> Window/Zoom> Zoom-Data

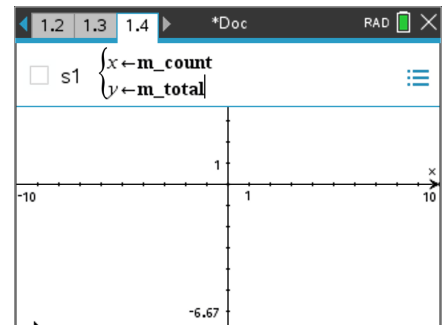
## PAYING OFF A LOAN

### TEACHER NOTES

```

Python Shell 5/5
>>>#Running loan_payoff.py
>>>from loan_payoff import *
Enter the principle: 20000
Enter the annual interest rate: .08
Enter the monthly payment: 500

```



23. Use the graph 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.

24. Set the graph type back to function.

**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

**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.**

**Menu> Graph Edit/Entry> Sequence**

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.



# Math Explorations with Python

TI-NSPIRE™ CX II TECHNOLOGY

25. Insert a new graphs page. (ctrl i)

Add the monthly count,  $m\_count$ , as the dependent variable.

Add the total amount paid,  $m\_balance$ , as the independent variable.

**Menu**> **Graph Entry/Edit**> **scatterplot**

**Menu**> **Window/Zoom**> **Zoom-Data**

26. Use the **graph 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.

## PAYING OFF A LOAN

### TEACHER NOTES

