Unit 5: Recursion & Lists — Activity 1: Recursion

In this lesson, you will define functions using explicit and recursive notation and use the functions to evaluate expressions.

Objectives:

- Define a function using explicit notation
- Define a function using recursion
- Use **return** to send values to another function

Functions play a big role in Python programming and in mathematics. Functions can be used to generate many different kinds of values and functions serve as Python subroutines which help to break a complex process into smaller, manageable parts.

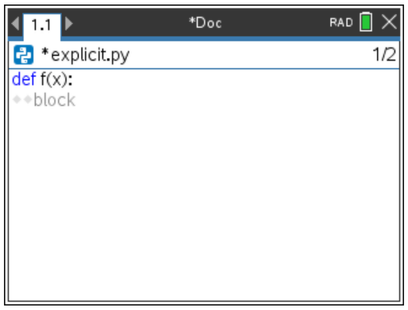
Functions can be defined using **explicit** or **recursive** notation.

Explicit Notation

1. Write a program that lets a user enter a number for *x* and the program will use the value to evaluate the explicit function *f*(*x*) = 3*x* – 1.

   First, define a function using **Menu > Built-Ins > Function > def function():**.

   Note: *f*(*x*) = 3*x* – 1 is an **explicit function** because it allows you to find the value of *f*(*x*) based on the value of *x*.



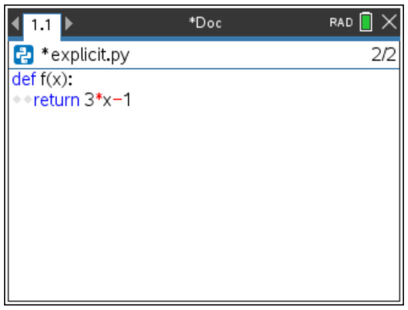
```
1.1  *Doc  RAD
*explicit.py  1/2
def f(x):
  block
```

2. Change the block to:

   return 3*x – 1

   return is found in **Menu > Built-Ins > Function**.

   return is used at the end of a function to send a value back to the calling statement.

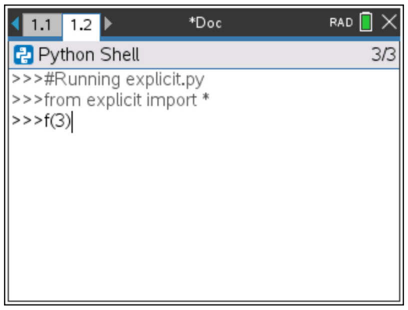


```
1.1  *Doc  RAD
*explicit.py  2/2
def f(x):
  return 3*x−1
```

3. Run the program as currently written. What happens?

   The function is now defined but it will not be executed until it is called from the main program or from the Python shell.

   In the Python shell, enter **f(3)**. Predict the output before you press **enter**. Test your prediction.



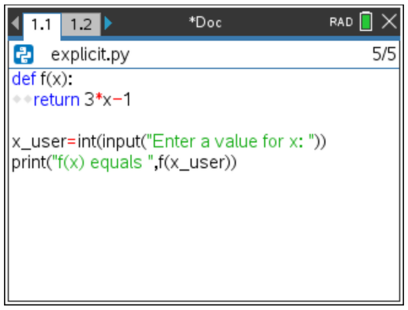
```
1.1  1.2  *Doc  RAD
Python Shell  3/3
>>>#Running explicit.py
>>>from explicit import *
>>>f(3)
```

How could you enhance your program so that when it runs it asks the user to enter a value for *x*, then displays a message showing the value of *f*(*x*) for that *x* value?

Code out your solution to this enhancement, then turn to the next page to see a sample solution.

4. You could use an **input()** statement in the main part of your program to allow the user to enter a number for *x*. Note that we've used **x_user** as the variable name for clarity.

   We've also used a **print** function to display a message and the value of ***f*(*x*_user)** back to the user. What is the code ***f*(*x*_user)** doing here?



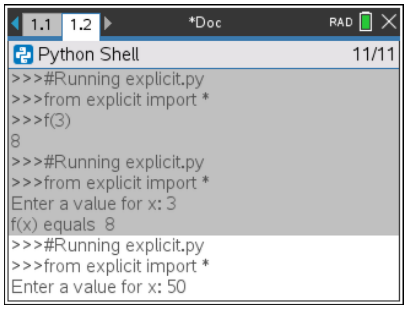
```
1.1  1.2   *Doc   RAD
explicit.py                        5/5
def f(x):
  return 3*x-1

x_user=int(input("Enter a value for x: "))
print("f(x) equals ",f(x_user))
```

5. Run the program to test how it works.

   When prompted, enter a value of 3 for *x*. Predict the output, then test your prediction.

   What should the output be if you enter a value of 50 for *x*? Test your prediction.



```
1.1  1.2   *Doc   RAD
Python Shell                      11/11
>>>#Running explicit.py
>>>from explicit import *
>>>f(3)
8
>>>#Running explicit.py
>>>from explicit import *
Enter a value for x: 3
f(x) equals  8
>>>#Running explicit.py
>>>from explicit import *
Enter a value for x: 50
```

Recursive Notation

1. Consider a sequence of numbers where the first number is 3, the second number is twice the value of the first number, the third number is twice the value of the second number, etc.. Write the first five terms in this sequence.

   The following **recursive formula** can be used to represent this sequence:

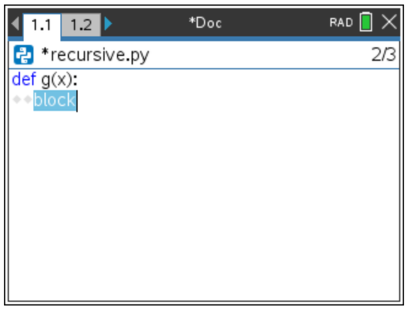
   ***g*(0) = 3** the first term

   g*(*n*) = 2*g*(*n*-1)** a rule for finding any term based on the term before it

2. Write a program that lets a user enter a number for *x* and the program will use the value to evaluate the recursive function *g*(*x*) shown above.

   First, define a function using **Menu > Built-Ins > Function > def function():**.

   Note: *g*(*x*) is a **recursive function** because it defines a function value (i.e., *g*(8)) based on a previous function value (i.e., 2**g*(8-1) = 2**g*(7)).



```
1.1  1.2   *Doc   RAD
*recursive.py                      2/3
def g(x):
  block
```

3. Change the block to:

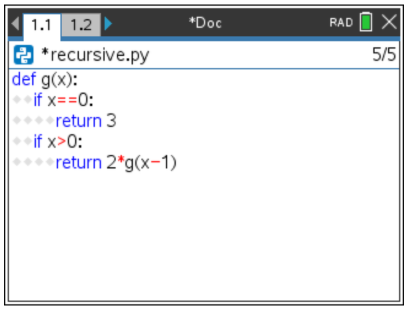
   if x==0:

   return 3

   if x>0:

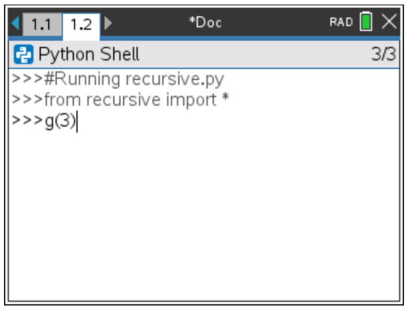
   return 2*g(x-1)

   How do you think this block of code works?



```
1.1  1.2   *Doc   RAD
*recursive.py                      5/5
def g(x):
  if x==0:
    return 3
  if x>0:
    return 2*g(x-1)
```

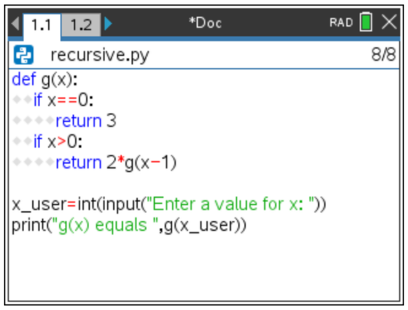
4. Run the program. In the Python shell, enter **g(3)**. Predict the output before you press **enter**. Test your prediction.



```
>>>#Running recursive.py
>>>from recursive import *
>>>g(3)
```

5. Enhance your program so that it asks the user to enter a value for *x*, then displays a message showing the value of *g*(*x*) for that *x* value.

   Test your program for numerous values of *x*.



```
def g(x):
  if x==0:
    return 3
  if x>0:
    return 2*g(x-1)

x_user=int(input("Enter a value for x: "))
print("g(x) equals ",g(x_user))
```

Comparing Explicit Notation & Recursive Notation

How would the first program evaluate *f*(2)?

How would the second program evaluate *g*(2)?

How are these methods similar? How are they different?